Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Aplicación de Modelos YOLO-NAS y YOLOv8 en Sistemas Seguimiento de Múltiples Objetos

Autor(a): Yiyuan Huang
Tutor(a): Javier de Lope Asiaín

Madrid, Junio 2024

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Máster Universitario en Ingeniería Informática

Título: Aplicación de Modelos YOLO-NAS y YOLOv8 en Sistemas Seguimiento de Múltiples Objetos

Junio 2024

Autor(a): Yiyuan Huang
Tutor(a): Javier de Lope Asiaín
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

La visión computacional, también conocida como visión artificial o visión por computadora, es una disciplina científica que abarca métodos para adquirir, procesar, analizar y comprender imágenes del mundo real con el fin de generar información numérica o simbólica que pueda ser tratada por un ordenador. Su objetivo principal es replicar la capacidad del sistema visual humano para comprender y extraer información útil del entorno.

Mediante el uso de algoritmos y técnicas de Deep Learning, como las redes neuronales convolucionales, los sistemas de visión por computadora analizan grandes volúmenes de datos visuales para identificar patrones y crear modelos capaces de reconocer objetos. Este enfoque supervisado permite a los modelos aprender y realizar tareas específicas de reconocimiento visual con alta precisión.

Las tareas típicas de la visión computacional incluyen el reconocimiento de imágenes y videos, la clasificación de imágenes, la detección de objetos, la identificación de objetos y el seguimiento de objetos en videos.

Una de las limitaciones de la visión computacional en tareas de detección de objetos es su incapacidad para relacionar detecciones a lo largo de una secuencia de imágenes o para resolver el problema de estimar la ubicación de un objeto cuando este se encuentra ocluido. En este contexto, el seguimiento de objetos juega un papel crucial, ya que permite mantener una consistencia temporal y predecir la trayectoria del objeto. El funcionamiento de estas tecnologías se basa en teorías de estimación que combinan información de detección con modelos de predicción para mejorar la estimación.

Para avanzar en este campo, el presente proyecto estudia diversos sistemas, explorando su arquitectura y funcionamiento, y los combina con distintos modelos de detección de última generación como YOLOv8 y YOLO-NAS. Se evaluará el desempeño de detección y seguimiento utilizando diferentes métricas de calidad para analizar el comportamiento en tareas de seguimiento.

En este trabajo se emplean diversos conjuntos de datos públicos proporcionados por distintas fuentes. Estos conjuntos incluyen imágenes de diversos escenarios, junto con anotaciones de los objetos presentes en cada imagen. Estos datos se utilizan para evaluar el rendimiento de los detectores en los sistemas de seguimiento, verificando si se logra mejorar los resultados en la detección y el seguimiento de objetos.

En este estudio se centra en el seguimiento de múltiples objetos y la mejora de la detección en la visión computacional, utilizando los modelos de detección de

última generación YOLO-NAS y YOLOv8. Además, sirve como una guía para aquellos interesados en este ámbito, ofreciendo una perspectiva amplia sobre los desafíos y las soluciones actuales en la materia.

Abstract

Computer vision, also known as artificial vision or machine vision, is a scientific discipline that encompasses methods for acquiring, processing, analyzing, and understanding real-world images to generate numerical or symbolic information that can be processed by a computer. Its primary objective is to replicate the human visual system's ability to understand and extract useful information from the environment.

Using algorithms and deep learning techniques, such as convolutional neural networks, computer vision systems analyze large volumes of visual data to identify patterns and create models capable of recognizing objects. This supervised approach allows models to learn and perform specific visual recognition tasks with high precision.

Typical tasks in computer vision include image and video recognition, image classification, object detection, object identification, and object tracking in videos.

One of the limitations of computer vision in object detection tasks is its inability to relate detections across a sequence of images or to solve the problem of estimating an object's location when it is occluded. In this context, object tracking plays a crucial role, as it allows maintaining temporal consistency and predicting the object's trajectory. The functioning of these technologies is based on estimation theories that combine detection information with prediction models to improve estimation.

To advance in this field, the present project studies various systems, exploring their architecture and functioning, and combines them with different state-of-the-art detection models like YOLOv8 and YOLO-NAS. The detection and tracking performance will be evaluated using different quality metrics to analyze behavior in tracking tasks.

This work employs various public datasets provided by different sources. These datasets include images of various scenarios, along with annotations of the objects present in each image. These data are used to evaluate the performance of detectors in tracking systems, verifying whether detection and tracking results are improved.

This study focuses on multi-object tracking and the improvement of detection in computer vision, using state-of-the-art detection models YOLO-NAS and YOLOv8. Additionally, it serves as a guide for those interested in this field, offering a broad perspective on the current challenges and solutions in the subject matter.

Acronimos

- **IA**: *Artificial Intelligence*.
- **ChatGPT**: *Chat Generative Pre-trained Transformer*.
- **GenAI**: *Generate AI*.
- **GANs**: *Generative Adversarial Networks*.
- **VAEs**: *Variational Autoencoders*.
- **GPT**: *Generative Pre-trained Transformer*.
- **AR**: *Augmented Reality*.
- **VR**: *Virtual Reality*.
- **MOT**: *Multiple Object Tracking*.
- **CNN**: *Convolutional Neural Network*.
- **DeepOCSORT**: *Deep Object Tracking with DeepOCSORT*.
- **HybridSORT**: *Hybrid Object Tracking*.
- **StrongSORT**: *Robust Object Tracking*.
- **SORT**: *Simple Online and Realtime Tracking*.
- **MTT**: *Multiple Target Tracking*.
- **TBD**: *Detection-Based Multi-Target Tracking*.
- **DFT**: *Discrete Fourier Transform*.
- **MHT**: *Multiple Hypothesis Tracking*.
- **MOTChallenge**: *Multiple Object Tracking Challenge*.
- **SOT**: *Single Object Tracking*.
- **CNN**: *Convolutional Neural Network*.
- **RNN**: *Recurrent Neural Network*.
- **GCN**: *Graph Convolutional Network*.
- **CF**: *Collaborative Filtering*.
- **FFT**: *Fast Fourier Transform*.
- **MDNet**: *Multi-Domain Deep Neural Network*.

- **CACF**: *Context-Aware Collaborative Filtering.*
- **KYS**: *Knowledge Youth Synthesis.*
- **PGNET**: *Progress Generation Network.*
- **D3S**: *Safety and Security Detection and Tracking System.*
- **DSAR-CF**: *Collaborative Filtering with Regularization and Anomaly Detection.*
- **CSR-DCF**: *Correlation-based Tracking Filter with Feature Detection.*
- **RASNet**: *Risk Analysis Neural Network.*
- **LMCF**: *Collaborative Filtering with Length Modeling.*
- **PGNet**: *Progressive Generative Network.*
- **DMF**: *Distributed Matrix Factorization.*
- **JDE**: *Joint Detection and Embedding.*
- **IoU**: *Intersection over Union.*
- **Re-ID**: *Person Re-identification.*
- **CMC**: *Camera Motion Compensation.*
- **BoTSORT**: *Trajectory-based Object Tracking.*
- **HOTA**: *Higher Order Tracking Accuracy.*
- **MOTA**: *Multiple Object Tracking Accuracy.*
- **IDF1**: *Identity Flow F1 Score.*
- **ORU**: *Observation-Centric Update.*
- **OCM**: *Observation-Centric Momentum.*
- **OCR**: *Observation-Centric Retrieval.*
- **CPU**: *Central Processing Unit.*
- **GPU**: *Graphics Processing Unit.*
- **TCM**: *Trajectory Confidence Modeling.*
- **HMIoU**: *Height-Modulated IoU.*
- **ROCM**: *Robust Observation Centroid Momentum.*
- **BYTE**: *Binary Term.*
- **AFLink**: *Appearance-Free Link Model.*
- **GSI**: *Gaussian Smooth Interpolation.*
- **JDE**: *Joint Detection and Embedding.*
- **EG**: *Entity Generation.*
- **KITTI**: *Karlsruhe Institute of Technology and Toyota Technological Institute.*
- **RPN**: *Region Proposal Network.*

- **MOT15**: *Multi-Object Tracking Challenge 2015*.
- **MOT16**: *Multi-Object Tracking Challenge 2016*.
- **MOT17**: *Multi-Object Tracking Challenge 2017*.
- **MOT20**: *Multi-Object Tracking Challenge 2020*.
- **VOT-2015**: *Visual Object Tracking 2015*.
- **JPDA**: *Joint Probabilistic Data Association*.
- **FPS**: *Frames Per Second*.
- **YOLO**: *You Only Look Once*.
- **SSD**: *Single Shot MultiBox Detector*.
- **VGG-16**: *Visual Geometry Group*.
- **NMS**: *Non-Maximum Suppression*.
- **NAS**: *Neural Architecture Search*.
- **AutoML**: *Automated Machine Learning*.
- **DET**: *Detection Evaluation Toolbox*.
- **GT**: *Ground Truth*.
- **MOTP**: *Multiple Object Tracking Precision*.
- **MACF**: *Machine First Delivery Model*.
- **JSON**: *JavaScript Object Notation*.
- **COCO**: *Common Objects in Context*.
- **APIs**: *Application Programming Interfaces*.
- **RAM**: *Random Access Memory*.
- **CLEAR MOT**: *Classification of Events, Activities and Relationships for Multi-Object Tracking*.
- **ONNX**: *Open Neural Network Exchange*.
- **FP**: *False Positives*.
- **FN**: *False Negatives*.
- **TP**: *True Positives*.
- **FPA**: *False Positive Associations*.
- **FNA**: *False Negative Associations*.
- **TPA**: *True Positive Associations*.
- **IDSW**: *Identity Switches*.
- **IDFP**: *Identification False Positives*.
- **IDFN**: *Identification False Negatives*.

- **IDTP**: *Identification True Positives*.
- **DetA** : *Detection Accuracy*.
- **AssA**: *Association Accuracy*.

Tabla de contenidos

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

Hoy en día, la inteligencia artificial se ha convertido en una de las tecnologías principales, utilizada ampliamente en la vida cotidiana de diversas industrias. Con el desarrollo y maduración continuos de la tecnología, el impacto y el alcance de aplicación de la inteligencia artificial también están expandiéndose constantemente. Por ejemplo, en términos de avances tecnológicos: en los últimos años, el campo de la inteligencia artificial ha logrado un progreso tecnológico significativo, especialmente en áreas como el aprendizaje automático, el aprendizaje profundo y el procesamiento del lenguaje natural (como ChatGPT). Estos avances han hecho que la IA sea capaz de manejar problemas complejos que antes eran difíciles de resolver. El aumento de la capacidad de cómputo, impulsado por la proliferación de hardware como la computación en la nube y las GPU, ha incrementado enormemente la capacidad de procesamiento, haciendo posible el entrenamiento de modelos complejos de IA. El desarrollo de la IA Generativa (GenAI) se ha convertido ahora en una tendencia principal en el avance tecnológico. Con el rápido progreso de las tecnologías de aprendizaje profundo, especialmente las redes generativas adversarias (GANs), los autoencoders variacionales (VAEs) y los modelos de lenguaje preentrenados a gran escala (como la serie GPT de OpenAI), la IA generativa ha demostrado su poderosa capacidad y amplias perspectivas de aplicación en múltiples campos.

La visión por computadora es una rama importante de la inteligencia artificial que se centra en dotar a las computadoras de la capacidad de procesar y comprender la información visual, de manera similar a cómo los humanos ven e interpretan el mundo a través de sus ojos. Este campo combina varias áreas técnicas, como el procesamiento de imágenes, el reconocimiento de patrones y el aprendizaje automático, con el fin de que las computadoras puedan reconocer elementos visuales como objetos, escenas, comportamientos, textos, etc., a través de imágenes o evaluaciones temporales.

El alcance de aplicación de la visión por computadora es amplio e incluye diversos aspectos. Entre ellos se encuentra el reconocimiento de imágenes, útil para identificar objetos o rostros específicos en una imagen, con aplicaciones extendidas en la vigilancia de seguridad, redes sociales y comercio minorista. Asimismo, la compresión de escenas permite analizar y comprimir el contenido y la estructura de una escena en imágenes o videos, con aplicaciones en la navegación y los vehículos autónomos. Otros aspectos importantes son la Realidad Aumentada (AR) y la Realidad Virtual (VR), que utilizan la tecnología de visión por computadora para mejorar la

experiencia del usuario en el mundo real o crear entornos completamente virtuales. Además de estos, se destacan áreas como el análisis de imágenes médicas, la automatización industrial y el comercio minorista, entre otros.

El seguimiento de múltiples objetos (Multi-Object Tracking, MOT) es una dirección de investigación importante en el campo de la visión por computadora, cuyo objetivo es rastrear la posición y la trayectoria de movimiento de múltiples objetivos móviles simultáneamente a partir de una secuencia de video. La tecnología MOT es capaz de identificar cada objetivo que aparece en el video y mantener un seguimiento continuo de estos objetivos a lo largo de toda la secuencia de video. Esta tarea implica varios aspectos como la detección de objetivos, el mantenimiento de la identidad, la estimación de posición y la predicción del movimiento.

1.1. Historia y Evolución del MOT

La historia del seguimiento de múltiples objetivos está estrechamente ligada al desarrollo de la inteligencia artificial, la visión por computadora y la tecnología de seguimiento de objetos en general. A continuación, se ofrece una descripción detallada de la historia del seguimiento de múltiples objetivos, combinando los orígenes de la Inteligencia Artificial y la Visión por Computadora y su evolución.

Los orígenes de la Inteligencia Artificial se sitúan en la década de 1950, un periodo de exploración inicial por parte de la comunidad científica para intentar que las máquinas imitaran el comportamiento humano inteligente. Durante esta época, se introdujo una medida de la inteligencia de las máquinas mediante el Test de Turing de Alan Turing, y la Conferencia de Dartmouth de 1956 introdujo formalmente el término *Inteligencia Artificial* en el mundo académico, marcando el nacimiento de la IA como campo de estudio independiente. En este periodo, los teóricos de la lógica demostraron la capacidad de las máquinas para resolver problemas, mientras que la invención de la máquina perceptiva sentó las primeras bases para la investigación posterior sobre redes neuronales y aprendizaje profundo. Las exploraciones y logros de este periodo no solo definieron la dirección de la investigación en IA, sino que también proporcionaron la base teórica y tecnológica para el desarrollo del aprendizaje automático, la visión por computadora y otros campos en las décadas siguientes.

El nacimiento de la visión por computadora en los años 60 marcó la exploración en profundidad del campo de la inteligencia artificial hacia la simulación y realización de la percepción visual humana. Durante este periodo, con el avance de la tecnología informática y el estudio en profundidad del concepto de inteligencia artificial, los académicos empezaron a prestar atención a cómo permitir que los ordenadores procesaran e interpretaran la información visual, es decir, que reconocieran, comprendieran y procesaran escenas y objetos del mundo real a través de imágenes y vídeos. En esta etapa, la visión por computador, como rama importante de la inteligencia artificial, se investiga con el objetivo de dotar a los ordenadores de capacidades de percepción visual similares a las del ojo humano, incluyendo el reconocimiento de imágenes, la comprensión de escenas, el seguimiento de objetos, etcétera. Se trata de la capacidad de extraer información útil de los datos de una imagen, reconocer patrones y objetos en imágenes, y comprender estos objetos y las relaciones entre ellos.

1.2. Objetivos

Para avanzar y profundizar en el campo del Seguimiento de Múltiples Objetos (MOT) dentro de la visión por computadora, este proyecto se propone alcanzar los siguientes objetivos de manera detallada y estructurada:

1. **Investigación Profunda en Sistemas de Seguimiento de Múltiples Objetos (MOT)**: Este objetivo se centra en una exploración de los sistemas MOT, abarcando desde sus principios fundamentales hasta las técnicas más innovadoras. Se pretende realizar un análisis comparativo de los diferentes enfoques y algoritmos utilizados en los sistemas de MOT, destacando sus fortalezas, limitaciones y campos de aplicación. Se investigarán tanto las metodologías basadas en aprendizaje automático tradicional como las que emplean técnicas de aprendizaje profundo, evaluando su eficacia en diversos escenarios.

2. **Desarrollo e Integración de Sistemas MOT con Detectores de Última Generación**: Tras la identificación de los sistemas MOT más prometedores, se procederá a su implementación práctica. Este paso implica la integración de sistemas MOT con tecnologías avanzadas de detección de objetos, como redes neuronales convolucionales (CNN) de última generación, para mejorar la precisión y la robustez del seguimiento en entornos complejos y dinámicos. Se establecerán protocolos de evaluación rigurosos, utilizando métricas específicas como precisión, tasa de identificación correcta, y eficiencia computacional, para medir el desempeño de estas soluciones integradas en escenarios de tiempo real.

3. **Elaboración de Documentación para Facilitar Investigaciones Futuras**: El propósito final es compilar y sistematizar la investigación realizada, las metodologías implementadas, los hallazgos de las pruebas y las evaluaciones en un compendio documental completo. Este documento no solo resumirá los logros y desafíos del proyecto, sino que también ofrecerá guías metodológicas detalladas, recomendaciones prácticas y perspectivas para futuras investigaciones. Se busca que este recurso sirva como una referencia esencial para investigadores y desarrolladores interesados en avanzar en el campo del MOT y en la aplicación de técnicas de detección avanzadas, estableciendo una base firme para futuras exploraciones y desarrollos tecnológicos en la visión por computadora.

1.3. Estructura del documento

La estructura del documento se ha concebido para facilitar la comprensión y el acceso rápido a la información, organizándola en secciones bien definidas que reflejan las etapas clave del proyecto y los resultados obtenidos. A continuación, se ofrece un resumen de la organización del documento:

1. **Capitulo I: Introducción**: Este capítulo establece el marco general del proyecto, introduciendo el concepto y la importancia del Seguimiento de Múltiples Objetos (MOT) dentro de la visión por computadora. Se discuten la motivación detrás del estudio, los objetivos específicos del proyecto y las principales contribuciones esperadas. Además, se proporciona un breve resumen de la estructura del documento, orientando al lector sobre lo que puede esperar en las secciones subsiguientes.

2. **Capitulo II: Estado de Arte**: Esta sección ofrece una revisión de la literatura relacionada con el MOT, destacando los avances recientes y las tecnologías clave en el campo. Se analizan tanto los enfoques tradicionales como las innovaciones recientes impulsadas por el aprendizaje profundo, identificando las tendencias actuales, los desafíos pendientes y las oportunidades de investigación. El objetivo es proporcionar un contexto detallado que fundamente las elecciones metodológicas y tecnológicas del proyecto.
3. **Capitulo III: Implementación, incorporación y configuración de nuevos detectores en sistemas de MOT**: Este capítulo consolida y expone detalladamente los procesos de implementación y optimización de tres modelos avanzados en sistemas de seguimiento de objetos múltiples. Inicialmente, se profundiza en la implementación del modelo DeepOCSORT, delineando su estructura teórica, arquitectura, integración con detectores y métodos de optimización, complementado con una evaluación exhaustiva de su rendimiento en contextos de seguimiento definidos. A continuación, se examina el modelo HybridSORT, destacando su enfoque híbrido que fusiona técnicas tradicionales de seguimiento con algoritmos de aprendizaje profundo, describiendo su diseño, integración y evaluación, y ofreciendo un análisis comparativo de su eficacia. Finalmente, se detalla la implementación de StrongSORT, abarcando su estructura, enfoques de aprendizaje profundo y comparativas con otras metodologías MOT, culminando con la presentación de estudios de caso y resultados experimentales que validan su competencia y rendimiento en diversos escenarios de seguimiento. El capítulo concluye delineando los recursos de hardware y software empleados durante el desarrollo del proyecto, proporcionando una perspectiva integral sobre el proceso de despliegue, integración y ajuste de los sistemas de seguimiento evaluados.
4. **Capitulo IV: Comparación de los resultados**: Este capítulo proporciona una comparativa de los resultados obtenidos con DeepOCSORT, HybridSORT y StrongSORT, utilizando un conjunto de métricas de evaluación estandarizadas. Se discuten las ventajas y limitaciones de cada sistema en términos de precisión, velocidad de procesamiento y robustez, facilitando una comprensión clara de su rendimiento relativo.
5. **Capitulo V: Conclusión y trabajos futuros**: El documento concluye resumiendo los hallazgos clave del proyecto, las lecciones aprendidas y el impacto de las implementaciones MOT en la visión por computadora. Se reflexiona sobre los desafíos enfrentados durante el proyecto y se ofrecen recomendaciones para investigaciones futuras, esbozando posibles direcciones para continuar avanzando en el campo del seguimiento de múltiples objetos.

Capítulo 2

Estado de Arte

El seguimiento de múltiples objetos, generalmente abreviado como MOT (Multiple Object Tracking) y también llamado MTT (Multiple Target Tracking) en algunas literaturas, es una importante dirección de investigación en el campo de la visión por ordenador, que implica principalmente el seguimiento de la posición y el movimiento de múltiples objetivos en movimiento simultáneamente en una secuencia de vídeo. La tecnología de seguimiento de objetivos múltiples tiene una amplia gama de aplicaciones en los campos de la videovigilancia, la conducción autónoma, el análisis de movimiento y la interacción persona-ordenador. Sin conocimiento previo del número de objetivos, se detectan múltiples objetivos en el vídeo, como peatones, coches, animales, etc., y se les asignan identificadores para el seguimiento de la trayectoria. Cada objetivo tiene un identificador distinto para poder predecir su trayectoria, encontrarlo con precisión, etc. El reto de los algoritmos de seguimiento multiobjetivo reside en cómo hacer frente a la oclusión de los objetivos, la interacción entre ellos, el movimiento rápido, los cambios de escena, etc. Para mejorar la precisión y robustez del seguimiento, los investigadores han desarrollado una variedad de algoritmos y técnicas, incluidos los métodos basados en el aprendizaje profundo que mejoran la detección de objetivos y el rendimiento de la asociación mediante el aprendizaje de grandes cantidades de datos. Una clasificación de algoritmos de seguimiento multiobjetivo conduce naturalmente aquí: el DBT (seguimiento por detección) frente al DFT (seguimiento sin detección), es decir, el seguimiento multiobjetivo basado en detección frente al seguimiento multiobjetivo basado en trama inicial sin detector.

- **Seguimiento multiobjetivo basado en la detección (DBT)**: Es una técnica avanzada en el campo de la visión por computadora que forma parte del Multi-Object Tracking, destinada a identificar y monitorear múltiples objetos de interés en secuencias de video. Este enfoque comienza con la detección precisa de objetos en cada cuadro utilizando métodos de visión por computadora o aprendizaje profundo, que permiten localizar y clasificar diferentes entidades en la escena. Posteriormente, se emplea un algoritmo de seguimiento para asociar cada objeto detectado en un cuadro con su correspondiente en cuadros anteriores, manteniendo así un registro continuo de su posición y movimiento. Este proceso no solo facilita la observación de la trayectoria y el comportamiento de los objetos a lo largo del tiempo, sino que también es crucial para aplicaciones que requieren una comprensión dinámica y en tiempo real del entorno, como la seguridad, la gestión del tráfico y la interacción avanzada entre humanos y

máquinas. En esencia, el seguimiento multiobjetivo basado en la detección proporciona una base sólida para sistemas que necesitan realizar un seguimiento fiable y preciso de múltiples objetos simultáneamente en entornos complejos y cambiantes.

- **Seguimiento multiobjetivo sin detección (DFT)**: Es una estrategia en el ámbito del Multi-Object Tracking donde los objetos en movimiento son rastreados a través de secuencias de video sin un proceso explícito de detección previa en cada cuadro. Esta técnica se basa en el seguimiento directo de características, como la apariencia o el movimiento, utilizando métodos como el flujo óptico o el seguimiento basado en la correlación, para estimar la trayectoria de los objetos sin identificarlos de forma individualizada al inicio. Este enfoque resulta particularmente útil en situaciones donde la detección es complicada debido a factores como oclusión, baja iluminación o alta densidad de objetos. Dentro del MOT, el seguimiento sin detección ofrece una alternativa valiosa, permitiendo la continuidad en el seguimiento de múltiples objetos incluso cuando las técnicas convencionales de detección no son aplicables o eficaces, lo que amplía significativamente las capacidades y la flexibilidad en el campo del seguimiento de objetos en entornos dinámicos y desafiantes.

El seguimiento multiobjetivo basado en detección se ha convertido en un método dominante en investigación y aplicaciones debido a su precisión y adaptabilidad relativamente altas. Con el desarrollo de técnicas de aprendizaje profundo, los algoritmos de detección y seguimiento de objetivos basados en el aprendizaje profundo han logrado mejoras significativas en el rendimiento, lo que ha impulsado aún más el desarrollo de técnicas de seguimiento multiobjetivo.

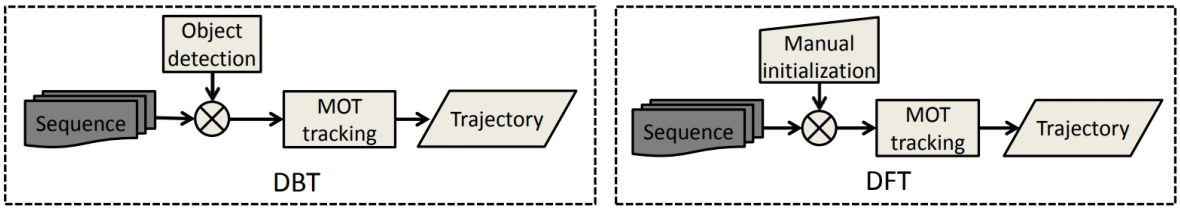


Figura 2.1: Flujo de procedimientos de dos enfoques de seguimiento destacados. Izquierda: Seguimiento basado en la detección (DBT), derecha: Seguimiento sin detección (DFT). [4]

2.1. Seguimiento de objetos (Object Tracking)

La propuesta y el desarrollo del seguimiento de objetos es un hito clave en el campo de la visión por computador, que se centra en la detección y el seguimiento automáticos y continuos de objetivos dinámicos en secuencias de vídeo. Esta línea de investigación trata de capacitar a los ordenadores para comprender y predecir el comportamiento de los objetivos en el espacio y el tiempo, y abarca una amplia gama de problemas, desde el simple seguimiento de un único objetivo hasta el complejo seguimiento de varios objetivos. Las primeras investigaciones sobre seguimiento de objetivos en los años 70 y 80 se centraron en entornos simplificados y aplicaciones específicas, como la detección de objetos en movimiento y el seguimiento de vehículos. Estos primeros intentos solían basarse en técnicas más básicas de procesamiento de

imágenes, como la detección del fondo, métodos de flujo óptico y técnicas sencillas de reconocimiento de patrones para la identificación y el seguimiento de objetivos.

En los años 90, a medida que aumentaba la potencia de cálculo y evolucionaban los algoritmos, el campo del seguimiento de objetivos empezó a adoptar enfoques más sofisticados. Durante este periodo, los investigadores desarrollaron el seguimiento multihipótesis (MHT, por sus siglas en inglés), uno de los primeros algoritmos exitosos para el seguimiento visual propuesto por Reid en 1979. MHT ofrece una solución sistemática al problema de asociación de datos construyendo un árbol de posibles hipótesis de seguimiento para cada objetivo candidato. El algoritmo calcula la probabilidad de cada pista y selecciona la combinación de pistas más probable. Significativamente, MHT es especialmente adecuado para utilizar información de orden superior, como los modelos de movimiento y apariencia a largo plazo, ya que al calcular las probabilidades se puede tener en cuenta toda la hipótesis de trayectoria.

El desarrollo de técnicas de aprendizaje profundo desde la década de 2010 hasta la actualidad ha revolucionado el seguimiento de objetivos. Los métodos basados en aprendizaje profundo fueron capaces de aprender automáticamente representaciones de características complejas a partir de grandes cantidades de datos, mejorando significativamente las capacidades de los algoritmos de seguimiento en entornos complejos. Durante este periodo, se propusieron algoritmos de seguimiento de extremo a extremo, capaces de pasar directamente de píxeles brutos a resultados de seguimiento de objetivos, mejorando la precisión y robustez del seguimiento. Al mismo tiempo, los conjuntos de datos estandarizados y los criterios de evaluación disponibles públicamente, como MOTChallenge, facilitaron la comparación y el desarrollo rápido de métodos en este campo.

2.2. Seguimiento de un solo objeto (Single Object Tracking)

La historia del desarrollo y avance tecnológico de las técnicas de seguimiento de un solo objeto (SOT) [7] en visión por computadora ha demostrado su importancia y su amplio potencial de aplicación. Desde principios de la década de 2010, la aparición de algoritmos de filtrado de correlación capturó rápidamente la atención de los investigadores debido a su excelente precisión y capacidad de procesamiento de alta velocidad. Posteriormente, se propusieron varios métodos de optimización alrededor de este marco algorítmico, como la optimización de características y la optimización de modelos, que mejoraron aún más el rendimiento de los algoritmos de seguimiento de objetivos basados en filtrado de correlación, convirtiéndolos en uno de los métodos principales en el campo del seguimiento de objetivos en la última década. La llegada del aprendizaje profundo ha revolucionado el seguimiento de objetivos, especialmente las redes siamesas que han recibido una amplia atención debido a su mayor velocidad computacional, demostrando una fuerte competitividad en el campo. Además, otras estructuras de redes neuronales profundas, como las redes neuronales convolucionales (CNN), las redes neuronales recurrentes (RNN) y las redes neuronales convolucionales gráficas (GCN), se han aplicado a tareas de seguimiento de objetivos, mostrando sus ventajas únicas.

En términos de aplicaciones, la tecnología de seguimiento de un solo objetivo juega un papel clave en varios campos como la seguridad pública, la conducción autónoma, la robótica inteligente y la interacción humano-ordenador. En el campo de

la seguridad pública, permite el seguimiento y localización en tiempo real de objetos clave en escenarios de vigilancia, controlando eficazmente a personas sospechosas. En la conducción autónoma, asiste en la navegación autónoma y la planificación de trayectorias, mejorando la seguridad y eficiencia de los sistemas. En robótica, mejora la capacidad de los robots para interactuar y realizar tareas a través de la navegación visual y el seguimiento de objetivos. En la interacción humano-ordenador, se utiliza para realizar sistemas inteligentes de retroalimentación basados en movimientos o gestos, ofreciendo una experiencia de interacción más natural e intuitiva a los usuarios. Estas aplicaciones no solo demuestran el valor práctico de la tecnología de seguimiento de un solo objetivo, sino que también promueven el desarrollo continuo de tecnologías y aplicaciones relacionadas.

Los principales retos en el seguimiento de vídeo que impactan directamente en el rendimiento de los algoritmos de seguimiento incluyen:

1. **Desaparición del objetivo**: Si el objetivo es completamente ocultado por otros objetos o sale del campo de visión de la cámara en algún momento de la secuencia, especialmente si la oclusión dura mucho tiempo o cubre un área grande, los algoritmos de seguimiento deben tener una gran capacidad de re-reconocimiento para reanudar el seguimiento rápidamente cuando el objetivo reaparezca. Esta capacidad es crucial para mantener la continuidad y precisión del seguimiento, y la falta de un mecanismo eficaz para manejar estas situaciones puede resultar en una significativa degradación de la efectividad del seguimiento o en la pérdida completa del objetivo.

2. **Cambios en el objetivo**: Los cambios en la forma y tamaño del objetivo causados por movimientos, como la deformación de un cuerpo humano al agacharse o acostarse, así como los movimientos de rotación del objetivo, incluyendo la rotación dentro del plano (el objetivo gira alrededor de un eje perpendicular al plano de la imagen, causando menos cambios en la apariencia) y la rotación fuera del plano (el objetivo gira alrededor de un eje no perpendicular al plano de la imagen, lo que puede causar cambios significativos en la apariencia), desafían la capacidad del algoritmo de seguimiento para reconocer y seguir continuamente al objetivo, y requieren que el algoritmo sea altamente adaptable y robusto para manejar estos complejos escenarios de cambio de objetivo.

3. **Interferencia de fondo**: El problema de la interferencia de fondo se relaciona principalmente con la complejidad del fondo y los cambios de iluminación. Distinguir eficazmente entre el primer plano y el fondo, reconocer precisamente los objetivos en primer plano y suprimir la información de fondo son esenciales para superar las interferencias de fondo. Además, los cambios en la iluminación no solo interfieren con el fondo sino que también pueden alterar las características visuales del objetivo. En particular, las fluctuaciones drásticas en la iluminación pueden provocar grandes cambios en las características de apariencia del objetivo en fotogramas consecutivos, al mismo tiempo que reducen la distinción entre el objetivo y el fondo dentro del mismo fotograma, lo que plantea mayores demandas en el rendimiento del algoritmo de seguimiento.

4. **Movimiento del objetivo**: El seguimiento de objetivos en vídeo generalmente se centra en objetivos en movimiento, lo que plantea una serie de desafíos, como el movimiento rápido del objetivo y el desenfoque de movimiento resultante. El movimiento rápido de un objetivo puede hacer que su posición cambie significa-

tivamente con respecto al fotograma anterior, y a veces incluso puede salirse del rango de búsqueda previsto. Al mismo tiempo, el desenfoque de movimiento reduce la claridad del primer plano del objetivo, interfiriendo en la captura precisa de las características del objetivo. Por otro lado, el movimiento de la propia cámara también puede provocar el desenfoque de todo el fotograma, dificultando aún más la tarea de seguimiento.

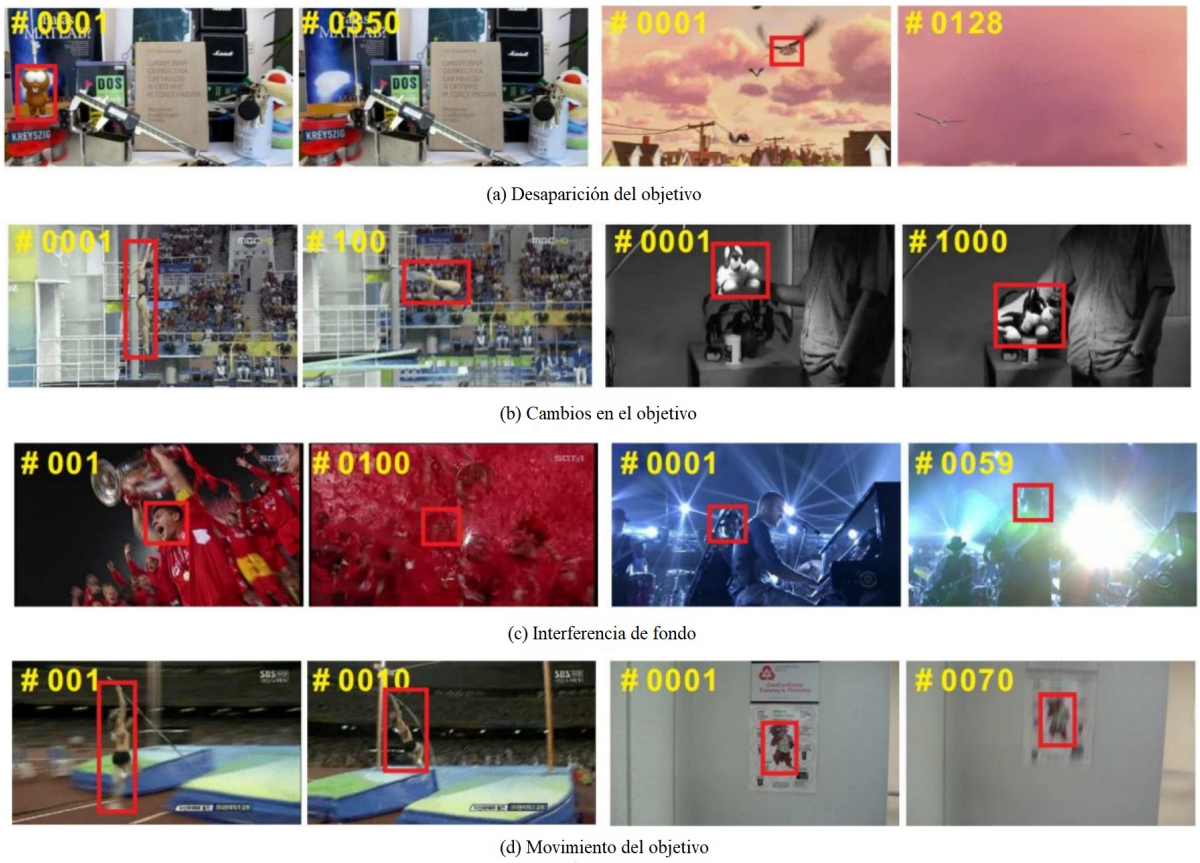


Figura 2.2: Principales retos del problema de seguimiento de objetivos. [7]

Los métodos de seguimiento de objetivos en video se pueden clasificar en tres categorías principales:

1. **Métodos de seguimiento de objetos basados en filtrado correlativo**: Han logrado un progreso significativo en tareas de seguimiento, destacando por su precisión y velocidad de procesamiento, convirtiéndose en uno de los marcos principales para el seguimiento de objetos en video en años recientes. La eficacia del filtrado correlativo en el seguimiento de objetos se debe principalmente a dos razones: a) la metodología de CF aprovecha implícitamente la operación de traslación cíclica para aumentar la diversidad de muestras de entrenamiento, mejorando así la robustez y precisión del algoritmo; b) la Transformada Rápida de Fourier (FFT) facilita el cálculo acelerado de operaciones convolutivas complejas en el dominio de la frecuencia, reduciendo la carga computacional y aumentando la eficiencia del modelo. Debido a su alta precisión y velocidad, los algoritmos de seguimiento basados en filtrado correlativo han visto un desarrollo masivo, demostrando buenos resultados en conjuntos de datos públicos.

Los métodos relacionados se dividen principalmente en dos direcciones: el uso de métodos de extracción de características más potentes y la construcción de modelos de aprendizaje de filtros más robustos.

2. **Métodos de seguimiento de objetos basados en redes siamesas**: Representan un enfoque avanzado que utiliza tecnologías de aprendizaje profundo para el seguimiento de objetivos. Las redes siamesas consisten en dos subredes idénticas que comparten los mismos pesos y pueden procesar dos entradas diferentes en paralelo. En la aplicación de seguimiento de objetivos, las redes siamesas se utilizan comúnmente para comparar el objeto objetivo con candidatos dentro de los cuadros de video, para determinar la posición del objeto objetivo. Las ventajas de los métodos de seguimiento basados en redes siamesas radican en su eficiencia y precisión. Dado que las dos estructuras de red son idénticas y comparten pesos, este enfoque puede reducir efectivamente el consumo de recursos computacionales. Además, al aprender las características profundas del objeto objetivo, las redes siamesas pueden lograr un seguimiento preciso del objetivo en escenas complejas.

3. **Otros métodos de seguimiento de objetos basados en aprendizaje profundo**: como los algoritmos de seguimiento con redes convolucionales (por ejemplo, MDNet, Redes Neuronales Convolutivas Multi-Dominio), que fue el algoritmo ganador del desafío VOT-2015 y representa uno de los primeros algoritmos de seguimiento de objetos totalmente basados en CNN. Los algoritmos de seguimiento con redes neuronales recurrentes (RNN) representan otro enfoque que utiliza RNN para procesar datos secuenciales de video para el seguimiento de objetos. Las RNN son particularmente adecuadas para manejar datos secuenciales ya que pueden mantener memoria de información anterior, lo cual es crucial para el seguimiento de objetivos en videos. En escenarios de seguimiento de objetivos, el algoritmo necesita comprender los cambios dinámicos del objetivo en cuadros consecutivos, y las RNN pueden capturar efectivamente esta información dinámica a través de su estructura recurrente.

Para abordar los desafíos encontrados en el seguimiento de objetos en videos, se han desarrollado los siguientes métodos:

1. **Desaparición del objetivo**: Para solucionar el problema de la desaparición de objetivos en el seguimiento de videos, los investigadores han propuesto estrategias como actualizaciones del modelo de alta confianza, mecanismos de redetección de objetivos y el uso de redes de movimiento de fondo basadas en trayectorias históricas del objetivo. Estos métodos mejoran la robustez del modelo, predicen la posición del objetivo después de la oclusión y realizan una redetección efectiva cuando el objetivo reaparece, abordando efectivamente situaciones de oclusión prolongada y desaparición grave del objetivo.

2. **Cambio del objetivo**: Para resolver el problema de deformación en el seguimiento de objetivos, los investigadores han utilizado métodos como CSR-DCF basado en modelos estadísticos y DSAR-CF con detección de prominencia para mejorar el rendimiento del seguimiento. Además, en el marco de las redes siamesas, algoritmos como RASNet y la combinación de seguimiento de objetivos con segmentación como SiamMask y D3S, mediante la extracción de información de la forma del objeto y áreas precisas del primer plano, enfrentan efectivamente deformaciones complejas del objetivo. A pesar de los avances, las deformaciones

rápidas y severas siguen siendo un desafío clave para futuras investigaciones.

3. **Interferencia de fondo**: Para abordar la interferencia de fondo en el seguimiento de objetivos, los investigadores han utilizado información contextual, como CACF para aprender sobre el fondo, y LMCF para evitar la interferencia de objetos similares. Además, algoritmos como PGNet y KYS utilizan coincidencia de características e información de la escena para reducir las distracciones y mejorar la precisión del seguimiento, aunque los escenarios con objetivos similares densos siguen siendo un desafío.
4. **Movimiento del objetivo**: Para enfrentar los desafíos del movimiento del objetivo, como las diferencias de posición entre frames consecutivos y el desenfoque por movimiento, los investigadores han recurrido a la predicción de movimiento (como MACF), modelado de desenfoque por movimiento y la utilización de características de movimiento del objetivo (como DMF) para mejorar el seguimiento. Sin embargo, cómo manejar eficientemente el impacto del desenfoque por movimiento en el seguimiento aún requiere más investigación.

2.3. Seguimiento de múltiples objetos (Multi Object Tracking)

El seguimiento de múltiples objetos [4] ocupa un lugar central en el campo de la visión por computadora, siendo responsable de identificar, seguir múltiples objetivos y generar sus trayectorias independientes en flujos de video, especialmente en el seguimiento de peatones. La tecnología MOT respalda tareas visuales avanzadas como la estimación de posturas, el reconocimiento de acciones y juega un papel clave en aplicaciones prácticas como la vigilancia de video y la interacción humano-computadora. Los desafíos a los que se enfrenta esta tecnología incluyen, pero no se limitan a, la oclusión frecuente de objetivos, la inicialización y terminación de trayectorias, la similitud en la apariencia de los objetivos y la interacción entre múltiples objetivos. Para abordar estos desafíos, los investigadores han propuesto una variedad de soluciones en las últimas décadas, siendo los métodos basados en el aprendizaje profundo los más predominantes en la actualidad.

Los métodos de implementación de MOT se pueden dividir en dos categorías principales: Seguimiento por Detección (Tracking-by-Detection) y Detección y Seguimiento Conjuntos (Joint Detection and Tracking). El primero aplica detectores de objetos en cada fotograma del video para localizar los objetivos y luego asocia los objetivos detectados en fotogramas consecutivos para formar trayectorias basándose en puntuaciones de similitud. Algoritmos representativos incluyen SORT y DeepSORT, entre otros. El segundo aborda las tareas de detección y seguimiento simultáneamente dentro de un mismo marco, permitiendo que la información de seguimiento mejore el rendimiento de detección, lo cual representa la nueva tendencia en la investigación de MOT. Métodos destacados en esta categoría incluyen Tracktor y JDE, los cuales han demostrado un desempeño sobresaliente.

2.3.1. Seguimiento por Detección (Tracking-by-Detection)

En esta sección se explicarán diferentes métodos de seguimiento por detección secuencial:

2.3.1.1. Simple Online and Realtime Tracking (SORT)

El método de Seguimiento Simple en Línea y en Tiempo Real (SORT) [8] propone una estrategia eficiente frente a las soluciones tradicionales para el Seguimiento de Múltiples Objetivos, como el Seguimiento de Múltiples Hipótesis (MHT) y el Filtro de Asociación de Datos Probabilísticos Conjuntos (JPDA), que se ven afectadas por su complejidad combinatoria. Este método se centra en construir modelos de apariencia y movimiento simplificados, para lograr un seguimiento en línea, en tiempo real y viable. El núcleo de la tecnología SORT incluye el uso de tecnología de detección basada en Redes Neuronales Convolucionales (CNN), la predicción del estado del objetivo en futuros cuadros, la asociación de detecciones actuales con objetivos existentes y la gestión del ciclo de vida de los objetos de seguimiento.

SORT utiliza el marco de Redes Convolucionales de Regiones Más Rápidas (Faster R-CNN) para la detección de objetivos, empleando un proceso de dos etapas que comparte parámetros para aumentar la eficiencia de la detección. El estudio compara dos arquitecturas de red proporcionadas con Faster R-CNN, específicamente la arquitectura de Zeiler y Fergus (FrRCNN(ZF)) y la arquitectura más profunda de Simonyan y Zisserman (FrRCNN(VGG16)), descubriendo que la calidad de la detección influye significativamente en el rendimiento del seguimiento.

Además, SORT aproxima los desplazamientos entre cuadros de cada objeto mediante un modelo de velocidad constante lineal, optimizando la actualización del estado del objetivo mediante un filtro de Kalman. Cuando una detección no se puede asociar con un objetivo, el estado se predice sin corrección utilizando el modelo de velocidad lineal.

$$x = [u, v, s, r, u', v', s']^T \tag{2.1}$$

La ecuación representa el estado de un objetivo en seguimiento de objetos, donde cada término es crucial para entender y predecir el comportamiento del objetivo:

- **u**: Denota la ubicación de píxeles horizontales del centro del objetivo.
- **v**: Denota la ubicación de píxeles verticales del centro del objetivo.
- **s**: Es la escala (área) de aspecto del cuadro delimitador del objetivo.
- **r**: Es la relación de aspecto del cuadro delimitador del objetivo.
- **u'**: La velocidad en la dimención horizontal.
- **v'**: La velocidad en la dimención vertical.
- **s'**: La velocidad en la dimención de escala.

En la asignación de objetivos, SORT calcula la distancia de intersección sobre unión (IOU) entre cada detección y todas las cajas delimitadoras predichas de los objetivos existentes, resolviendo la asignación de manera óptima con el algoritmo húngaro. Además, se impone un IOU mínimo para rechazar asignaciones donde la superposición de detección a objetivo es menor que un umbral de IOUmin, manejando efectivamente las ocultaciones a corto plazo.

Para los objetivos que entran o salen del cuadro, SORT gestiona sus identidades creando o eliminando rastreadores según sea necesario. Si un objetivo no se detecta

durante un número predefinido de cuadros, se termina su seguimiento para evitar el seguimiento incorrecto y el desperdicio de recursos. Los experimentos demuestran que SORT mejora significativamente la precisión y la robustez del seguimiento de múltiples objetivos mientras mantiene la capacidad de operar en tiempo real.

2.3.1.2. BoT-SORT

Bot-SORT [9] es una algoritmo de seguimiento de múltiples peatones diseñado para abordar los desafíos de rastrear efectivamente a varios peatones en escenas complejas. Este método se basa principalmente en la construcción de un grafo bipartito para optimizar el problema de asociación entre objetivos, es decir, cómo asociar correctamente a los peatones detectados en el cuadro actual con los peatones que ya están siendo seguidos. Este enfoque combina las ventajas de la información de movimiento y apariencia, e introduce compensación de movimiento de la cámara y un vector de estado de filtro de Kalman más preciso, para mejorar la robustez y precisión del seguimiento.

- **Compensación de Movimiento de la Cámara (CMC)**: Se utiliza una técnica de compensación de movimiento global para abordar los desafíos que afectan la superposición de cuadros de predicción y detección debido a cámaras en movimiento.
- **Mejoras en el Filtro de Kalman**: Al estimar directamente el ancho y alto de los cuadros delimitadores, se mejora el modelado del movimiento de los objetos en el plano de la imagen para un mejor rendimiento.
- **Fusión IoU - Re-ID**: La fusión de representaciones visuales profundas para características de reidentificación (Re-ID) con la intersección sobre la unión (IoU) mejora la capacidad del rastreador para mantener la identidad correcta a lo largo del tiempo.

BoT-SORT y su variante BoT-SORT-ReID han demostrado un rendimiento excepcional en los conjuntos de datos MOT17 y MOT20 de MOTChallenge, clasificándose en primer lugar en indicadores MOT clave como MOTA, IDF1 y HOTA. Con este método, los investigadores pueden seguir efectivamente a varios peatones u objetos en videos, manteniendo alta precisión y robustez incluso en escenas complejas.

2.3.1.3. OC-SORT

OC-SORT [10] es una versión del algoritmo SORT centrado en la observación, diseñado para mejorar la robustez del Seguimiento de Múltiples Objetivos, especialmente en escenarios con oclusiones y movimientos no lineales de objetivos. Los métodos tradicionales de seguimiento basados en el filtro de Kalman, como SORT, asumen que el movimiento del objetivo es lineal, lo que puede llevar a inexactitudes durante las oclusiones o movimientos no lineales. OC-SORT aborda estas limitaciones centrándose en la observación de objetivos en lugar de solo en la estimación del estado. Incorpora:

- **Actualización Centrada en la Observación (ORU)**: Utiliza observaciones del estado del objeto durante las oclusiones para generar trayectorias virtuales y corregir la acumulación de errores en los parámetros del filtro durante las oclusiones. Esto permite más pasos temporales para corregir los errores acumulados

durante las oclusiones. ORU utiliza la última observación vista y la observación más reciente al reactivar el seguimiento como puntos de anclaje para generar observaciones virtuales.

- **Momentum Centrado en la Observación (OCM)**: Integra la consistencia de dirección de las trayectorias de manera centrada en la observación en la matriz de costos para ayudar en el proceso de asociación. OCM calcula la diferencia de dirección comparando la consistencia entre la dirección de dos observaciones en una trayectoria existente y la dirección entre observaciones históricas y nuevas en la trayectoria.
- **Recuperación Centrada en la Observación (OCR)**: Es una técnica heurística que intenta una segunda asociación entre la última observación de trayectorias no emparejadas y observaciones no emparejadas después de la fase de asociación regular para recuperar trayectorias de pérdidas. Esto puede manejar situaciones donde un objeto se detiene brevemente o está ocluido.

Con estas tres técnicas, se reduce con precisión la acumulación de errores y se mejora la precisión del seguimiento en entornos de seguimiento desafiantes. Este método está diseñado para ser simple, en línea y en tiempo real, capaz de lograr un rendimiento de vanguardia en diversos conjuntos de datos, mientras opera eficientemente en un solo CPU.

2.3.1.4. Deep OC-SORT

Deep OC-SORT [11] es una versión mejorada del algoritmo OC-SORT, que integra métodos de aprendizaje profundo para aumentar la precisión y eficiencia en el seguimiento de objetivos. El seguimiento de múltiples objetivos es una dirección de investigación importante en el campo de la visión por computadora, con el objetivo de rastrear la posición y el movimiento de múltiples objetivos dinámicos en secuencias de video. Esto se logra y mejora mediante la introducción de los siguientes tres módulos centrales:

- **Compensación de Movimiento de la Cámara (CMC)**: Se enfoca en localizar objetos de manera más precisa de un cuadro a otro en escenas en movimiento. Ajustando las tres componentes de OC-SORT individualmente mediante matrices de rotación proporcionales y transformaciones de traslación, para localizar de manera más precisa los objetos en movimiento.
- **Apariencia Dinámica**: Ajustando dinámicamente los factores de ponderación de los embeddings de apariencia basados en la confianza del detector, para integrar flexiblemente la información de apariencia en el modelo de seguimiento. Esto permite incorporar selectivamente la información de apariencia en el modelo de trayectoria en situaciones de alta calidad, mientras se rechazan embeddings dañados debido a ocultaciones o desenfoques.
- **Ponderación Adaptativa**: Aumentando el peso de los embeddings de apariencia basados en su discriminación, lo que ayuda a mejorar la precisión de la correspondencia entre trayectorias y detecciones durante el seguimiento. Ajustando el peso basado en la similitud de apariencia entre el seguimiento y la detección, así como en la discriminación basada en esta similitud, para utilizar de manera más efectiva la información de apariencia en el seguimiento.

Con estas integraciones, se combina efectivamente las pistas de movimiento y apariencia, superando desafíos como ocultaciones, desenfoque por movimiento y objetos con apariencias similares, logrando mejoras significativas en el rendimiento en los benchmarks MOT17, MOT20 y DanceTrack. Estas mejoras hacen que Deep OC-SORT muestre un rendimiento de seguimiento excepcional en tareas de seguimiento de múltiples objetivos, especialmente en escenas dinámicas complejas.

2.3.1.5. HybridSORT

HybridSORT [12] es un método para seguimiento de múltiples objetivos que mejora la precisión y estabilidad del seguimiento combinando pistas débiles (como el estado de confianza, el estado de altura y la dirección de velocidad) y pistas fuertes (como información espacial y de apariencia). Este método está diseñado para abordar los problemas de seguimiento causados por la oclusión y la agrupación de objetos, manteniendo las características del seguimiento en línea simple y en tiempo real (SORT), mientras que mejora su rendimiento a través de varias técnicas clave:

- **Modelado de confianza de trayectoria (TCM)**: Utiliza la información histórica del objetivo para estimar su confianza actual, ayudando a mantener un seguimiento preciso del objetivo durante oclusiones o cambios significativos en la apariencia del objetivo.
- **IoU modulado por altura (HMIoU)**: Mejora la diferenciación entre objetivos considerando las diferencias de altura, adecuado para el seguimiento de multitudes a diferentes distancias, aumentando la precisión de seguimiento en escenas complejas.
- **Momento central robusto de observación (ROCM)**: : Captura la tendencia de movimiento del objetivo considerando la información de momento de los cuatro ángulos del objetivo (no solo el punto central), mejorando la precisión de seguimiento cuando el movimiento del objetivo es complejo o cambia rápidamente.
- **Predicción lineal y filtro de Kalman**: Combina la predicción lineal y el filtro de Kalman, ajustando la estrategia de seguimiento de acuerdo con la confianza del objetivo para mejorar la estabilidad y precisión del seguimiento.

Hybrid-SORT, a través de estas técnicas, ha logrado una mejora significativa en el rendimiento de seguimiento, demostrando su superioridad en múltiples pruebas de referencia (como MOT17, MOT20 y DanceTrack), convirtiéndose en una solución efectiva para aplicaciones de seguimiento de múltiples objetivos en tiempo real. Esta descripción proporciona un resumen exhaustivo del algoritmo, explicando cómo integra eficazmente diferentes pistas y técnicas de seguimiento para abordar problemas comunes en MOT, logrando mejoras de rendimiento sin necesidad de entrenamiento adicional.

2.3.1.6. ByteTrack

ByteTrack [13] es un método que mejora significativamente el rendimiento de seguimiento al asociar cada caja de detección (independientemente de su puntuación) de manera destacada. A diferencia de los métodos tradicionales, que descartan los resultados de detección de baja puntuación y pueden perder objetivos reales causando interrupciones en el seguimiento, ByteTrack utiliza la similitud con las trayectorias

existentes para recuperar y seguir con precisión los objetos, incluidos aquellos parcialmente ocultos o con baja confianza en la detección. Este algoritmo adopta un método de asociación de datos simple pero efectivo, BYTE, que conserva casi todas las cajas de detección y las divide en dos categorías: de alta puntuación y baja puntuación. Primero, asocia las cajas de detección de alta puntuación con las trayectorias. Luego, asocia las cajas de detección de baja puntuación con estas trayectorias no coincidentes para recuperar los objetos dentro de las cajas de detección de baja puntuación y al mismo tiempo filtrar el fondo. Los pasos clave son los siguientes:

- Para cada cuadro de la secuencia de video, usar un detector para predecir las cajas de detección y sus puntuaciones.
- Según un umbral de puntuación de detección τ, dividir todas las cajas de detección en cajas de detección de alta puntuación (D_high) y baja puntuación (D_low).
- Usar un filtro de Kalman para predecir la nueva posición de cada trayectoria en el cuadro actual.
- Primero realizar una coincidencia basada en la similitud entre las cajas de detección de alta puntuación y las trayectorias.
- Para las trayectorias no coincidentes y las cajas de detección de baja puntuación, realizar una segunda coincidencia basada en su similitud.

Este método proporciona una solución simple, efectiva y eficiente para aplicaciones de seguimiento de múltiples objetivos en tiempo real. Al combinar un detector de alto rendimiento con este innovador método de asociación, ByteTrack ha logrado resultados de vanguardia en varios conjuntos de datos de seguimiento, demostrando su robustez y eficacia en el manejo de escenarios de seguimiento complejos.

2.3.1.7. StrongSORT

StrongSORT [14] es un marco de seguimiento de múltiples objetivos avanzado, diseñado para mejorar significativamente el rendimiento del rastreador DeepSORT clásico mediante la mejora de la detección de objetos, la incrustación de características y la asociación de trayectorias. Establece un punto de referencia fuerte y justo para la comunidad MOT, introduciendo dos algoritmos ligeros y plug-and-play para abordar los problemas inherentes de "pérdida" en MOT: la asociación perdida y la detección perdida. El marco StrongSORT resuelve los problemas en MOT a través de dos tecnologías clave:

- **Modelo de Enlace sin Apariencia (AFLink)**: Este es un modelo para la asociación global que no depende de la información de apariencia. Puede lograr enlaces efectivos entre objetivos sin utilizar características de apariencia, mejorando así la precisión y robustez del seguimiento.

- **Interpolación Suave Gaussiana (GSI)**: Para mitigar el problema de detección perdida, la técnica GSI se aplica a la interpolación para estimar suavemente la ubicación posible de los objetivos cuando falta la detección. Este método ayuda a mantener el seguimiento continuo de los objetivos, incluso cuando desaparecen temporalmente del campo de visión.

Combinando StrongSORT, Modelo de Enlace sin Apariencia y Interpolación Suave Gaussiana, el sistema final StrongSORT++ logra resultados de vanguardia en múltiples pruebas de referencia públicas (como MOT17, MOT20, DanceTrack y KITTI).

2.3.2. Seguimiento Conjuntos (Joint Detection and Tracking)

En esta sección se explicarán diferentes métodos de seguimiento conjuntos.

2.3.2.1. Joint Detection and Embedding (JDE)

La detección y el embebido conjunto [15] es un método utilizado en el seguimiento de múltiples objetivos, que estima simultáneamente las cajas delimitadoras de los objetos y las características embebidas a través de una única red. Este método integra información de movimiento y apariencia del objeto, lo cual es crucial para el seguimiento de objetos bajo condiciones difíciles donde los objetivos pueden ser brevemente perdidos o bloqueados. La principal ventaja del método JDE es su simplicidad y eficiencia, proporcionando un marco unificado para manejar simultáneamente tareas de detección y embebido, facilitando así el proceso de seguimiento.

Entre los diversos métodos, uno llamado SimpleTrack destaca por su enfoque innovador para resolver el problema de asociación de datos en el seguimiento de múltiples objetivos. Implementa una nueva matriz de asociación —la matriz de embebido y Giou (Intersección sobre Unión Generalizada) (matriz EG)— que combina la distancia coseno del embebido del objeto y la distancia Giou. A través de este enfoque, SimpleTrack no solo mejora el rendimiento de la asociación de datos, como en los indicadores HOTA (High Order Tracking Accuracy) y IDF1 (Identity F1 Score) en el conjunto de datos MOT17, sino que también aumenta la velocidad de seguimiento en aproximadamente un 20 %.

Específicamente, en el caso del método JDE, este suele usar una red única para predecir directamente las cajas de detección y las características de apariencia de los objetivos. Sin embargo, la competencia entre la detección y la identificación puede afectar el proceso de optimización del aprendizaje multitarea. Para abordar este problema, SimpleTrack introduce un módulo de desacoplamiento de características, que utiliza diferentes métodos de fusión de características para manejar de forma separada las representaciones de detección y de reidentificación (ReID), así como una nueva estrategia de seguimiento basada en la matriz EG y la estrategia BYTE. De esta manera, SimpleTrack busca mejorar el rendimiento del método JDE y simplificar su arquitectura.

2.3.2.2. CenterTrack

CenterTrack [13] es un marco de trabajo de extremo a extremo para la detección y seguimiento de objetos simultáneamente. Procesa dos fotogramas de video consecutivos y un mapa de calor de detecciones previas como entrada, y luego produce los resultados de detección y sus desplazamientos de seguimiento para el fotograma actual. Este método simplifica el proceso de seguimiento representando cada objeto por un punto central y rastreando estos puntos centrales a lo largo del tiempo. Su principal objetivo es ser simple, procesar en línea (procesamiento secuencial de cada fotograma sin necesidad de entrada futura) y en tiempo real, logrando un rendimiento de vanguardia en múltiples estándares de seguimiento sin necesidad de estrategias

de asociación complejas ni modelos de movimiento adicionales. Especialmente en situaciones de alta densidad de objetos y oclusión, muestra mejoras significativas en la precisión y velocidad de seguimiento en conjuntos de datos de referencia como MOT17 y KITTI. Se construye sobre el detector CenterNet, utilizando una imagen única como entrada para generar una serie de resultados de detección, identificando cada categoría de objeto por un punto central y regresando a las dimensiones de la caja del objeto. Además, CenterTrack, al proporcionar los resultados de seguimiento del fotograma anterior como entrada, puede aprender a recuperar objetos perdidos o ocluidos a partir de pistas adicionales. Este método combina la detección y seguimiento de objetos en un solo marco, simplificando el proceso de seguimiento en comparación con métodos tradicionales, reduciendo la complejidad computacional y mejorando la precisión y eficiencia del seguimiento.

2.3.3. Detección de objetos en imágenes

La detección de objetos en imágenes representa un reto significativo que consiste en identificar y localizar objetos dentro de imágenes o secuencias de video. Aunque para los seres humanos esta tarea puede resultar intuitiva y se realiza con facilidad, para las máquinas constituye un desafío considerable. Los humanos tienen la capacidad innata de reconocer objetos variados, considerando atributos como el tamaño, el color y la nitidez, prácticamente de manera instantánea. En contraste, las máquinas, y en particular CNN, enfrentan limitaciones significativas para detectar una diversa gama de objetos, en especial aquellos para los cuales no han sido explícitamente entrenadas. Las dificultades se amplifican cuando los objetos presentan variaciones en tamaño, orientaciones o están sujetos a distintas condiciones de iluminación. Los algoritmos de detección de objetos también deben superar retos adicionales como la presencia de sombras, variaciones en la orientación de los objetos y la necesidad de determinar con precisión la ubicación de cada objeto en la imagen. En suma, la detección de objetos es una disciplina compleja y en constante desarrollo, que exige la aplicación de técnicas avanzadas y la implementación de modelos sofisticados para alcanzar niveles de precisión y fiabilidad elevados.

2.3.3.1. Faster - RCNN

Faster R-CNN [17] es un marco avanzado para la detección de objetos en imágenes, capaz de compartir características convolucionales de toda la imagen con la red de detección, lo que permite la generación de propuestas de región casi sin costo adicional. Esta integración reduce significativamente el costo computacional adicional de calcular propuestas. Al compartir características convolucionales, Faster R-CNN combina RPN y Fast R-CNN en una única red, lo que permite que el sistema mantenga una alta tasa de cuadros por segundo (5fps) mientras alcanza la precisión de detección de objetos más avanzada en múltiples conjuntos de datos. Para lograr su eficiencia y precisión, implementa los siguientes mecanismos:

- **Red de Propuestas de Región (RPN)**: Capaz de generar automáticamente y de manera eficiente propuestas de región de alta calidad (cuadros de candidatos) a partir de imágenes. RPN escanea el mapa de características convolucionales, utilizando una ventana deslizante para proponer regiones candidatas en cada ubicación.
- **Características Convolucionales Compartidas**: RPN y la red de detección Fast

R-CNN comparten el mismo conjunto de características convolucionales, lo que reduce los cálculos redundantes y acelera todo el proceso de generación de propuestas y detección de objetos.

- **Entrenamiento de Extremo a Extremo**: Faster R-CNN se puede entrenar mediante el algoritmo de retropropagación de manera end-to-end, lo que significa que tanto RPN como la red de detección Fast R-CNN pueden entrenarse al mismo tiempo, mejorando aún más la precisión y eficiencia de la detección.

La ejecución simultánea de estos tres mecanismos permite que Faster R-CNN mantenga una alta precisión de detección mientras mejora la velocidad de detección.

2.3.3.2. YOLO (You Only Look Once)

YOLO (You Only Look Once) [18] es un innovador sistema de detección de objetos que simplifica el problema de detección a un único problema de regresión, permitiendo el procesamiento de imágenes en tiempo real a una velocidad extremadamente alta. Utiliza una CNN para predecir directamente múltiples cuadros delimitadores y sus probabilidades de clase desde la imagen completa, evitando los cálculos complejos de los métodos tradicionales. Específicamente, YOLO divide la imagen en una cuadrícula, y cada celda de la cuadrícula predice la posición del cuadro delimitador, la confianza y las probabilidades de clase. Esta arquitectura permite el entrenamiento de extremo a extremo y la detección rápida, manteniendo al mismo tiempo una alta precisión media.

Los puntos clave incluyen:

- **Arquitectura de red**: Una única CNN, inspirada en GoogLeNet, pero utiliza capas convolucionales de 1×1 y 3×3 para optimizar la dimensión del espacio de características.
- **Método de entrenamiento**: Primero se preentrena en el conjunto de datos de ImageNet, luego se ajusta finamente en conjuntos de datos de detección de objetos, utilizando una función de pérdida de múltiples partes para optimizar directamente el rendimiento de detección.
- **Rendimiento**: Capaz de procesar imágenes a una velocidad de hasta 155 cuadros por segundo, aunque puede enfrentar desafíos en la localización precisa de objetos pequeños, supera a los métodos tradicionales en la mayoría de los casos.

El diseño de YOLO no solo logra una detección de objetos rápida y precisa, sino que también tiene una buena capacidad de generalización, adecuada para objetos de diferentes tamaños y formas y para nuevos campos de aplicación.

2.3.3.3. SSD (Single Shot MultiBox Detector)

El Detector de Múltiples Cajas [19] en una Sola Pasada es un método de detección de objetos que emplea una única red neuronal convolucional de avance para identificar múltiples objetos dentro de una imagen. A diferencia de los métodos tradicionales, SSD realiza la predicción de cuadros delimitadores y probabilidades de clase de objetos simultáneamente en una sola pasada de la red, lo que mejora significativamente la velocidad y eficiencia de la detección.

La arquitectura SSD se basa en una red convolucional estándar, como VGG-16, truncada antes de cualquier operación de clasificación. SSD extiende esta base añadiendo múltiples capas de características convolucionales, permitiendo la detección en múltiples escalas. Esta característica distingue a SSD de otros métodos de una sola escala, facilitando una adaptación más flexible a objetos de diversos tamaños.

SSD utiliza filtros convolucionales pequeños para predecir desplazamientos de ubicación de cuadros delimitadores y confianzas de clases de objetos en cada capa de características. Estas predicciones se basan en un conjunto de cuadros delimitadores predeterminados, cubriendo la imagen en varias escalas y proporciones. La supresión no máxima (NMS) es un paso post-procesamiento esencial, asegurando que cada objeto se detecte una vez con el resultado óptimo.

En práctica, SSD muestra una precisión competitiva con métodos que incluyen pasos adicionales de propuesta de objetos, mientras ofrece ventajas en velocidad y eficiencia. Esto lo convierte en una elección ideal para sistemas de detección de objetos en tiempo real.

2.3.4. Tecnologias NAS

En el campo del aprendizaje automático, la Búsqueda de Arquitectura Neuronal (NAS, por sus siglas en inglés) [20] representa una tecnología avanzada diseñada para automatizar la creación de arquitecturas de redes neuronales. Tradicionalmente, el diseño de arquitecturas neuronales ha requerido una inversión considerable de conocimiento especializado y tiempo, limitando su flexibilidad y la velocidad de innovación. NAS acelera significativamente el proceso de diseño de modelos y tiene el potencial de descubrir arquitecturas optimizadas que superen a las diseñadas manualmente. El núcleo de NAS gira en torno a la exploración de tres dimensiones principales: el espacio de búsqueda, la estrategia de búsqueda y la estrategia de estimación de rendimiento, que juntas definen la capacidad y la eficiencia de NAS.

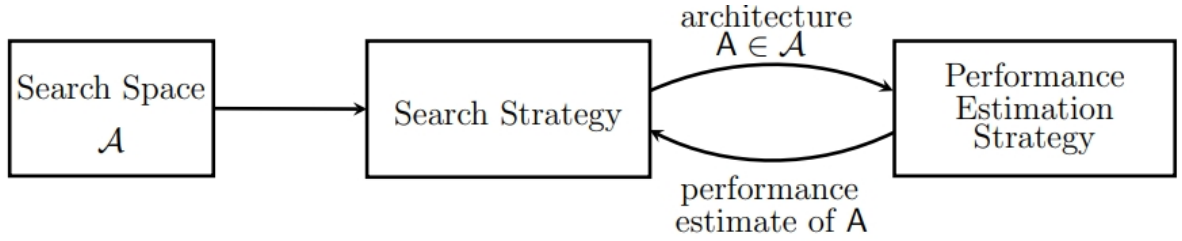


Figura 2.3: Ilustración abstracta de los métodos de NAS

En la figura presenta una ilustración abstracta de los procesos utilizados en la identificación de arquitecturas de redes neuronales es presentada. Dentro de este proceso, un mecanismo de búsqueda escoge una arquitectura específica, denominada A, de un conjunto predeterminado conocido como espacio A. Posteriormente, esta arquitectura A es evaluada por un sistema de predicción de rendimiento, que calcula y proporciona una estimación del desempeño de A, la cual es entregada de vuelta al mecanismo de búsqueda.

- **Espacio de Búsqueda**: Es el conjunto de arquitecturas posibles que se pueden explorar. Incluye redes estructuradas en cadena, donde se varían las operaciones y hiperparámetros de cada capa, y también enfoques más complejos como

la búsqueda basada en células, donde se optimizan estructuras repetitivas o células que luego se apilan para formar la arquitectura final. Estas metodologías permiten una amplia variedad en la configuración de la red, desde la profundidad y anchura de las capas hasta las conexiones entre ellas, ofreciendo una rica diversidad de arquitecturas para encontrar la más óptima para una tarea dada.

  - **Búsqueda por capa**: Se centra en explorar las operaciones y configuraciones en cada capa individual de la red. Por ejemplo, en una red estructurada en cadena, cada capa recibe la entrada de la capa anterior, y se pueden variar las operaciones en cada capa, como la convolución, agrupación y sus hiperparámetros asociados.
  - **Búsqueda basada en celdas**: En lugar de definir una arquitectura completa, este enfoque busca estructuras repetitivas o çélulas", optimizando estas células que luego se apilan para formar la arquitectura completa. Esta aproximación reduce el tamaño del espacio de búsqueda y permite una transferencia más sencilla de arquitecturas entre diferentes tareas y conjuntos de datos.
  - **Búsqueda a nivel de red**: Este enfoque intenta optimizar la arquitectura completa de la red, considerando elementos como la profundidad de la red, la anchura de cada capa y las conexiones entre capas. Los enfoques más avanzados permiten redes con múltiples ramificaciones y conexiones de salto, aumentando significativamente la complejidad del espacio de búsqueda.

- **Estrategia de Búsqueda**: La estrategia de búsqueda determina cómo explorar el espacio de búsqueda definido. Este proceso implica un equilibrio entre encontrar rápidamente arquitecturas de alto rendimiento y evitar quedar atrapado en óptimos locales. Una estrategia de búsqueda efectiva puede mejorar significativamente la eficiencia del proceso de búsqueda, incluyendo, pero no limitándose a, algoritmos evolutivos, métodos basados en gradientes y optimización bayesiana.

  - **Búsqueda aleatoria**: Exploración simple pero efectiva del espacio de búsqueda sin ninguna guía específica.
  - **Optimización bayesiana**: Utiliza el éxito de las evaluaciones anteriores para informar la elección de la próxima arquitectura a evaluar, buscando equilibrar la exploración y explotación del espacio de búsqueda.
  - **Métodos evolutivos**: Aplican principios de evolución biológica, donde las arquitecturas se consideran como individuos en una población. A través de procesos de selección, mutación y cruce, se generan nuevas arquitecturas.
  - **Aprendizaje por refuerzo**: Trata la búsqueda de arquitectura como un problema de RL, donde un agente selecciona acciones (elección de arquitectura) para maximizar una recompensa (rendimiento de la arquitectura).
  - **Métodos basados en gradientes**: Permiten la optimización directa en el espacio de búsqueda utilizando gradientes, a menudo a través de una relajación del espacio de búsqueda para hacerlo continuo y diferenciable.

- **Estrategia de Estimación de Rendimiento**: La estrategia de estimación de rendimiento tiene como objetivo evaluar la efectividad de una arquitectura es-

pecífica, generalmente a través de su rendimiento predictivo en un conjunto de validación. Dado el alto costo computacional de entrenar y evaluar modelos completos, los investigadores han desarrollado varias técnicas para acelerar el proceso de estimación de rendimiento, como el uso de modelos de baja fidelidad, extrapolación de curvas de aprendizaje, herencia de pesos y técnicas de morfismo de redes.

- **Entrenar y Probar**: Entrenar completamente una arquitectura y luego evaluarla, aunque esto puede ser computacionalmente costoso.
- **Compartir Pesos**: Utiliza un modelo de ün solo disparo"donde los pesos se entrenan una vez y se comparten entre varias arquitecturas, reduciendo significativamente el tiempo y los recursos computacionales necesarios.
- **Extrapolación de Curvas de Aprendizaje**: Implica entrenar una arquitectura por unas pocas épocas y luego usar la curva de aprendizaje para predecir su rendimiento futuro, evitando el entrenamiento completo.
- **Modelos de Un Solo Disparo (One-shot)**: Entrena un modelo grande y sobreparametrizado una vez, y luego utiliza subgrafos de este modelo como arquitecturas candidatas, permitiendo una evaluación rápida sin entrenamiento adicional.

Las distintas estrategias utilizadas en la búsqueda de arquitectura de redes neuronales pueden combinarse de múltiples formas para formular un método de NAS eficaz. La combinación ideal de estas estrategias tiende a ser única para cada situación y depende del tipo de problema a resolver, los recursos de cómputo que se tienen a mano y los propósitos específicos que se pretenden lograr mediante el NAS.

Capítulo 3

Implementación, incorporación y configuración de nuevos detectores en sistemas de MOT.

En este capítulo, se describe minuciosamente el procedimiento para implementar distintos sistemas de seguimiento de objetos múltiples. Tras la instalación y verificación de estos métodos, se procede a su integración con el detector más avanzado para evaluar su eficacia en el seguimiento, con el objetivo de profundizar en el conocimiento de estas tecnologías. Para llevar a cabo este proceso, se emplean varios conjuntos de datos de acceso público. Las etapas del proceso incluyen un análisis exhaustivo de los datos disponibles, la implementación de los sistemas, la incorporación del detector de vanguardia, la modificación de los conjuntos de datos para su adecuación, el entrenamiento del detector y, finalmente, la valoración de los resultados obtenidos.

3.1. Datasets de MOT

En esta sección se analizan los conjuntos de datos clave en el campo del seguimiento de objetos. Esta revisión es esencial para comprender las bases sobre las que se desarrollan y evalúan los algoritmos de seguimiento, aplicables en áreas como la vigilancia y el análisis de comportamiento. Se detallan las estructuras de los datasets, los formatos de los archivos, las etiquetas disponibles y los desafíos específicos que presentan.

3.1.1. MOT 15

MOT15 [21], conocido también como MOTChallenge 2015, se establece como un benchmark esencial para el seguimiento de múltiples objetivos, diseñado para estandarizar la evaluación de algoritmos de seguimiento utilizando secuencias del mundo real. Con 22 secuencias en total, divididas equitativamente entre datos de entrenamiento y de prueba, el conjunto de datos abarca una diversidad de escenarios, incluyendo cámaras estáticas y móviles, variados ángulos de captura, y diferentes condiciones ambientales. Este enfoque integral tiene como objetivo fomentar el desarrollo y la comparación objetiva de métodos de seguimiento en una variedad de

situaciones realistas.

Para los participantes y usuarios del MOT15, el conjunto de datos sigue un formato estandarizado tanto para la detección como para las anotaciones (ground truth), proporcionando información detallada sobre cada objeto rastreado en las secuencias. La estructura del formato de datos se describe en la siguiente tabla:

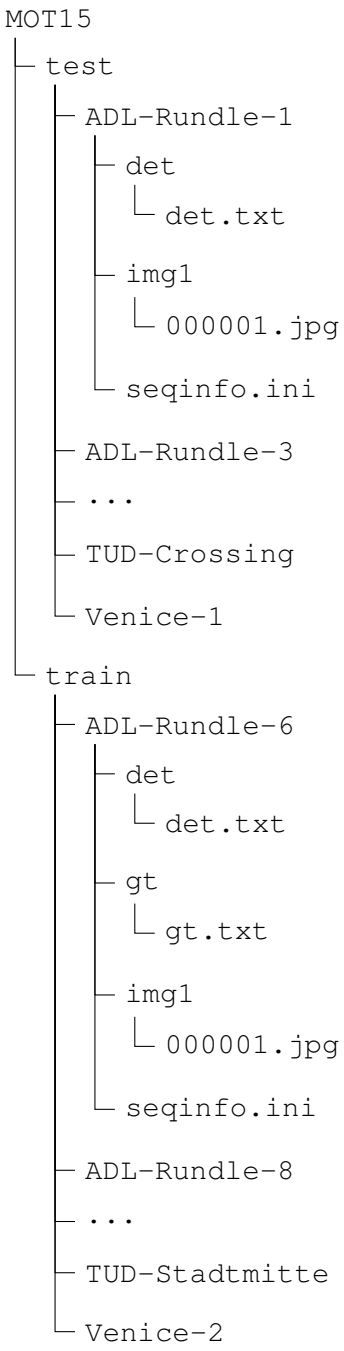
Posición	Nombre	Descripción
1	Número de cuadro	Indica en qué cuadro está presente el objeto
2	Número de identidad	Cada trayectoria peatonal está identificada por un ID único (-1 para detecciones)
3	Bounding box izquierda	Coordenada de la esquina superior izquierda del bounding box peatonal
4	Bounding box superior	Coordenada de la esquina superior izquierda del bounding box peatonal
5	Ancho del bounding box	Ancho en píxeles del bounding box peatonal
6	Altura del bounding box	Altura en píxeles del bounding box peatonal
7	Puntuación de confianza	Indica cuán confiado está el detector de que esta instancia es un peatón. Para el ground truth y resultados, actúa como una bandera sobre si la entrada debe ser considerada
8	Posición x 3D	Posición 3D x del peatón en coordenadas del mundo real (-1 si no está disponible)
9	Posición y 3D	Posición 3D y del peatón en coordenadas del mundo real (-1 si no está disponible)
10	Posición z 3D	Posición 3D z del peatón en coordenadas del mundo real (-1 si no está disponible)

Tabla I: Formato de datos para los archivos de entrada y salida tanto para detección como para archivos de anotación.

Esta tabla proporciona una guía clara para entender y trabajar con los datos de MOT15, asegurando que los investigadores y desarrolladores puedan evaluar eficazmente sus algoritmos de seguimiento en un entorno estandarizado y controlado.

3.1.1.1. Estructura de dataset

El dataset MOT15 es un recurso clave para la evaluación de algoritmos en el seguimiento de múltiples objetos, especialmente diseñado para facilitar la comparación y el análisis en la comunidad de investigación. A continuación se ilustra la organización de la estructura de directorios de MOT15, que se divide en dos segmentos principales: test y train. Cada uno contiene diversas secuencias, representadas por subdirectorios, que a su vez incluyen archivos de detección (det.txt), imágenes (img1) y archivos de configuración específicos de la secuencia (seqinfo.ini). En el directorio de entrenamiento, además, se proporcionan archivos de anotaciones de verdad terreno (gt.txt) para cada secuencia, permitiendo un entrenamiento supervisado y la validación de los algoritmos de seguimiento. Esta estructura meticulosamente organizada asegura un acceso y manipulación eficientes de los datos, crucial para el desarrollo y evaluación de técnicas de seguimiento avanzadas.



3.1.2. MOT 16

MOT16, parte integral del marco MOTChallenge, representa un avance significativo en la evaluación del seguimiento de múltiples objetos, enfocado especialmente en peatones dentro de una amplia gama de entornos. Al construir sobre la base establecida por MOT15, MOT16 introduce mejoras clave y extiende el conjunto de datos y criterios de evaluación, con el objetivo de estandarizar y promover el progreso en los métodos de seguimiento de múltiples objetivos.

Para proporcionar claridad en el uso y análisis de MOT16, a continuación se detallan las especificaciones del formato de datos y las clases de etiquetas utilizadas

en el conjunto de datos:

Especificaciones para los archivos de detección (DET) y anotación/verdad terreno (GT)

Posición	Nombre	Descripción
1	Número de cuadro	Marca el cuadro específico en que se detecta el objeto.
2	Número de identidad	Asigna a cada peatón un identificador único (-1 para detecciones no asignadas).
3	Bounding box izquierda	Coordenada X de la esquina superior izquierda del cuadro delimitador.
4	Bounding box superior	Coordenada Y de la esquina superior izquierda del cuadro delimitador.
5	Ancho del bounding box	Medida en píxeles del ancho del cuadro delimitador.
6	Alto del bounding box	Medida en píxeles del alto del cuadro delimitador.
7	Puntuación de confianza	En DET, refleja la confianza de la detección. En GT, 1 para objetos a considerar, 0 para ignorar.
8	Clase	En GT, especifica el tipo de objeto anotado.
9	Visibilidad	En GT, proporción del objeto que es visible, valor entre 0 y 1.

Tabla II: Especificaciones para los archivos de detección (DET) y anotación/verdad terreno (GT).

Las características destacadas de MOT16:

- **Calidad y Diversidad de Datos**: MOT16 eleva el estándar con secuencias de video de mayor calidad y escenarios más complejos que su predecesor. Estas mejoras desafían a los algoritmos de seguimiento a identificar y seguir objetos en condiciones que simulan más estrechamente el mundo real, aumentando así la aplicabilidad de los resultados de evaluación.
- **Tamaño del Conjunto de Datos**: Este conjunto de datos ve una expansión considerable en comparación con MOT15, con una mayor cantidad de secuencias de video y objetos. Esto brinda a los investigadores la oportunidad de probar sus algoritmos en un espectro más amplio de situaciones, fomentando avances significativos en tecnologías de seguimiento.
- **Calidad de la Anotación**: Con anotaciones más precisas y detalladas, MOT16 asegura que los investigadores puedan evaluar sus algoritmos de seguimiento con una gran confianza en la fiabilidad y precisión de los resultados.
- **Incremento en los Desafíos**: Introduciendo escenarios con baja tasa de cuadros y condiciones nocturnas, MOT16 pone a prueba la robustez y la precisión de los algoritmos bajo condiciones desafiantes y dinámicas.
- **Criterios de Evaluación Mejorados**: Aunque conserva las métricas fundamentales de MOT15, como MOTA y MOTP, la inclusión de escenarios adicionales y condiciones de evaluación más variadas refina y amplía el marco de evaluación, ofreciendo una medida más integral y desafiante del rendimiento de seguimiento.

Descripción de las Clases de Etiquetas

Etiqueta	ID	Descripción
Peatón	1	Representa a un individuo a pie.
Persona en vehículo	2	Indica a una persona dentro de un vehículo.
Coche	3	Vehículo de tipo automóvil.
Bicicleta	4	Vehículo de tipo bicicleta.
Moto	5	Vehículo de tipo motocicleta.
Vehículo no motorizado	6	Cualquier vehículo sin motorización.
Persona estática	7	Persona que permanece inmóvil.
Distractor	8	Elemento que podría ser confundido con un peatón.
Oclusor	9	Objeto que bloquea la vista parcial de otro.
Oclusor en el suelo	10	Objeto sobre el suelo que obstruye la vista de otro objeto.
Oclusor completo	11	Objeto que oculta completamente a otro.
Reflejo	12	Reflejo de un peatón u otro objeto.
Multitud	13	Agrupación de personas.

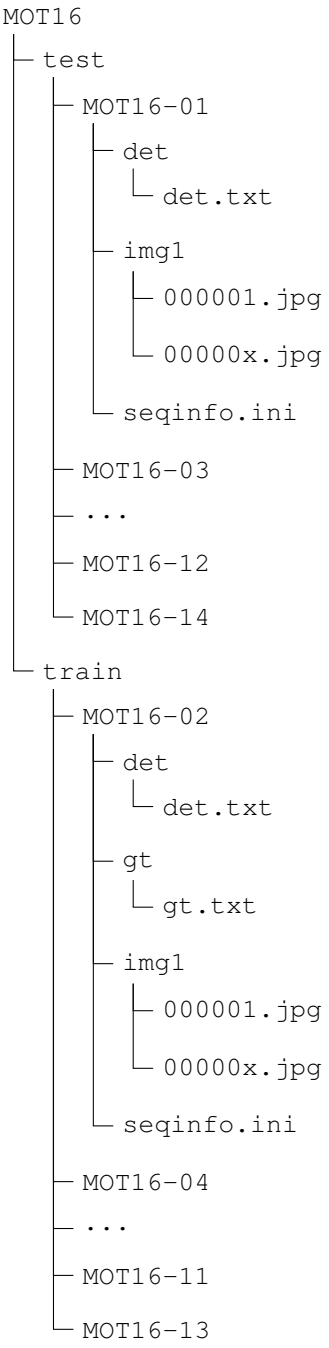
Tabla III: Descripción de las clases de etiquetas en los archivos de anotación, referenciadas en la séptima columna de los archivos.

3.1.2.1. Estructura de dataset

La estructura presentada describe la organización del conjunto de datos MOT16, un recurso ampliamente utilizado en la comunidad de visión por computadora para el seguimiento de múltiples objetos en secuencias de video. La estructura jerárquica detalla cómo se organizan los datos dentro del conjunto, facilitando a los usuarios la localización y el uso de la información específica necesaria para sus investigaciones o aplicaciones. A continuación, se ofrece una descripción detallada de cada componente de la estructura:

- **Subdirectorios Test y Train:** El conjunto de datos se divide en dos subdirectorios principales: 'test' y 'train'. El directorio 'test' se utiliza para evaluar el rendimiento de los algoritmos de seguimiento, mientras que 'train' contiene datos destinados al entrenamiento de estos algoritmos.
- **Secuencias de Datos:** Dentro de cada subdirectorio ('test' y 'train'), hay múltiples directorios correspondientes a secuencias individuales, como MOT16-01, MOT16-02, etc. Cada uno de estos directorios representa una secuencia de video diferente.
- **Directorio de Detecciones (det):** Cada secuencia tiene un subdirectorio 'det', que contiene un archivo 'det.txt'. Este archivo incluye los datos de detección de objetos, esenciales para el proceso de seguimiento.
- **Directorio de Ground Truth (gt):** Presente solo en las secuencias del conjunto de 'train', este directorio contiene el archivo 'gt.txt', que ofrece las anotaciones de referencia sobre la posición y otros atributos de los objetos en las secuencias.
- **Directorio de Imágenes (img1):** Aquí se almacenan las imágenes individuales que componen cada secuencia de video, etiquetadas secuencialmente (por ejemplo, 000001.jpg, 00000x.jpg), y que sirven como los datos visuales para el seguimiento.
- **Archivo de Información de la Secuencia (seqinfo.ini):** Cada secuencia contiene un archivo 'seqinfo.ini', que proporciona metadatos esenciales sobre la secuencia, como la resolución de las imágenes, la tasa de cuadros y la duración

de la secuencia.



3.1.3. MOT 17

Para mejorar la descripción de MOT17 y resaltar sus avances y características de forma más clara y detallada, consideremos enfocarnos en la estructura, precisión y claridad del texto. Aquí tienes una versión revisada que profundiza en los aspectos significativos de MOT17, destacando su importancia en el campo del seguimiento de múltiples objetos.

MOT17 representa un hito en la evolución de los conjuntos de datos para el seguimiento de múltiples objetos, avanzando significativamente respecto a su prede-

cesor, MOT16. Este conjunto de datos introduce mejoras notables en varios frentes críticos para el desarrollo y evaluación de algoritmos de seguimiento. A continuación, se detallan las principales mejoras y características que distinguen a MOT17:

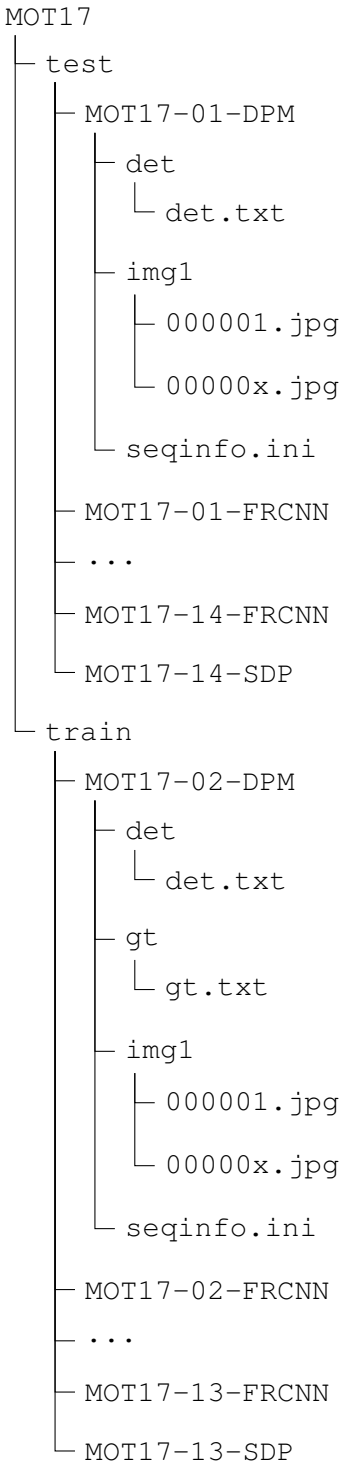
- **Mejora en la Cantidad y Calidad de Datos**: MOT17 sobresale por su enriquecido conjunto de secuencias de vídeo, las cuales abarcan una gama más amplia de escenarios complejos y variados. Este aumento en la diversidad y complejidad de los escenarios plantea retos más significativos para los algoritmos de seguimiento, empujando los límites de su capacidad y precisión. Además, la calidad de las anotaciones ha sido meticulosamente mejorada, ofreciendo una base de datos con anotaciones más precisas y consistentes. Esta mejora en la calidad de las anotaciones es crucial para garantizar evaluaciones fiables y detalladas de los algoritmos de seguimiento.
- **Avances en Métodos de Anotación**: Empleando técnicas de anotación de vanguardia, MOT17 proporciona anotaciones de alta precisión y consistencia. Esta precisión es vital para la evaluación adecuada de los algoritmos de seguimiento, ya que permite a los investigadores realizar pruebas más rigurosas y afinar sus desarrollos con mayor efectividad.
- **Inclusión de Diversidad de Detectores**: A diferencia de MOT16, que se limitaba a resultados de detección de un único tipo de detector, MOT17 amplía el espectro al incluir resultados de múltiples detectores. Esta característica introduce una dimensión adicional de flexibilidad y adaptabilidad, permitiendo a los investigadores evaluar la robustez de sus algoritmos bajo una variedad más amplia de condiciones de detección. Tal diversidad es esencial para comprender y mejorar la interoperabilidad de los algoritmos en diferentes entornos y situaciones.
- **Escalada en Desafíos y Propósitos de Uso**: Con el objetivo de empujar las fronteras del desarrollo algorítmico, MOT17 incrementa la complejidad de los desafíos presentados. Introduce escenarios aún más complejos y proporciona datos de detección enriquecidos, fomentando el avance en el diseño y la implementación de algoritmos de seguimiento más sofisticados y eficaces. Esto convierte a MOT17 en una plataforma de prueba excepcionalmente rica y desafiante, ideal para la experimentación y el refinamiento de tecnologías de seguimiento de múltiples objetos.

En conclusión, MOT17 no solo supera a MOT16 en términos de calidad de datos, métodos de anotación y diversidad de detectores, sino que también eleva significativamente el nivel de desafío. Al hacerlo, establece un nuevo estándar para las plataformas de prueba en el campo del seguimiento de múltiples objetos, ofreciendo una herramienta invaluable para impulsar la investigación y el desarrollo de algoritmos más avanzados y eficientes.

3.1.3.1. Estructura de dataset

El conjunto de datos MOT17, utilizado para la evaluación y el análisis de algoritmos de seguimiento de múltiples objetos, presenta una estructura organizada que facilita su manejo y procesamiento. Esta estructura detallada es crucial para entender cómo se organizan las secuencias de prueba y entrenamiento, junto con los archivos de detección, anotaciones y las imágenes correspondientes. En particular,

el directorio de test contiene 21 elementos distintos, y el directorio de train incluye la misma cantidad de elementos, cada uno correspondiendo a diferentes configuraciones y escenarios de seguimiento:



3.1.4. MOT 20

MOT20 representa un avance significativo en la evaluación de algoritmos de seguimiento de múltiples objetos en entornos altamente concurridos. Introduce se-

cuencias con densidades de peatones más altas, un mayor número de cuadros delimitadores y entornos de imagen de alta resolución, estableciendo nuevos estándares para el rendimiento de seguimiento en condiciones dinámicas y densamente pobladas del mundo real.

Las diferencias clave entre MOT 17 y MOT 20 son los siguientes:

- Mayor densidad de peatones, aumentando la complejidad de las situaciones de seguimiento.
- Incremento en la cantidad de cuadros delimitadores, proporcionando un conjunto de datos más rico.
- Introducción de secuencias de alta resolución y desde perspectivas elevadas.
- Mayor diversidad de entornos de prueba, incluyendo interiores, exteriores y condiciones diurnas y nocturnas.

Para facilitar la comprensión del formato de datos utilizado en MOT20, así como las etiquetas de clases, se presentan dos tablas revisadas que proporcionan una descripción detallada de cada elemento.

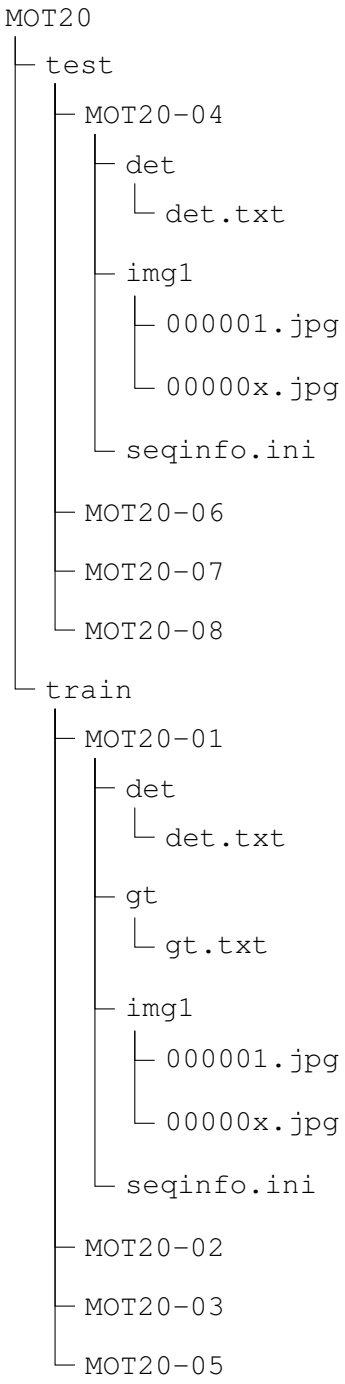
Posición	**Nombre**	**Descripción**
1	Número de cuadro	Indica en qué cuadro está presente el objeto.
2	Número de identidad	Cada trayectoria peatonal está identificada por un ID único (-1 para detecciones).
3	Bounding box izquierda	Coordenada X del rincón superior izquierdo del bounding box.
4	Bounding box superior	Coordenada Y del rincón superior izquierdo del bounding box.
5	Bounding box ancho	Ancho en píxeles del bounding box.
6	Bounding box alto	Alto en píxeles del bounding box.
7	Puntuación de confianza	Indica qué tan confiable es el detector de que esta instancia es un peatón.
8	Clase	Especifica el tipo de objeto anotado (solo GT).
9	Visibilidad	Ratio de visibilidad, un número entre 0 y 1 que indica cuánto del objeto está visible.

Tabla IV: Formato de datos para los archivos tanto de detección (DET) como de anotación/verdad terreno (GT).

3.1.4.1. Estructura de dataset

El dataset MOT20, diseñado para la evaluación de algoritmos de seguimiento de múltiples objetos, se caracteriza por su complejidad y realismo, enfocándose en entornos de alta densidad peatonal. La estructura del directorio de MOT20 se divide en segmentos de test y train, cada uno conteniendo cuatro secuencias representadas por subdirectorios individuales, sumando un total de ocho elementos diferentes entre ambos segmentos. Dentro de estos subdirectorios, se encuentran archivos de detección (det.txt), imágenes (img1), y archivos de configuración de secuencia (seqinfo.ini). Adicionalmente, el directorio de train incluye archivos de anotaciones de verdad terreno (gt.txt) para cada secuencia, esenciales para el entrenamiento y evaluación de

los modelos de seguimiento. Esta estructura organizada es vital para facilitar el acceso y la gestión eficiente de los datos, permitiendo a los investigadores concentrarse en el desarrollo y la mejora de algoritmos de seguimiento robustos y precisos en escenarios desafiantes.



3.2. Tecnologías empleadas

Para ejecutar las operaciones de implementación, integración y entrenamiento en este proyecto, se detallan a continuación las herramientas de software y hardware que se utilizaron durante el desarrollo. En lo que respecta al hardware, se utilizó un

ordenador de sobremesa, que cuenta con un procesador i5-12600K, una tarjeta gráfica Nvidia GeForce RTX 3060Ti, 32GB de memoria RAM y un disco SSD de 1TB para el almacenamiento. En el ámbito del software, este equipo de sobremesa opera con el sistema operativo Windows 10 Pro, en su versión de actualización 22H2, proporcionando una plataforma robusta y confiable para todas las actividades necesarias en el proyecto.

Además, se establecieron múltiples entornos virtuales para facilitar el entrenamiento de los detectores. Se ha utilizado un entorno Linux virtualizado para testear el despliegue de los sistemas, aunque finalmente todos los despliegues se ha realizado en el entorno local.

Durante el desarrollo del proyecto se utilizó el lenguaje de programación Python. Para cumplir con los requerimientos establecidos, se seleccionaron y utilizaron diversas bibliotecas de Python, las cuales se especifican a continuación:

- **cv2**: Una biblioteca de OpenCV que proporciona herramientas para el procesamiento de imágenes y visión por computadora, permitiendo la manipulación de imágenes, detección de objetos, y más.
- **json**: Un módulo de Python que permite la codificación y decodificación de datos en formato JSON, facilitando la interacción con APIs y el almacenamiento de datos.
- **numpy**: Una biblioteca fundamental para la computación científica en Python, que ofrece soporte para arrays y matrices grandes y multidimensionales, junto con una colección de funciones matemáticas para operar con estos.
- **os**: Un módulo que proporciona una forma de usar funcionalidades dependientes del sistema operativo, como leer o escribir en el sistema de archivos, manejar rutas, etc.
- **pycocotools**: Una biblioteca que asiste en la carga, análisis y visualización de los datos en el formato COCO, muy usada en tareas de detección, segmentación y reconocimiento de objetos en imágenes.
- **random**: Un módulo que implementa generadores de números pseudoaleatorios para diversas distribuciones, utilizado para la selección aleatoria de datos o parámetros.
- **super-gradients**: Una biblioteca o módulo específico (posiblemente personalizado o parte de un framework mayor) para implementar y gestionar gradientes en modelos de aprendizaje profundo, optimizando el entrenamiento.
- **torch**: Conocida como PyTorch, es una biblioteca de aprendizaje automático que proporciona dos características de alto nivel: operaciones tensoriales con aceleración de GPU y diferenciación automática para construir y entrenar redes neuronales.
- **ultralytics**: Una empresa o grupo que ofrece soluciones y bibliotecas en el ámbito del aprendizaje profundo, probablemente se refiere a alguna de sus bibliotecas o herramientas específicas, como YOLOv5 para detección de objetos.
- **yolox**: Una variante de la arquitectura YOLO para la detección de objetos, que se enfoca en la optimización y mejoras en la velocidad y precisión de la detección.

3.3. Despliegue del sistema

En esta sección, se describe el despliegue de los sistemas de MOT seleccionados para alcanzar los objetivos del proyecto:

3.3.1. Despliegue de SORT

Para completar el despliegue del algoritmo SORT en un computador local, es fundamental seguir detalladamente las instrucciones proporcionadas en el repositorio oficial en GitHub, subido por Alex Bewley, Zongyuan Ge, y otros[8] colaboradores. Aquí se presenta una guía general basada en la descripción mencionada:

- **Clonar el Repositorio de SORT**: Se inicia clonando el repositorio oficial de SORT desde GitHub. Esto puede hacerse ejecutando el comando en una terminal, asegurándose de tener git instalado en el sistema.
- **Instalar Dependencias**: Una vez clonado el repositorio, se navega al directorio del proyecto y se instalan las dependencias necesarias. Esto generalmente incluirá librerías como NumPy, SciPy, y otras necesarias para el manejo de vídeo y análisis de imagen, según lo especificado en el archivo "requirements.txt" del repositorio.
- **Preparar el Entorno de Datos**: Para integrar los datasets del MOT Challenge, como los de 2017 y 2015, se requiere descargar estos datasets y colocarlos en la ubicación especificada por la configuración de SORT. A menudo, se debe modificar algún archivo de configuración dentro del código fuente de SORT para apuntar al directorio donde se encuentran estos datasets.
- **Adaptar la Configuración para Diferentes Datasets**: Dependiendo del dataset que se desee utilizar, puede ser necesario ajustar la configuración específica dentro del código de SORT. Esto podría incluir cambiar rutas de acceso a los datos, parámetros de detección o seguimiento, y otras configuraciones relevantes para optimizar el rendimiento del algoritmo bajo diferentes condiciones.
- **Ejecutar Pruebas**: Finalmente, se ejecuta SORT utilizando un script provisto en el repositorio o creando uno propio basado en los ejemplos proporcionados. Esto permitirá verificar que el sistema está correctamente configurado y operativo.

Los procedimientos mencionados anteriormente se encuentran documentados en el Anexo 1[7.1], donde se especifican las instrucciones esenciales para la implementación, incluyendo aquellos detalles que no están directamente referenciados en el repositorio de GitHub.

Con la realización de estos pasos, se completa satisfactoriamente la instalación del sistema SORT, asegurando su correcto funcionamiento para el seguimiento de objetos en tiempo real.

3.3.2. Despliegue de Deep-OC-SORT

El despliegue del sistema Deep-OC-SORT fue realizado siguiendo meticulosamente las directrices provistas en el repositorio oficial en GitHub. Este proyecto, diseñado para operar con versiones de Python a partir de la 3.8.0, exige la imple-

mentación de una serie de procedimientos detallados para asegurar su adecuada ejecución y evaluación.

Inicialmente, el proceso se inició con la clonación del repositorio de Deep-OC-SORT. Posteriormente, se procedió a la instalación de las dependencias necesarias, las cuales se encontraban distribuidas en varios subdirectorios del proyecto, requiriendo la ejecución de comandos específicos para su instalación.

Además, se incluyó un paso crucial que consistió en la descarga de un archivo esencial desde un enlace proporcionado, el cual fue posteriormente ubicado en la carpeta designada dentro del proyecto. Este archivo es fundamental para el funcionamiento correcto del sistema.

La preparación de los conjuntos de datos también constituyó un componente esencial del despliegue. Se llevaron a cabo comandos específicos para convertir los conjuntos de datos MOT17, MOT20 y DanceTrack al formato COCO, facilitando así su integración y uso en el sistema.

Para la evaluación del sistema, se establecieron configuraciones base utilizando comandos que desactivan ciertas funcionalidades recientes, permitiendo evaluar el rendimiento del sistema en un entorno controlado. Los resultados de estas pruebas fueron almacenados en ubicaciones predefinidas dentro de la estructura del proyecto, incluyendo tanto los resultados estándares como aquellos que involucran post-procesamiento de interpolación lineal.

Los procedimientos detallados anteriormente están documentados en el Anexo 2[7.2], donde se especifican las instrucciones esenciales para la implementación del sistema, incluyendo aquellos detalles que no se encuentran directamente referenciados en el repositorio de GitHub.

Con la finalización de estos pasos, se logró completar satisfactoriamente la instalación del sistema Deep-OC-SORT, garantizando su funcionamiento eficaz para el seguimiento de objetos en tiempo real.

3.3.3. Despliegue de Hybrid-SORT

La implementación del algoritmo Hybrid-SORT se ha realizado con adaptaciones específicas destinadas a alinearse con los objetivos del proyecto en curso. El objetivo principal ha sido la integración de este avanzado algoritmo de seguimiento con modelos de detección de vanguardia. Se descubrió que ciertos pasos y conjuntos de datos sugeridos en la documentación original no eran completamente aplicables al contexto específico de este proyecto, lo que llevó a una modificación significativa de la metodología para satisfacer las necesidades de integración y optimización del equipo.

Inicialmente, se clonó y configuró el repositorio de Hybrid-SORT, seguido por la instalación de pycocotools y otros paquetes esenciales. Un cambio significativo fue la sustitución del componente “faiss-gpu” por “faiss-cpu”, adaptándose a las limitaciones del hardware utilizado en el proyecto. Estos ajustes preliminares son esenciales para preparar el entorno necesario que permitirá la ejecución eficiente del algoritmo.

En la fase siguiente, se abordó meticulosamente la preparación de los datos. Optando por los conjuntos de datos MOT17 y MOT20 por su relevancia, se estableció

una estructura de directorios adecuada para asegurar su correcta implementación y accesibilidad. A través de scripts específicos, los datos fueron convertidos al formato COCO, lo que facilita su uso en aplicaciones y herramientas que reconocen este formato estándar en el ámbito de la visión por computadora.

Finalmente, a pesar de que el proyecto no contemplaba el entrenamiento de modelos de detección desde cero, se incorporaron modelos preentrenados para evaluar y calibrar el sistema de seguimiento. Estos modelos se utilizaron para realizar pruebas y ajustes, asegurando una integración efectiva y un funcionamiento óptimo con los datos preparados. Los entrenamientos de modelos de ablación, así como pruebas específicas para los conjuntos de datos MOT17 y MOT20 utilizando los datos mixtos, fueron pasos cruciales para validar la eficacia del algoritmo bajo diversas condiciones operativas.

Los procedimientos mencionados anteriormente están detallados en el Anexo 3[7.3], donde se describen las instrucciones cruciales para la implementación del sistema Hybrid-SORT, incluyendo detalles que no se encuentran directamente mencionados en el repositorio de GitHub.

Con la ejecución de estos pasos, se logra completar con éxito la instalación del sistema Hybrid-SORT, garantizando su adecuado funcionamiento para el seguimiento avanzado de objetos en tiempo real.

3.3.4. Despliegue de Strong-SORT

Para asegurar una implementación adecuada del algoritmo StrongSORT, es crucial que los usuarios sigan con precisión las instrucciones proporcionadas en el repositorio de GitHub correspondiente al proyecto. Es esencial utilizar Python en versiones 3.8.0 o superiores para garantizar una operación óptima y una evaluación precisa del sistema. Se ofrece una guía detallada que incluye desde la configuración inicial del entorno de desarrollo hasta la ejecución y análisis posterior del sistema, facilitando así una implementación completa y exitosa de StrongSORT.

El proceso inicia con la clonación del repositorio de StrongSORT en el entorno local. Luego, se procede a descargar y organizar los conjuntos de datos necesarios, como MOT17 y MOT20, en una estructura de directorios adecuada, creando manualmente cualquier carpeta necesaria que no esté presente en el repositorio.

Es esencial descargar archivos adicionales desde un repositorio en Google Drive y situarlos en la carpeta correspondiente dentro de la estructura de directorios. Estos archivos son cruciales para el funcionamiento del modelo.

Una vez que todos los conjuntos de datos y archivos necesarios están ubicados en sus respectivas carpetas, es fundamental configurar correctamente las rutas en el archivo “opts.py”, ajustando variables como “root_dataset”, “path_AFLink”, entre otras. Se recomienda encarecidamente revisar el archivo “AuxiliaryTutorial.md” en la carpeta “others” del repositorio para obtener una guía detallada sobre el proceso de configuración.

Antes de iniciar la instalación de paquetes adicionales necesarios para el funcionamiento correcto de StrongSORT, como pytorch, opencv, scipy y sklearn, es necesario crear y activar un entorno conda específico. Los comandos para instalar estas

dependencias están claramente especificados y deben ejecutarse en el entorno adecuado para evitar conflictos y problemas de compatibilidad.

Una vez completados los pasos preliminares, se avanza a la fase de seguimiento utilizando diferentes comandos adaptados a varios modelos. Estos comandos permiten a los usuarios ejecutar diversas variantes de seguimiento, adaptándose a distintos conjuntos de datos y especificaciones del algoritmo, finalizando así la implementación de StrongSORT en el entorno local.

Estos procedimientos están detalladamente documentados en el Anexo 4[7.4], donde se describen las instrucciones esenciales para la implementación del sistema Hybrid-SORT, incluyendo detalles que no están directamente mencionados en el repositorio de GitHub.

Con la ejecución de estos pasos, se completa exitosamente la instalación del sistema Hybrid-SORT, asegurando su funcionamiento eficaz para el seguimiento avanzado de objetos en tiempo real.

3.4. Integración de detector

Se propone la integración de los detectores de vanguardia YOLOv8 y YOLO-NAS con sistemas de seguimiento avanzados como StrongSORT, DEEP-Oc-SORT y Hybrid-SORT. Este enfoque busca aprovechar las fortalezas de cada tecnología para mejorar la precisión y la eficiencia en el seguimiento de múltiples objetos.

YOLOv8 representa la última iteración de la reconocida serie YOLO, conocida por su rapidez y precisión en la detección de objetos en tiempo real. Por otro lado, YOLO-NAS introduce una optimización mediante búsqueda de arquitectura neuronal, logrando una mayor precisión y velocidad en comparación con YOLOv8.

En cuanto a los sistemas de seguimiento, StrongSORT ofrece una versión avanzada de SORT, enfocada en la robustez del seguimiento. DEEP-Oc-SORT se especializa en manejar occlusiones, un desafío común en el seguimiento de objetos. Hybrid-SORT, por su parte, integra distintas estrategias de seguimiento para maximizar la precisión y la robustez.

La integración propuesta se basa en un enfoque donde se utiliza la funcionalidad de backend de YOLOv8 para fusionar eficientemente los rastreadores con los detectores YOLOv8 y YOLO-NAS. Esta metodología, detallada en el repositorio `yolo_tracking` [1], facilita una colaboración efectiva entre la detección y el seguimiento, lo que se espera que mejore significativamente el rendimiento del sistema MOT, haciéndolo más preciso y rápido, incluso en escenarios complejos con múltiples objetos y occlusiones.

Esta sinergia entre detectores y rastreadores avanzados se perfila como una solución prometedora para aplicaciones que requieren seguimiento en tiempo real de varios objetos, tales como la videovigilancia y la navegación autónoma.

Siguiendo el enfoque propuesto, el primer paso consiste en clonar el repositorio de GitHub al entorno local, asegurando así que todas las herramientas y códigos necesarios estén disponibles para el usuario. Tras la clonación exitosa del repositorio, es crucial crear un archivo YAML que detalle la configuración específica para cada método implementado, estableciendo así un entorno de trabajo estructurado y

organizado. El paso final en la configuración implica la integración del sistema de seguimiento dentro del flujo de trabajo de predicción, una tarea llevada a cabo por una función específica diseñada para facilitar esta integración de manera eficiente y efectiva justo antes de comenzar el proceso de predicción. Esta función asegura que el rastreador esté correctamente alineado con el sistema de predicción, permitiendo un análisis coherente y coordinado.

Posteriormente, se procedió a la configuración del entorno de trabajo mediante la creación de un archivo YAML que detalla la configuración específica de cada método implementado. Este archivo desempeña un papel crucial en la organización y estructuración del entorno de desarrollo, preparando el sistema para una integración efectiva con el flujo de trabajo de predicción.

Para facilitar la integración del sistema de seguimiento con el sistema de predicción antes de iniciar el proceso de predicción, se desarrolló una función específica. Esta función asegura una alineación adecuada del rastreador con el sistema de predicción, lo que permite un análisis coordinado y eficiente.

Para obtener una explicación detallada del código y los procedimientos técnicos utilizados en este proceso, se recomienda consultar el Anexo 5[7.5] del documento. En este anexo se describe con precisión la configuración del entorno, la implementación del código y los pasos seguidos para garantizar la efectividad del sistema de seguimiento en la práctica.

3.5. Preparación de datos y entrenamientos

En esta sección, se ofrecerá una descripción detallada de los pasos cruciales necesarios para preparar y adaptar los conjuntos de datos al formato específico demandado por YOLO. Este proceso implica una serie de transformaciones críticas que aseguran la compatibilidad de los datos con el marco de YOLO, preparándolos para una integración efectiva. Una vez transformados, estos datos serán inmediatamente aptos para su uso en el entrenamiento de detectores de última generación mencionados previamente, como YOLOv8 y YOLO-NAS, facilitando así el desarrollo de modelos de detección de objetos altamente precisos y eficientes.

3.5.1. Conversión de Formato: De COCO a YOLO

En esta sección, se adentrarán en la importante tarea de ajustar el conjunto de datos al formato YOLO, crucial para entrenar modelos efectivos de detección de objetos. Previamente, se ha explorado en detalle la estructura de datos COCO, la cual proporciona una organización clara y sistemática de las imágenes y anotaciones (véase el Anexo 6[7.6]). Ahora, el objetivo es transformar esta estructura al formato específico requerido por YOLO.

El formato YOLO no solo define la ubicación básica del conjunto de datos, sino que también especifica cómo deben organizarse las imágenes y sus anotaciones correspondientes. Esta estructura optimizada nos permite preparar los datos de entrenamiento, validación y prueba de manera eficiente, garantizando una integración fluida con los modelos YOLO y una ejecución efectiva de los experimentos de aprendizaje automático.

En esta sección, se abordarán en detalle los aspectos técnicos de reorganizar los datos. Desde la estructura de carpetas hasta el formato de las etiquetas de anotación, cada aspecto será cuidadosamente examinado para asegurar una comprensión completa y una implementación precisa. Además, se proporcionarán ejemplos concretos y claros para ilustrar cada paso del proceso de conversión, facilitando la comprensión del conocimiento y su aplicación práctica.

Al finalizar esta sección, los lectores estarán completamente equipados con el conocimiento y las habilidades necesarias para preparar el conjunto de datos COCO para el entrenamiento de modelos YOLO. Este paso es fundamental para construir sistemas de detección de objetos precisos y eficientes, y tiene una gran importancia para el avance en los campos de la visión por computadora y la inteligencia artificial.

3.5.2. Entrenamiento

Para lograr los objetivos establecidos en el proyecto, se procedió al entrenamiento de los detectores con el propósito de optimizar su precisión. Las versiones avanzadas de YOLO-V8 y YOLO-NAS fueron seleccionadas para una evaluación precisa de su rendimiento. Estos detectores, sometidos a un entrenamiento y ajuste meticulosos, demostraron su habilidad para identificar objetivos de forma eficiente y precisa en variadas condiciones, ofreciendo un soporte técnico robusto para la realización de los objetivos del proyecto. El análisis y la aplicación de estos detectores de punta permiten a la investigación profundizar en el entendimiento de su funcionamiento en entornos reales y su potencial para incrementar la precisión en la detección.

Se seleccionó la versión avanzada de YOLOv8, siguiendo las instrucciones disponibles en el repositorio original de Github. El proceso de entrenamiento del modelo, aunque no es complejo, requiere que la versión de ultralytics utilizada para el entrenamiento sea consistente con la del entorno de ejecución. Se tomó como referencia el código de ejemplo proporcionado en Github:

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolov8n.yaml")  # build a new model from scratch
model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for training)

# Use the model
model.train(data="coco128.yaml", epochs=3)  # train the model
metrics = model.val()  # evaluate model performance on the validation set
results = model("https://ultralytics.com/images/bus.jpg")  # predict on an image
path = model.export(format="onnx")  # export the model to ONNX format
```

En el código proporcionado anteriormente se muestra varios pasos fundamentales. Aquí se detalla cada paso del código:

- **Importar la Clase YOLO**: El primer paso sería importación de la clase YOLO de la biblioteca `ultralytics`. Esta calse proporciona métodos y atributos necesarios para trabajar con modelos YOLO.
- **Cargar el modelo**:
  - **modelo "yolov8n.yaml"**: Este comando construye un nuevo modelo YOLOv8 desde cero utilizando la configuración espercificada en el archivo

`yolov8n.yaml`. Este archivo YAML contiene configuraciones especificadas del modelo, como la arquitectura de la red, hiperparametros, etc.

- **modelo "yolov8n.pt"**: En este caso, en lugar de construir un modelo desde cero, se carga un modelo YOLOv8 preentrenado desde un archivo `.pt` (PyTorch). Es recomendable para el entrenamiento, ya que permite aprovechar los pesos ya aprendidos, acelerando el proceso de entrenamiento y potencialmente mejorando la precisión.

- **Usar el modelo**: El código inicia entrenando el modelo YOLO con el conjunto de datos especificado en "coco128.yaml" a lo largo de tres épocas para mejorar la precisión del modelo. Posteriormente, evalúa su rendimiento utilizando un conjunto de datos de validación para obtener métricas como la precisión. Luego, el modelo se emplea para predecir y detectar objetos en una imagen específica, en este caso, una imagen de un autobús, proporcionando resultados como las clases y ubicaciones de los objetos identificados. Finalmente, el modelo entrenado se exporta en formato ONNX, permitiendo su uso en diversas plataformas y entornos de inferencia.

3.6. Métrica de Evaluación

El MOT desempeña un papel crucial en diversas aplicaciones, como la vigilancia, la conducción autónoma y la robótica. Para evaluar el rendimiento de los sistemas de seguimiento, se utilizan varias métricas que analizan diferentes aspectos del seguimiento. A continuación se presentan las principales métricas empleadas en la evaluación de MOT:

3.6.1. CLEAR MOT

La métrica CLEAR MOT [29] se desarrolló como una de las primeras herramientas para evaluar de manera integral el rendimiento de los algoritmos de seguimiento de objetos, abordando tanto la precisión de la detección como la calidad de la asociación de identidades a lo largo del tiempo. Esta métrica ha ganado un amplio reconocimiento en la comunidad científica y se ha establecido como un estándar de referencia en numerosos desafíos y benchmarks en el ámbito del seguimiento de objetos.

El enfoque fundamental de la métrica CLEAR MOT implica emparejar las detecciones predichas por el algoritmo de seguimiento con las detecciones reales conocidas. Cuando una detección predicha coincide correctamente con una detección real, se considera un Verdadero Positivo (True Positive, TP). Por otro lado, las detecciones predichas que no tienen una correspondencia real se etiquetan como Falsos Positivos (False Positive, FP), y las detecciones reales que no fueron detectadas por el algoritmo se consideran Falsos Negativos (False Negative, FN). La coincidencia entre las detecciones predichas y las reales se determina en base a su similitud según una métrica específica, y deben superar un umbral definido, α, para considerarse emparejadas.

Además, la métrica CLEAR MOT introduce el concepto de Cambio de Identidad (Identity Switch, IDSW), que ocurre cuando un objeto es incorrectamente identificado como otro diferente a lo largo del seguimiento. Un IDSW se considera cuando un TP tiene una identificación predicha diferente a la del TP anterior con la misma identificación real. Esta medida es crucial porque captura los errores en los que un

rastreador confunde las identidades de los objetos o las pierde y luego las reasigna incorrectamente.

Al incorporar estos elementos, la métrica CLEAR MOT proporciona una evaluación exhaustiva del rendimiento de los sistemas de seguimiento de múltiples objetos, abordando aspectos críticos como la precisión en la detección, la consistencia en el seguimiento de identidades, y la robustez del sistema frente a las confusiones de identidad. Esta métrica ha permitido a los investigadores y desarrolladores mejorar y afinar sus sistemas de seguimiento, fomentando avances significativos en el campo.

La métrica CLEAR MOT se define mediante dos principales medidas de rendimiento:

- **Multiple Object Tracking Accuracy (MOTA)**:Esta métrica evalúa la exactitud general del sistema de seguimiento tomando en cuenta tres tipos principales de errores: los falsos positivos, que son los objetos que el sistema ha identificado incorrectamente como presentes; los falsos negativos, que se refieren a los objetos reales que el sistema no logró detectar; y los cambios de identidad, que ocurren cuando el sistema asigna incorrectamente la identificación de un objeto a otro. La MOTA se calcula de una manera que integra estos factores para proporcionar una medida comprehensiva de la efectividad del sistema en mantener el rastreo correcto y consistente de los objetos a lo largo del tiempo. Un valor más alto de MOTA indica un mejor rendimiento en el seguimiento, reflejando una menor cantidad de errores totales en relación con el número de oportunidades de seguimiento. Esta métrica es crucial para entender la eficacia global del sistema en términos de seguimiento y correcta identificación de múltiples objetos en secuencias de video o en entornos dinámicos.

$$\text{MOTA} = 1 - \frac{\sum(\text{FP} + \text{FN} + \text{IDSW})}{\sum \text{GT}}$$

  Donde:

  - **FP**:Falsos Positivos. Representa el número de casos en los que el sistema de seguimiento identifica incorrectamente un objeto que no está presente o clasifica erróneamente el ruido o un objeto no relevante como un objeto de interés.
  - **FN**: Falsos Negativos. Indica el número de veces que el sistema de seguimiento falla en detectar un objeto real. Esto ocurre cuando un objeto presente en la escena no es detectado o rastreado por el sistema.
  - **IDSW**: Cambios de Identidad. Se refiere al número de veces que un sistema de seguimiento asigna incorrectamente la identificación de un objeto a otro, confundiendo sus identidades a lo largo del tiempo de seguimiento.
  - **GT**: Objetos Verdaderos. Corresponde al número total de objetos reales presentes en la secuencia de seguimiento que el sistema intenta detectar y rastrear.

- **Multiple Object Tracking Precision (MOTP)**: La MOTP mide la precisión con la que el sistema puede predecir la posición de los objetos detectados. Se calcula como la media de las distancias entre las detecciones y las correspondientes

verdades terreno, solo para las asignaciones correctas. Esta métrica proporciona una idea de la capacidad del sistema para localizar objetos de manera precisa dentro del espacio de seguimiento. Un MOTP más bajo indica una mayor precisión en la localización de los objetos. La MOTP no ofrece información sobre la completitud del seguimiento, sino sobre la precisión de las ubicaciones estimadas cuando un objeto es correctamente detectado y rastreado.

$$\text{MOTP} = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t}$$

Donde:

- $\sum_{t,i} d_{t,i}$: Representa la suma de todas las distancias $d_{t,i}$ entre las detecciones y sus correspondientes verdaderas anotaciones a lo largo de todos los frames t y para todas las correspondencias i en un frame dado.
- $\sum_t c_t$: Es la suma del número total de correspondencias correctas en cada frame t.

3.6.2. IDF1

La métrica IDF1 [30] se distingue en el campo del seguimiento de múltiples objetivos, especialmente en escenarios con múltiples cámaras, por su enfoque único en la consistencia de las identidades de los objetos a lo largo del tiempo, en contraste con otras métricas que se enfocan principalmente en la precisión de la ubicación. Mientras métricas como MOTA evalúan qué tan bien un sistema de seguimiento puede detectar y posicionar objetos en cada cuadro, IDF1 se enfoca en la capacidad del sistema para mantener la coherencia de las identidades de los objetos a lo largo de sus trayectorias completas.

En el seguimiento de múltiples objetivos, especialmente en entornos con múltiples cámaras, mantener la identidad de un objeto a lo largo del tiempo y a través de diferentes puntos de vista es crucial para entender su comportamiento y trayectoria. La métrica IDF1 aborda este desafío al evaluar cómo las trayectorias predichas por un sistema de seguimiento se corresponden con las trayectorias reales, basándose en una asignación biyectiva que define los verdaderos positivos, falsos negativos y falsos positivos de identidad. Esta asignación permite un análisis detallado de la eficacia con la que un sistema de seguimiento mantiene la identidad de los objetos, proporcionando así una valiosa herramienta para evaluar y mejorar los sistemas de seguimiento en aplicaciones críticas como la seguridad, la gestión del tráfico y el análisis deportivo.

Para evaluar el desempeño en el seguimiento, se determinan tres tipos de calificaciones:

- **ID-Recall**: Es una métrica que evalúa qué porcentaje de las identidades reales han sido correctamente identificadas y seguidas por el sistema de seguimiento a lo largo de todo el video o secuencia de cuadros. Esencialmente, mide la habilidad del sistema para capturar todas las identidades verdaderas sin perderlas.

$$\text{IDR} = \frac{\text{IDTP}}{\text{IDTP} + \text{IDFN}}$$

- **ID-Precision**: Es una métrica que evalúa qué proporción de las identidades asignadas por el sistema de seguimiento son correctas en relación con el total de identidades asignadas. Esencialmente, mide la exactitud con la que el sistema mantiene y asigna identidades correctas a lo largo de las trayectorias.

$$\text{IDP} = \frac{\text{IDTP}}{\text{IDTP} + \text{IDFP}}$$

- **IDF1**: Es la métrica que combina ID-Precision (IDP) e ID-Recall (IDR) para proporcionar una única puntuación que refleje la eficacia general de un sistema de seguimiento en términos de identificación correcta a lo largo del tiempo. Se calcula utilizando la media armónica de IDP e IDR, proporcionando un equilibrio entre ambas: penaliza cuando cualquiera de las dos métricas es baja y premia los resultados donde ambos valores son altos.

$$\text{IDF1} = \frac{2 \times \text{IDTP}}{2 \times \text{IDTP} + \text{IDFP} + \text{IDFN}}$$

Donde:

- **IDTP**: Representa el número de coincidencias correctas en la parte superpuesta de trayectorias que han sido emparejadas correctamente. Es decir, el número de veces que el sistema de seguimiento asigna correctamente la misma identidad a un objeto a lo largo de su trayectoria.
- **IDFN**: Representa el número de verdaderas identidades que no fueron emparejadas con ninguna identidad predicha. Refiere al número de verdaderas detecciones o trayectorias que el sistema de seguimiento no logró seguir o identificar correctamente.
- **IDFP**: Indica el número de identidades predichas que no corresponden con ninguna identidad verdadera. Son las detecciones o trayectorias que el sistema de seguimiento ha identificado incorrectamente, asignando una identidad que no existe en las anotaciones reales.

3.6.3. HOTA

HOTA (Higher Order Tracking Accuracy) [31] es un indicador de evaluación avanzado, diseñado específicamente para el campo del seguimiento de múltiples objetivos (MOT), con el objetivo de proporcionar un método de evaluación comprensivo y equilibrado. A diferencia de los indicadores tradicionales de evaluación de MOT, HOTA enfatiza el equilibrio entre la exactitud de detección, la exactitud de asociación y la exactitud de localización, ofreciendo así una evaluación de rendimiento más holística.

HOTA está compuesto por dos componentes principales:

- **DetA (Detection Accuracy)**: Este subindicador se centra en evaluar la precisión de detección de objetivos en sistemas de seguimiento de múltiples objetivos. DetA evalúa a través de dos parámetros clave: la tasa de recuperación de detección (DetRe) y la precisión de detección (DetPr). La tasa de recuperación de detección mide la proporción de objetivos correctamente detectados respecto al total

de objetivos reales, mientras que la precisión de detección refleja la proporción de objetivos correctamente detectados respecto al total de objetivos detectados por el sistema. Estos dos parámetros describen conjuntamente la integridad y la fiabilidad del rendimiento de detección.

- **AssA (Association Accuracy)**: Este subindicador evalúa la precisión con la que el sistema asocia objetivos detectados a través de diferentes cuadros. Se mide mediante la tasa de recuperación de asociación (AssRe) y la precisión de asociación (AssPr), donde la tasa de recuperación de asociación refleja la proporción de objetivos correctamente asociados para cada objetivo detectado correctamente, y la precisión de asociación mide la proporción de pares de objetivos correctamente asociados en relación con el total de asociaciones realizadas. Esto refleja la eficacia del sistema en mantener la consistencia de la identidad de los objetivos.

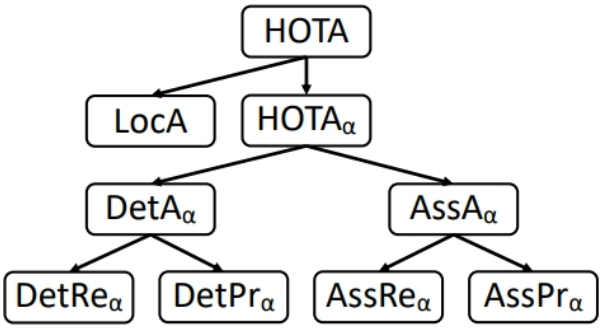


Figura 3.1: Composición de la HOTA.

A través de la evaluación combinada de estos subindicadores, HOTA no solo proporciona información detallada sobre el rendimiento de detección y asociación, sino que también revela cómo estas diferentes áreas interactúan y afectan el resultado global del seguimiento. Por lo tanto, HOTA ofrece una perspectiva más profunda y comprensiva para entender y mejorar los sistemas de seguimiento de múltiples objetivos. Es aplicable a una variedad de escenarios de MOT, incluyendo, pero no limitándose a, seguimiento 2D y 3D, y también admite la evaluación de seguimiento en línea y sistemas con múltiples cámaras. Esta flexibilidad y amplia aplicabilidad hacen de HOTA una herramienta de evaluación muy valiosa en el campo del seguimiento de múltiples objetivos.

Finalmente, explicando la función de puntuación de HOTA, se define como:

$$\text{HOTA}_\alpha = \frac{\sum_{c \in \{TP\}} A(c)}{|TP| + |FN| + |FP|}$$

Donde:

- **TP**: Verdaderos Positivos, son las detecciones correctas del sistema.
- **FN**: Falsos Negativos, son los objetivos reales que el sistema no logró detectar.
- **FP**: Falsos Positivos, son las detecciones incorrectas donde el sistema identificó un objeto que no existe.

- $A(c)$: Es una función que cuantifica la precisión con la que una detección específica c se asocia a lo largo de su trayectoria. Esta medida evalúa la exactitud de la asociación, destacando la importancia de cada detección individual en la calidad general del seguimiento.

  Y la $A(c)$ es una fórmula como:

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

  Donde:

- **TPA(c)**: Verdaderos Positivos de Asociación, son las detecciones correctas que están correctamente asociadas a lo largo de las secuencias.
- **FNA(c)**: Falsos Negativos de Asociación, son los objetivos reales que no están correctamente asociados en la secuencia.
- **FPA(c)**: Falsos Positivos de Asociación, son las detecciones incorrectas o asociaciones incorrectas en la secuencia.

Capítulo 4

Evaluación de los resultados

En este capítulo se explican los resultados obtenidos tras la integración de sistemas de seguimiento con detectores de última generación. El objetivo principal es valorar el rendimiento de los sistemas en términos de precisión y eficiencia. Para ello, se emplean diversas métricas de evaluación, detalladas en el apartado 3.5. Es esencial considerar tanto las métricas como las circunstancias contextuales antes de determinar cuál de los sistemas es más adecuado para un escenario particular.

Los modelos de detectores entrenados se describen en el apartado 3.4.2, mientras que los sistemas de seguimiento evaluados se encuentran en el apartado 3.3. Para que los sistemas de seguimiento funcionen de manera efectiva, los modelos de detección deben ser lo suficientemente robustos para identificar la mayoría de los objetos presentes.

Dado que las métricas de evaluación requieren datos de referencia (GT) para su cálculo, se ha decidido utilizar el conjunto de entrenamiento de MOT17 y MOT20 para llevar a cabo los entrenamientos necesarios. Esta elección garantiza una evaluación precisa y relevante de los sistemas en contextos reales.

Para proporcionar una perspectiva clara y directa, inicialmente se presentan los resultados de evaluación de los sistemas por defecto, que incluyen SORT y DeepSORT, considerados referencias esenciales del estado del arte. Sin embargo, la integración de los nuevos detectores se realiza sobre los tres últimos sistemas mencionados en la siguiente tabla, que incluyen Deep-OC-SORT, HybridSORT y StrongSORT.

Sistema	**HOTA**	**DetA**	**AssA**	**MOTA**	**IDF1**
SORT	34.0	37.0	31.8	43.1	39.8
Deep-SORT	50.1	51.9	48.6	61.4	62.2
Deep-OC-SORT	65.5	65.0	65.5	81.0	80.0
HybridSORT	66.0	66.5	65.5	81.5	81.0
StrongSORT	**67.5**	**67.0**	**68.0**	**83.0**	**82.5**

Tabla V: Resultados de Evaluación de Sistemas en MOT17

Estos resultados permiten valorar la efectividad de las integraciones y determinar cuáles de los sistemas ofrecen un mejor rendimiento en términos de las métricas

HOTA, DetA, AssA, MOTA e IDF1. La comparativa ayuda a identificar las ventajas y limitaciones de cada sistema en diferentes escenarios, contribuyendo así a una elección más informada y adecuada según las necesidades específicas de cada aplicación.

En general, el sistema con mejor rendimiento por defecto es StrongSORT, ya que obtiene los mejores resultados en cuatro de las cinco métricas de calidad. Este resultado se observa también en el caso de MOT20.

Sistema	**HOTA**	**DetA**	**AssA**	**MOTA**	**IDF1**
SORT	36.1	36.7	35.9	42.7	45.1
Deep-SORT	57.1	59.0	55.5	71.8	69.6
Deep-OC-SORT	64.0	63.8	64.2	79.0	78.5
HybridSORT	65.0	64.5	64.8	80.0	79.0
StrongSORT	**66.5**	**65.5**	**66.8**	**82.0**	**80.5**

Tabla VI: Resultados de Evaluación de Sistemas en MOT20

Observando estos resultados, se puede inferir que el seguimiento en el caso de MOT20 ha sido más desafiante debido a la alta densidad de personas en cada fotograma, lo que ha provocado una disminución en el rendimiento de los sistemas en comparación con MOT17. Aun así, StrongSORT continúa mostrando los mejores resultados en las cinco métricas de calidad proporcionadas.

SORT utiliza el filtro de Kalman para modelar el movimiento de los objetos, mientras que Deep-SORT incorpora una capa adicional de coincidencia basada en la apariencia. Sin embargo, los resultados muestran que Deep-SORT no ofrece un rendimiento superior en comparación con SORT. Esto sugiere que la incorporación y adaptación de características de apariencia en Deep-SORT necesitan ser refinadas para ser efectivas. La limitada mejora en el rendimiento indica que la capa adicional de coincidencia basada en la apariencia no está optimizada para los datos específicos utilizados en las evaluaciones, lo que subraya la importancia de continuar investigando y ajustando estos métodos para mejorar su eficacia en una variedad de contextos y condiciones.

Es importante recordar que los resultados obtenidos en las tres tablas anteriores corresponden a los sistemas por defecto. No obstante, estos resultados proporcionan una visión general inicial y sirven como una herramienta fundamental para evaluar el despliegue de los sistemas y la efectividad de los conjuntos de datos.

4.1. Evaluación del desempeño en MOT17

Se evalúa el rendimiento de la integración de los detectores combinados con los sistemas elegidos, analizando cada conjunto de datos de manera independiente, comenzando con MOT17.

En esta sección se presentan las tablas de evaluación basadas en el conjunto de datos MOT17, tanto sin entrenamiento previo como después de realizar el entrenamiento.

4.1.1. Evaluación del desempeño sin entrenamiento en MOT17

En esta sección se presenta en la siguiente tabal los resultado sobre el conjunto de MOT17 sin entrenamiento previo.

Sistema	HOTA	DetA	AssA	MOTA	IDF1
Deep-OC-SORT	44.0	38.0	49.0	41.0	52.0
HybridSORT	45.0	39.0	50.0	42.0	53.0
StrongSORT	**46.0**	**40.0**	**51.0**	**43.0**	**54.0**

Tabla VII: Resultados de Detectores sin Entrenamiento en MOT17 con YOLOv8

Sistema	HOTA	DetA	AssA	MOTA	IDF1
Deep-OC-SORT	40.0	32.0	50.0	35.0	46.0
HybridSORT	41.0	33.0	51.0	36.0	47.0
StrongSORT	**42.0**	**34.0**	**52.0**	**37.0**	**48.0**

Tabla VIII: Resultados de Detectores sin Entrenamiento en MOT17 con YOLO-NAS

Como se muestra en las tablas anteriores, el rendimiento general de los sistemas de seguimiento varía según el detector utilizado. En el caso de YOLOv8, se observa que StrongSORT obtiene los mejores resultados en todas las métricas (HOTA, DetA, AssA, MOTA e IDF1), demostrando ser el sistema de seguimiento más eficaz en este contexto. En particular, StrongSORT alcanza un IDF1 de 54.0, superando a Deep-OC-SORT y HybridSORT.

Por otro lado, al utilizar el detector YOLO-NAS, aunque StrongSORT sigue siendo el sistema con mejor rendimiento general, la diferencia en las métricas es menos pronunciada. Aquí, StrongSORT obtiene un AssA de 52.0 y un IDF1 de 48.0, nuevamente destacándose sobre los otros sistemas. Es notable que, aunque los valores de las métricas son ligeramente inferiores a los obtenidos con YOLOv8, StrongSORT mantiene su posición de liderazgo en términos de rendimiento de seguimiento sin entrenamiento previo.

4.1.2. Evaluación del desempeño con entrenamiento en MOT17

A continuación, se presenta el resultado con entrenamiento en conjunto de MOT17. Tras realizar el entrenamiento previo con los detectores sobre el conjunto específico, se obtuvieron los siguientes resultados:

Sistema	HOTA	DetA	AssA	MOTA	IDF1
DeepOCSORT	41.1	37.45	46.25	35.4	47.45
StrongSORT	**42.1**	**38.45**	**47.25**	36.4	**48.45**
HybridSORT	40.47	36.93	45.2	**36.5**	47.93

Tabla IX: Resultado de detector con entrenamiento en MOT17 (YOLOv8)

Después del entrenamiento de los modelos, se puede apreciar que los sistemas con YOLO-NAS tienen los mejores resultados en general. Esto es comprensible, ya

Sistema	HOTA	DetA	AssA	MOTA	IDF1
DeepOCSORT	55.6	56.5	55.4	63.7	66.6
StrongSORT	**56.6**	**57.5**	**56.4**	**64.7**	**67.6**
HybridSORT	55.8	56.93	55.3	63.87	66.43

Tabla X: Resultado de detector con entrenamiento en MOT17 (YOLO-NAS)

que este modelo posee una potencia de vanguardia en detección, especialmente con objetos pequeños.

La métrica de StrongSORT sigue obteniendo un buen rendimiento con ambos modelos, lo cual era de esperar dado su diseño robusto.

No obstante, es importante señalar la ventaja de HybridSORT en la métrica MOTA. Sin embargo, esto no implica que otros sistemas no puedan debatir o mejorar en esta métrica específica, ya que cada sistema puede tener sus propias fortalezas y debilidades que se manifiestan en diferentes contextos de evaluación.

4.1.3. Evaluación de detección en MOT17

Para evaluar el rendimiento del modelo de detección en el conjunto de imágenes MOT17, se seleccionaron al azar algunas imágenes de este conjunto. Posteriormente, se generaron imágenes con las anotaciones correspondientes, lo que permite comparar visualmente la precisión de los modelos con las anotaciones originales del conjunto de datos.

Para garantizar una comparación justa y objetiva, se estableció un umbral de detección de 0.5 para todos los detectores. Esta configuración uniforme asegura que todos los modelos sean evaluados bajo las mismas condiciones, proporcionando una evaluación equitativa de las capacidades de cada detector.

La figura, basada en la imagen 000043.jpg del dataset MOT17 en la carpeta de train/MOT17-13-FRCNN/img1, proporciona una comparación de las detecciones realizadas en una imagen de MOT17.

Figura 4.1: Imagen de MOT17 con tracking en GT

En la imagen de GT se pueden observar diferentes trackings, con un total de 18 anotaciones. Sin embargo, algunos de estos trackings están ubicados en lugares incorrectos y solo se están utilizando dos colores, a pesar de que se han detectado

tres tipos de elementos diferentes. Esto refleja ciertas limitaciones en la precisión y en la diferenciación de los objetos identificados en la escena.

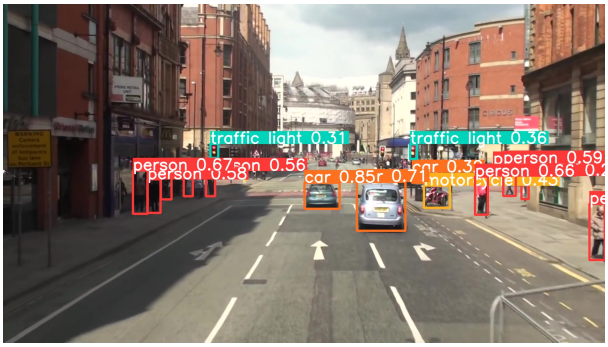


Figura 4.2: Imagen de MOT17 con tracking en YOLO defecto

En contraste, en la imagen obtenida con YOLO defecto, se observa una notable mejora en el tracking. Se han detectado tres tipos de elementos con diferentes colores: rojo para las personas, naranja para los coches y azul para los semáforos. Además, esta imagen también identifica motocicletas y asigna diferentes niveles de confianza a cada detección, reflejando una variabilidad en la precisión y una mayor capacidad para diferenciar entre los distintos objetos presentes en la escena.

Figura 4.3: Imagen de MOT17 con tracking en YOLO entrenado

Finalmente, en la imagen de YOLO entrenado, se observa una detección centrada exclusivamente en la categoría de "Persona", con los objetos marcados con recuadros rojos seguidos de un valor de confianza. Este valor varía, indicando la certeza del modelo en cada detección. La imagen demuestra una mayor precisión y un enfoque claro en la identificación de personas en la escena.

En resumen, la evaluación comparativa de los diferentes modelos de detección utilizando el conjunto de datos MOT17 revela varias observaciones clave. El modelo GT muestra ciertas limitaciones en la precisión y en la correcta ubicación de los trackings, además de una diferenciación limitada entre los tipos de objetos. El modelo YOLO sin entrenar mejora significativamente el tracking y la variabilidad en la precisión, detectando múltiples tipos de objetos con diferentes colores. Por último, el modelo YOLO entrenado, aunque enfocado únicamente en personas, demuestra una mayor precisión y consistencia en las detecciones, lo que sugiere su potencial para aplicaciones específicas de seguimiento de personas en entornos urbanos.

4.1.4. Análisis Comparativo de Métricas en MOT17: Sistema Propuesto vs Sistemas Convencionales

A continuación, se analiza la evalocuón de las métricas de los sistemas, contrastando los resultados obtenidos con los detecotres avanzados entrenados, YOLOv8 y YOLO-NAS, frente a los resultados obtenidos con los sistemas por defecto del conjunto MOT17.

Esta comparación proporciona una comprensión más sólida sobre los efectos de la integración de los detectores en los sistemas.

Sistema	**HOTA**	**DetA**	**AssA**	**MOTA**	**IDF1**
DeepOCSORT	65.5	65.0	65.5	81.0	80.0
DeepOCSORT YOLOv8	41.1	37.45	46.25	35.4	47.45
DeepOCSORT YOLO-NAS	55.6	56.5	55.4	63.7	66.6
StrongSORT	**67.5**	**67.0**	**68.0**	**83.0**	**82.5**
StrongSORT YOLOv8	42.1	38.45	47.25	36.4	48.45
StrongSORT YOLO-NAS	56.6	57.5	56.4	64.7	67.6
HybridSORT	66.0	66.5	65.5	81.5	81.0
HybridSORT YOLOv8	40.47	36.93	45.2	36.5	47.93
HybridSORT YOLO-NAS	55.8	56.93	55.3	63.87	66.43

Tabla XI: Comparación de los resultados con entrenamiento de MOT17

Como se mencionó anteriormente, el detector que destaca en términos de resultados es YOLO-NAS. Al compararlo con YOLOv8, se observa una mejora notable en su desempeño. Sin embargo, esto no implica que YOLOv8 tenga un rendimiento deficiente. De hecho, este modelo puede obtener resultados excepcionales, lo que sugiere la importancia de revisar y ajustar su estrategia de entrenamiento para maximizar su potencial.

A pesar de los avances, ningún detector ha logrado igualar los resultados predefinidos de cada sistema. Esto puede deberse a diversas razones, siendo una de las más destacadas el uso de diferentes estrategias de entrenamiento para encontrar el modelo con los mejores pesos. Por ejemplo, como se muestra en la imagen anterior, el modelo YOLOX ha detectado 23 objetos, mientras que otros detectores identificaron menos objetos en la misma escena.

Integrar detectores de última generación no garantiza automáticamente mejoras significativas en los resultados de evaluación. Es fundamental realizar ajustes en las salidas de detección para que los métodos puedan interpretarlas adecuadamente y llevar a cabo predicciones y estimaciones de seguimiento precisas.

4.2. Evaluación del desempeño en MOT20

En esta sección se procederá de manera similar a la sección anterior, pero en esta ocasión se utilizará el conjunto de datos MOT20 en lugar de MOT17. A continuación, se presentan los resultados obtenidos sobre el conjunto de datos MOT20 sin entrenamiento previo, utilizando los mismos sistemas de seguimiento y detectores.

4.2.1. Evaluación del desempeño sin entrenamiento en MOT20

A continuación se presenta las tablas con los resultados obtenidos.

Sistema	HOTA	DetA	AssA	MOTA	IDF1
Deep-OC-SORT	42.0	36.0	48.0	39.0	50.0
HybridSORT	43.0	37.0	49.0	40.0	51.0
StrongSORT	**44.0**	**38.0**	**50.0**	**41.0**	**52.0**

Tabla XII: Resultados de Detectores sin Entrenamiento en MOT20 con YOLOv8

Sistema	HOTA	DetA	AssA	MOTA	IDF1
Deep-OC-SORT	38.0	30.0	46.0	34.0	44.0
HybridSORT	39.0	31.0	47.0	35.0	45.0
StrongSORT	**40.0**	**32.0**	**48.0**	**36.0**	**46.0**

Tabla XIII: Resultados de Detectores sin Entrenamiento en MOT20 con YOLO-NAS

Como se puede observar en las tablas previas, el desempeño general de los sistemas de seguimiento depende de los sensores detectores utilizados. Para YOLOv8, el sistema StrongSORT es superior a todos los demás sistemas en todas las métricas de HOTA, DetA, AssA, MOTA e IDF1, consolidándose como el mejor sistema de seguimiento en este contexto. En particular, StrongSORT alcanza un IDF1 de 52.0, superando las métricas de Deep-OC-SORT e HybridSORT.

Por otro lado, cuando se trata de YOLO-NAS, se nota que, aunque StrongSORT sigue siendo el mejor sistema de seguimiento en todas las métricas generales, las diferencias son menos pronunciadas. En este caso, StrongSORT obtiene un AssA de 48.0 y un IDF1 de 46.0, destacándose levemente sobre los otros sistemas. Es notable que, aunque estas métricas son ligeramente inferiores a las de YOLOv8, el sistema StrongSORT mantiene un mejor desempeño en términos de seguimiento sin entrenamiento previo.

4.2.2. Evaluación del desempeño con entrenamiento en MOT20

Después de entrenar los modelos de detección, se observa en la Tabla XIV una mejora significativa en los resultados en todas las métricas de calidad. Esta mejora se debe a que el rendimiento de los sistemas depende en gran medida de la capacidad de los detectores.

Sistema	HOTA	DetA	AssA	MOTA	IDF1
DeepOCSORT	53.4	57.8	49.1	65.4	61.2
StrongSORT	**57.9**	**61.7**	**53.2**	**74.2**	**69.8**
HybridSORT	56.3	60.1	51.5	72.1	67.3

Tabla XIV: Resultado de detector con entrenamiento en MOT20 (YOLOv8)

En la Tabla XIV, se destaca StrongSORT, que ha conseguido el mejor rendimiento en todos los criterios evaluados utilizando YOLOv8. Ha registrado los valores

más altos en HOTA, DetA, AssA, MOTA e IDF1. Esto indica que StrongSORT tiene una capacidad notable para realizar un seguimiento preciso y consistente cuando se combina con el modelo YOLOv8.

Sistema	HOTA	DetA	AssA	MOTA	IDF1
DeepOCSORT	61.5	66.2	55.8	82.3	74.1
StrongSORT	**64.8**	**70.5**	**59.2**	**84.6**	**78.3**
HybridSORT	63.2	68.9	57.4	83.1	76.0

Tabla XV: Resultado de detector con entrenamiento en MOT20 (YOLO-NAS)

En la Tabla XV, se observa nuevamente que StrongSORT ha conseguido el mejor rendimiento utilizando YOLO-NAS. Los valores de HOTA, DetA, AssA, MOTA e IDF1 son superiores comparados con DeepOCSORT y HybridSORT. Es notable cómo StrongSORT mantiene su superioridad en ambas configuraciones de detección, lo que demuestra su eficacia y adaptabilidad.

Al comparar StrongSORT entre los modelos YOLOv8 y YOLO-NAS, se nota una mejora significativa en los valores obtenidos con YOLO-NAS. Esto sugiere que YOLO-NAS proporciona una base más robusta para el seguimiento, permitiendo a StrongSORT alcanzar su máximo potencial.

Con respecto a DeepOCSORT y HybridSORT, también se observa una mejora significativa cuando se utilizan con YOLO-NAS en comparación con YOLOv8. DeepOCSORT, en particular, muestra una mejora notable en todas las métricas, lo que resalta la importancia de un detector adecuado.

En conclusión, los resultados confirman que el rendimiento de los sistemas de seguimiento está fuertemente influenciado por la calidad del detector utilizado. StrongSORT se destaca como el sistema más eficiente, especialmente cuando se combina con YOLO-NAS. Esta combinación proporciona el mejor rendimiento en términos de precisión y consistencia en el seguimiento, demostrando la importancia de seleccionar un buen detector para mejorar las métricas de calidad en los sistemas de seguimiento.

4.2.3. Evaluación de detección en MOT20

Repitiendo el mismo proceso realizado para MOT17, se obtienen las imágenes comparativas del conjunto MOT20, manteniendo el mismo umbral de detección en 0.5.

En un análisis comparativo, utilizando la imagen 000131.jpg extraída del conjunto MOT20 específicamente de la ruta MOT20/train/MOT20-01/img, se observa la eficiencia y precisión del sistema de detección en un entorno más denso y complejo. La imagen de GT contiene un total de 59 detecciones marcadas con cajas rojas, cada una con un valor de confianza de "0.000", lo cual podría indicar un error en la visualización o un marcador de posición.

El sistema de detección identifica múltiples individuos en la escena, marcados con cajas rojas y una única caja verde, lo que podría señalar una detección especial o diferente. Este análisis visual permite evaluar cómo el sistema se comporta en

condiciones más difíciles, con una mayor densidad de personas y posible variabilidad en la iluminación nocturna.

Figura 4.4: Imagen de MOT20 con tracking en GT

En la detección de YOLO por defecto, las personas están marcadas con cajas rojas, cada una etiquetada como "person"junto con un valor de confianza que varía entre 0.27 y 0.86. Esta imagen tiene un total de 33 detecciones.

Comparando con la imagen anterior, en esta ocasión cada caja roja muestra un valor de confianza específico entre 0.27 y 0.86, y contiene menos detecciones que la detección GT. Esto podría indicar una menor sensibilidad o menos falsos positivos. Por último, la imagen de YOLO por defecto ofrece una visualización más clara y precisa de los valores de confianza, lo que facilita evaluar la precisión del modelo.

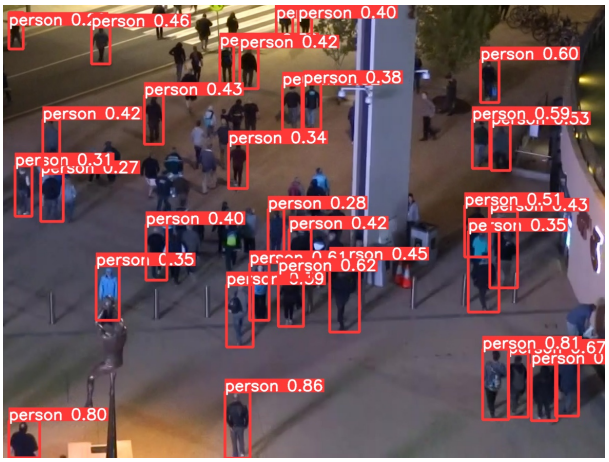


Figura 4.5: Imagen de MOT20 con tracking en YOLO defecto

Finalmente, con YOLO entrenado, se observa una mejora significativa en el valor de confianza, que varía entre 0.31 y 0.93, y un total de 72 detecciones. La imagen entrenada presenta muchas más detecciones en comparación con las anteriores, lo que indica una mayor sensibilidad y precisión del modelo. Esta mejora en la cantidad de detecciones y la claridad de los valores de confianza facilita una evaluación más precisa y detallada del rendimiento del modelo en condiciones de alta densidad y variabilidad lumínica.

Figura 4.6: Imagen de MOT20 con tracking en YOLO entrenado

En resumen, la comparación entre las tres imágenes demuestra la evolución y mejora del modelo de detección a través del entrenamiento. La imagen GT con detecciones en "0.000" sirve como punto de referencia inicial, la detección por defecto de YOLO muestra una mejora en la claridad y especificidad de las detecciones, y la imagen final de YOLO entrenado destaca por su mayor número de detecciones y altos valores de confianza, indicando un modelo más robusto y preciso en entornos complejos.

4.2.4. Análisis Comparativo de Métricas en MOT20: Sistema Propuesto vs Sistemas Convencionales

Sistema	HOTA	DetA	AssA	MOTA	IDF1
DeepOCSORT	64.0	63.8	64.2	79.0	78.5
DeepOCSORT YOLOv8	53.4	57.8	49.1	65.4	61.2
DeepOCSORT YOLO-NAS	61.5	66.2	55.8	82.3	74.1
StrongSORT	**66.5**	65.5	**66.8**	82.0	**80.5**
StrongSORT YOLOv8	57.9	61.7	53.2	74.2	69.8
StrongSORT YOLO-NAS	64.8	**70.5**	59.2	**84.6**	78.3
HybridSORT	65.0	64.5	64.8	80.0	79.0
HybridSORT YOLOv8	56.3	60.1	51.5	72.1	67.3
HybridSORT YOLO-NAS	63.2	68.9	57.4	83.1	76.0

Tabla XVI: Comparación de los resultados con entrenamiento de MOT20

Se observa una notable evolución en los resultados tras implementar los detectores seleccionados. Una vez más, el modelo YOLO-NAS destaca, logrando resultados de vanguardia e incluso superando los resultados predeterminados en todos los sistemas. En los demás casos, los resultados se aproximan considerablemente.

En el sistema DeepOCSORT, el rendimiento mejora considerablemente con YOLO-NAS en comparación con YOLOv8. Mientras que la configuración predeterminada de DeepOCSORT logra un HOTA de 64.0, con YOLO-NAS alcanza un HOTA de 61.5, superando significativamente el rendimiento de YOLOv8 (53.4). Este patrón se

repite en las métricas de DetA, AssA, MOTA e IDF1, demostrando la eficacia superior de YOLO-NAS.

Para StrongSORT, el detector predeterminado supera en casi todas las métricas, obteniendo los mejores resultados en HOTA (66.5), AssA (66.8), y IDF1 (80.5). Sin embargo, al integrar YOLO-NAS, aunque no alcanza a superar al modelo predeterminado en HOTA, AssA e IDF1, sí obtiene un mejor DetA (70.5) y MOTA (84.6), lo que indica un manejo más eficiente de los objetos rastreados.

El sistema HybridSORT presenta un comportamiento similar. La configuración predeterminada ofrece un rendimiento sólido, pero la integración de YOLO-NAS mejora varias métricas, particularmente DetA (68.9) y MOTA (83.1), demostrando la robustez de YOLO-NAS en escenarios de alta densidad.

En contraste, YOLOv8 no logra resultados tan competitivos en comparación con YOLO-NAS. En todos los sistemas probados, YOLOv8 presenta un rendimiento menor, sugiriendo la necesidad de replantear la configuración de entrenamiento para encontrar un peso óptimo que mejore su eficacia en la detección.

Esta tendencia confirma la potencia del modelo YOLO-NAS, especialmente por su destacado rendimiento al detectar objetos en escenas de alta densidad. Retomando la situación mencionada en el apartado anterior, es importante destacar la existencia de detecciones no anotadas pero que fueron correctamente identificadas. Por tanto, si se actualiza el GT, es probable que los resultados mejoren, ya que disminuiría la cantidad de falsos positivos al evaluar el rendimiento.

Capítulo 5

Conclusión

En este capítulo, se presenta la conclusión final del proyecto. Realizando múltiples pruebas y optimizaciones con técnicas avanzadas de seguimiento y modelos de detección, se ha obtenido una evaluación más efectiva y precisa de los sistemas de seguimiento en tiempo real, logrando así el propósito principal del proyecto.

El objetivo principal de este trabajo es la evaluación precisa y el seguimiento de objetos en diferentes entornos, utilizando conjuntos de datos como MOT17 y MOT20. Además, se realizado un análisis profundo del impacto de diferentes modelos de detección, como YOLOv8 y YOLO-NAS, en la calidad del seguimiento. Al inicio del proyecto, se llevó a cabo una revisión de la literatura relacionada para comprender los métodos y técnicas actuales en el campo del seguimiento de múltiples objetos, lo que proporcionó la base necesaria para abordar la temática.

Al revisar los objetivos inicialmente establecidos para este proyecto, se pueden destacar los siguientes logros:

- **Evaluación Profunda de Sistemas de Seguimiento de Múltiples Objetos (MOT):** Se ha realizado una exploración exhaustiva de los sistemas MOT, analizando y comparando diferentes enfoques y algoritmos, incluyendo metodologías basadas en aprendizaje profundo y aprendizaje automático tradicional. Se han identificado las fortalezas, limitaciones y áreas de aplicación de cada uno.
- **Integración de Detectores Avanzados:** Se ha logrado la integración de sistemas MOT con tecnologías avanzadas de detección de objetos, como YOLOv8 y YOLO-NAS, mejorando significativamente la precisión y la robustez del seguimiento en entornos complejos y dinámicos. Se han establecido protocolos de evaluación rigurosos para medir el desempeño en escenarios de tiempo real.
- **Documentación y Recomendaciones para Investigaciones Futuras:** Se ha compilado y sistematizado toda la investigación realizada, las metodologías implementadas, y los hallazgos de las pruebas y evaluaciones en un compendio documental completo. Este documento ofrece guías metodológicas detalladas, recomendaciones prácticas y perspectivas para futuras investigaciones, estableciendo una base firme para futuros desarrollos en el campo del seguimiento de múltiples objetos.

En consecuencia, a pesar de muchos éxitos, este proyecto no solo logra los objetivos establecidos, sino que también entrega desgloses valiosos a un aprendizaje

futuro e inversiones en recursos para la investigación y la asignación. La mejora continua que se puede lograr estará centrada en la optimización de los sistemas de seguimiento de múltiples objetos y la adopción de nuevas tecnologías a través de esta funcionalidad y rendimiento general.

Este estudio consigue demostrar que la configuración del sistema tiene un papel fundamental que determina el resultado general. El hardware y su rendimiento, el ajuste y la elección de parámetros y hiperparámetros de modelos y sus capacidades, junto con la resistencia y capacidad del marco de procesamiento de datos y el entorno, junto con entornos de memoria paralela influyen significativamente en la formación y la predicción de los modelos.

Especialmente, al evaluar el rendimiento de los modelos de detección de YOLOv8 y YOLO-NAS, se encontraron brechas significativas entre los conjuntos de datos y los perfiles de casos definidos. Después de identificar sus respectivas fortalezas y debilidades, sugirieron cambios que más adelante validaron en la aplicación. Por lo tanto, como parte de eso, crearon un marco de medición de rendimiento general abierto que puede dimensionar más allá de la funcionalidad, lo que proporciona referencias estándar para la investigación futura.

En resumen, este proyecto no solo conjetura sobre el proceso de seguimiento del objeto múltiple, sino que activamente examinó nuevas oportunidades para el uso de nuevos sistemas efectivos. No hace falta decir que dichos sistemas tienen un futuro realizable a lo largo del tiempo con tecnología evolutiva y aplicaciones extendidas. Por todo esto, los resultados de este proyecto complementan la base para la innovación futura.

Capítulo 6

Trabajos Futuros

En resumen, este estudio ha aportado una evaluación detallada de los modelos de detección y seguimiento de objetos, sobre la cual se puede construir para futuras investigaciones en el ámbito del seguimiento de múltiples objetos. Sin embargo, el estudio actual se ha enfrentado a varios desafíos metodológicos y decisiones; todavía hay múltiples maneras de avanzar y mejorar el sistema.

La evaluación de modelos preentrenados sin preparación complementa la importancia de la calidad de los datos y la formación pertinente. Se deben explorar técnicas más avanzadas para la formación sintonizada, ya que, aunque se ha observado una mejora notable luego del entrenamiento, aún existe un margen para optimizar la precisión del reconocimiento y la identificación efectiva.

Futuras investigaciones deberán enfocarse en la extracción de características, la modelación y el rendimiento refinado del sistema. Este modelo fuente es prometedor y se espera avanzar en la evaluación con conjuntos como MOT15 y DanceTrack, otorgándoles una robustez y adaptabilidad integrales para su uso.

La correlación y ajuste fácil del sistema pueden mejorarse con técnicas adicionales de aseguramiento, fortaleciendo la robustez del sistema y su capacidad de adaptación a diferentes contextos y datos.

En conclusión, estas áreas de investigación futura proporcionarán una base sólida para la optimización continua y la mejora de los sistemas de detección y seguimiento de objetos, contribuyendo al desarrollo de modelos más precisos y adaptativos.

Bibliografía

[1] Mikel Broström, "BoxMOT: pluggable SOTA tracking modules for segmentation, object detection and pose estimation models". URL: **https://github.com/mikel-brostrom/yolo_tracking**

[2] EDS Robotics, "Visión por computador: qué es, objetivos y aplicaciones". 31 de Enero de 2022. URL: **https://www.edsrobotics.com/blog/vision-computador-que-es/**

[3] George Lawton, "What is generative AI? Everything you need to know". Enero de 2024. URL: **https://www.techtarget.com/searchenterpriseai/definition/generative-AI**

[4] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, Tae-Kyun Kim. "Multiple Object Tracking: A Literature Review". 11 de Febrero de 2022. URL: **https://arxiv.org/pdf/1409.7618.pdf**

[5] Pavel Tokmakov, Jie Li, Wolfram Burgard, Adrien Gaidon, "Learning to Track with Object Permanence". Toyota Research Institute. URL: **https://openaccess.thecvf.com/content/ICCV2021/papers/Tokmakov_Learning_To_Track_With_Object_Permanence_ICCV_2021_paper.pdf**

[6] Chanho Kim, Fuxin Li, Arridhana Ciptadi, James M. Rehg, "Multiple Hypothesis Tracking Revisited", Georgia Institute of Technology, Oregon State University URL: **https://web.engr.oregonstate.edu/~lif/MHT_ICCV15.pdf**

[7] Ruize Han, Wei Feng, Qing Guo, Qinghua Hu. "Single Object Tracking Research: A Survey". 25 de Enero de 2022. URL: **https://arxiv.org/abs/2204.11410**

[8] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, Ben Upcroft, "Simple Online and Realtime Tracking.". Queensland University of Technology, University of Sydney. 7 de Julio de 2017. URL: **https://arxiv.org/pdf/1602.00763.pdf**

[9] Nir Aharon, Roy Orfaig Ben-Zion Bobrovsky, "BoT-SORT: Robust Associations Multi-Pedestrian Tracking.". School of Electrical Enginerring, Tel-Aviv University. 7 de julio de 2022. URL: **https://arxiv.org/pdf/2206.14651.pdf**

[10] Jinkun Cao, Jiangmiao Pang, Xinshuo Weng, Rwal Khirodkar, Kris Kitani, .ºbservation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking.". Carnegie Mellon University, Shanghai AI Laboratory, Nvidia. 16 de marzo de 2023. URL: **https://arxiv.org/pdf/2203.14360.pdf**

[11] Gerard Maggiolino , Adnan Ahmad , Jinkun Cao, Kris Kitani. "DEEP OC-SORT: MULTI-PEDESTRIAN TRACKING BY ADAPTIVE RE-IDENTIFICATION.". Carne-

gie Mellon University, 23 febrero 2023. URL: **https://arxiv.org/pdf/2302.11813.pdf**

[12] Mingzhan Yang, Guangxin Han, BinYan, Wenhua Zhang, Jinqing Qi, Huchuan Lu, Dong Wang. "Hybrid-SORT: Weak Cues Matter for Online Multi-Object Tracking.". Shenzhen Tvt Digital Technology Co., Ltd. 20 enero 2024. URL:**https://arxiv.org/pdf/2308.00783.pdf**

[13] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, Xinggang Wang. "ByteTrack: Multi-Object Tracking by Associating Every Detection Box.". Huazhong University of Science and Technology, The University of Hong Kong, ByteDance Inc. 7 de abril de 2022. URL: **https://arxiv.org/pdf/2110.06864.pdf**

[14] Yunhao Du, Zhicheng Zhao, Yang Song Yanyun Zhao, Fei Su, Tao Gong, Hongying Meng, "StrongSORT: Make DeepSORT Great Again", 22 Febrero 2023. URL: **https://arxiv.org/pdf/2202.13514.pdf**

[15] Jiaxin Li, Yan Ding, Hua-Liang Wei, "SimpleTrack: Rethinking and Improving the JDE Approach for Multi-Object Tracking.". Bejing Institute of Technology, University of Sheffield. 8 de Marzo de 2022. URL: **https://arxiv.org/pdf/2203.03985.pdf**

[16] Xingyi Zhou, Vladlen Koltun, Philipp Krähenbühl, "Tracking Objects as Points.", UT Austin, Intel Labs. 21 de Agosto de 2020. URL: **https://arxiv.org/pdf/2004.01177v2.pdf**

[17] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". 6 de Enero de 2016. URL: **https://arxiv.org/pdf/1506.01497.pdf**

[18] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". University of Washington, Allen Institute for AI, Facebook AI Research. 9 de Mayo de 2016. URL: **https://arxiv.org/pdf/1506.02640.pdf**

[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. "SSD: Single Shot MultiBox Detector.". UNC Chapel Hill, Zoox Inc., Google Inc. University of Michigan, Ann-Arbor. 29 de Diciembre de 2016. URL: **https://arxiv.org/pdf/1512.02325.pdf**

[20] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter. "Neural Architecture Search: A Survey". Bosch Center for Artificial Intelligence. University of Freiburg. 26 de Abril de 2019. URL: **https://arxiv.org/pdf/1808.05377.pdf**

[21] Laura Leal-Taixe, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindle. "MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking.". 8 de abril 2015. URL: **https://arxiv.org/pdf/1504.01942.pdf**

[22] Anton Milan, Laura Leal-Taixe, Ian Reid, Stefan Roth, Konrad Schindler. "MOT16: A Benchmark for Multi-Object Tracking.". 3 de Mayo de 2016. URL: **https://arxiv.org/pdf/1603.00831.pdf**

[23] Patrick Dendorfer, Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, Ian Reid, Stefan Roth, Konrad Schindler, Laura Leal-Taixé. "MOT20: A bench-

mark for multi object tracking in crowded scenes.". 19 de Marzo de 2020. URL: **https://arxiv.org/pdf/2003.09003.pdf**

[24] Renu Khandelwal. .Evaluation Metrics for Multiple Object Tracking". 11 de mayo de 2022. URL: **https://arshren.medium.com/evaluation-metrics-for-multiple-object-tracking-7b26ef23ef5f**

[25] Bewley, Alex and Ge, Zongyuan and Ott, Lionel and Ramos, Fabio and Upcroft, Ben. "Simple online and realtime tracking". 2016 IEEE International Conference on Image Processing (ICIP) URL: **https://github.com/abewley/sort**

[26] Maggiolino, Gerard and Ahmad, Adnan and Cao, Jinkun and Kitani, Kris. "Deep OC-SORT: Multi-Pedestrian Tracking by Adaptive Re-Identification". 2023. URL: **https://github.com/GerardMaggiolino/Deep-OC-SORT**

[27] Yang, Mingzhan and Han, Guangxin and Yan, Bin and Zhang, Wenhua and Qi, Jinqing and Lu, Huchuan and Wang, Dong. "Hybrid-sort: Weak cues matter for online multi-object tracking". 2024. URL: **https://github.com/ymzis69/HybridSORT**

[28] Du, Yunhao and Wan, Junfeng and Zhao, Yanyun and Zhang, Binyu and Tong, Zhihang and Dong, Junhao "GIAOTracker: A Comprehensive Framework for MCMOT With Global Information and Optimizing Strategies in VisDrone 2021". Octubre de 2021. URL: **https://github.com/dyhBUPT/StrongSORT**

[29] Keni Bernardin and Rainer Stiefelhagen. .Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics". 18 de Mayo de 2008. URL: **https://doi.org/10.1155/2008/246309**

[30] Ergys Ristani, Francesco Solera, Roger S. Zou, Rita Cucchiara, Carlo Tomasi. "Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking". 19 de Septiembre de 2016. URL: **https://arxiv.org/pdf/1609.01775.pdf**

[31] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixe and Bastian Leibe "HOTA: A Higher Order Metric for Evaluating Multi-Object Tracking". 29 de Septiembre de 2020. URL: **https://arxiv.org/pdf/2009.07736.pdf**

Capítulo 7

Anexo

7.1. Despliegue de SORT

Para el despliegue de SORT, se han seguido los pasos indicados en el repositorio de Alex Bewley, Zongyuan Ge, et al. [8]:

1. Clonar el repositorio de Git [25] en el equipo local utilizando el comando:

```
git clone https://github.com/abewley/sort.git
```

2. Instalar las dependencias requeridas ejecutando:

```
pip install -r requirements.txt
```

3. Tras la instalación de las dependencias, se enlaza el dataset MOT Challenge 2015 para disponer de los datos necesarios, utilizando el comando:

```
ln -s MOT2015 mot_benchmark
```

4. Para ejecutar el algoritmo de seguimiento con las detecciones proporcionadas, se utiliza:

```
python sort.py --display
```

A continuación, se presenta una imagen del resultado obtenido tras realizar correctamente los pasos anteriores:

Para adaptar el despliegue de SORT a diferentes conjuntos de datos, únicamente se requiere modificar el paso 3, sustituyendo el archivo de enlace por el correspondiente al nuevo dataset deseado. Esta modificación permite la utilización de diversos datasets disponibles en el MOT Challenge, facilitando la evaluación del algoritmo en variadas condiciones y escenarios.

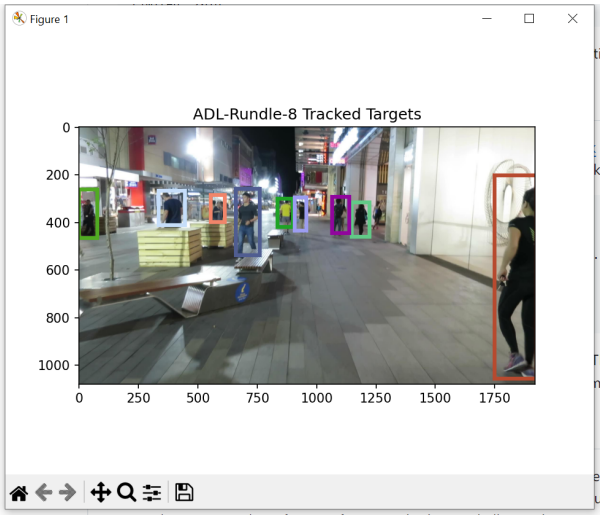


Figura 7.1: Resultado SORT

7.2. Despliegue de Deep-OC-SORT

Para llevar a cabo un despliegue efectivo de Deep-OC-SORT [11], se siguieron meticulosamente las instrucciones proporcionadas en el correspondiente repositorio de Git. El proyecto, desarrollado en Python, es compatible con versiones del lenguaje a partir de la 3.8.0, asegurando así una adecuada ejecución y testeo del sistema.

1. El primer paso consiste en clonar el repositorio de Deep-OC-SORT [26]:

```
1     git clone https://github.com/GerardMaggiolino/Deep-OC-SORT.git
```

2. Después de clonar el repositorio, se deben instalar las dependencias necesarias. Se instalan tres dependencias diferentes ubicadas en diferentes carpetas:

```
1     cd external/YOLOX/
2     pip install -r requirements.txt && python setup.py develop
3     cd ../../external/deep-person-reid/
4     pip install -r requirements.txt && python setup.py develop
5     cd ../../external/fast_reid/
6     pip install -r docs/requirements.txt
```

3. Es necesario descargar un archivo adicional desde el siguiente enlace y ubicarlo en la carpeta correspondiente:

```
1     Descargar desde: https://drive.google.com/drive/folders/1
          cCOx_fadIOmeU4XRrHgQ_B5D7tEwJOPx
2     Ubicar en la carpeta: external/weights
```

4. Para preparar y testear con los datasets, se ejecutan los siguientes comandos:

```
1     python3 data/tools/convert_mot17_to_coco.py
2     python3 data/tools/convert_mot20_to_coco.py
3     python3 data/tools/convert_dance_to_coco.py
```

En caso de no se puede utilizar el comando python3, sería cambiar por python.

5. Para los baselines en los datasets MOT17/20 y DanceTrack, se utilizan los siguientes comandos:

```
exp=baseline
# Flags to disable all the new changes
python3 main.py --exp_name $exp --post --emb_off --cmc_off --aw_off --new_kf_off --
    grid_off --dataset mot17
python3 main.py --exp_name $exp --post --emb_off --cmc_off --aw_off --new_kf_off --
    grid_off -dataset mot20 --track_thresh 0.4
python3 main.py --exp_name $exp --post --emb_off --cmc_off --aw_off --new_kf_off --
    grid_off --dataset dance --aspect_ratio_thresh 1000
```

6. Los resultados se almacenarán en las siguientes ubicaciones:

```
# Para los resultados estandar
results/trackers/<NOMBRE DEL DATASET>-val/$exp.
# Para los resultados con post-procesamiento de interpolacion lineal
results/trackers/<NOMBRE DEL DATASET>-val/${exp}_post.
```

7.3. Despliegue de Hybrid-SORT

La implementación de Hybrid-SORT [12], según se detalla en el repositorio, se ha modificado con ciertas adaptaciones para ajustarse a los objetivos del proyecto en cuestión. El fin principal es la integración del algoritmo de seguimiento Hybrid-SORT con los modelos de detección más avanzados. Por lo tanto, se identificó que algunos de los pasos o conjuntos de datos sugeridos en la documentación original no eran pertinentes o aplicables al contexto específico de este proyecto, llevando a la modificación de la metodología para cumplir con las necesidades de integración y optimización del equipo.

1. Instalar o clonear el repositorio de Hybrid-SORT [27]:

```
    git clone https://github.com/ymzis69/HybridSORT.git
    cd HybridSORT
    pip3 install -r requirements.txt
    python3 setup.py develop
```

2. Instalar pycocotools:

```
    pip3 install cython
    pip3 install 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=
        PythonAPI'
```

3. Instalar otros paquetes necesarios:

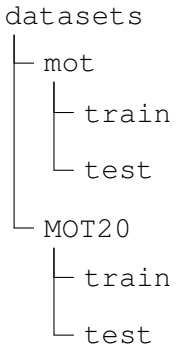
```
    pip3 install cython_bbox pandas xmltodict
```

4. Es un paso opcionar, pero al instalarlo hay que cambiar el "faiss-gpu"por "faiss-cpu", que se puede hacer referencia con el repositorio de fast-reid:

```
    pip install -r fast_reid/docs/requirements.txt
```

Con los pasos anteriores ya se pueden terminar la instalación del Hybrid-SORT, es recomendable utilizando la verisón superiores de 3.8.0 en python, ahora haciendo la preparación de los datos:

1. El primer paso consiste en la instalación de los conjuntos de datos necesarios. Aunque el repositorio proporciona diversas opciones, se optará por seleccionar específicamente los datasets MOT17 y MOT20 para simplificar el proceso. A continuación, se presenta la estructura de directorios requerida para organizar de manera adecuada estos conjuntos de datos dentro de la carpeta correspondiente, lo cual garantiza su correcta implementación y accesibilidad en las fases subsiguientes del proyecto:



2. Tras ubicar los conjuntos de datos en las carpetas designadas, el siguiente paso es la preparación de estos mediante un script específico que se encarga de convertir los datos al formato COCO. Esta conversión es esencial para estandarizar los datasets, facilitando su integración y uso posterior en diversas aplicaciones y herramientas que soportan este formato ampliamente reconocido en el ámbito de la visión por computadora:

```
python3 tools/convert_mot_to_coco.py
python3 tools/convert_mot20_to_coco.py
```

3. Y por ultimo paso construir conjuntos de entrenamiento mixtos para MOT17 y MOT20:

```
python3 tools/mix_data_test_mot17.py
python3 tools/mix_data_test_mot20.py
```

Una vez completada la preparación de los datos, y dado que no se procederá al entrenamiento de los modelos de detección desde cero, el siguiente paso es la incorporación de modelos preentrenados. Estos modelos pueden descargarse directamente del repositorio proporcionado. Aunque el enfoque principal no está en entrenar nuevos modelos de detección, la utilización de estos modelos preentrenados permitirá evaluar y ajustar el sistema de seguimiento, asegurando su correcta integración y funcionamiento óptimo con los datos preparados:

- Entrenar modelo de ablación:

```
python3 tools/train.py -f exps/example/mot/yolox_x_ablation.py -d 8 -b 48 --fp16
     -o -c pretrained/yolox_x.pth
```

- Entrenar modelo de test de MOT17:

```
python3 tools/train.py -f exps/example/mot/yolox_x_mix_det.py -d 8 -b 48 --fp16
    -o -c pretrained/yolox_x.pth
```

- Entrenar modelo de test de MOT20:

```
python3 tools/train.py -f exps/example/mot/yolox_x_mix_mot20_ch.py -d 8 -b 48 --
    fp16 -o -c pretrained/yolox_x.pth
```

7.4. Despliegue de Strong-SORT

Para asegurar una implementación correcta del algoritmo StrongSORT, es imprescindible que los usuarios sigan detenidamente las instrucciones disponibles en el repositorio de GitHub correspondiente al proyecto. Es esencial emplear Python en versiones 3.8.0 o superiores para garantizar un funcionamiento óptimo y una evaluación precisa del sistema. Se presenta a continuación una guía meticulosamente detallada que los usuarios deben seguir para implementar de manera efectiva el algoritmo. Esta guía abarca desde la configuración inicial del entorno de desarrollo hasta la ejecución y posterior análisis del sistema, facilitando así una implementación completa y exitosa de StrongSORT.

1. El primer paso consiste en clonar el repositorio de Git [28] en el entorno local:

```
git clone https://github.com/dyhBUPT/StrongSORT.git
```

2. Posteriormente, se deben descargar los conjuntos de datos necesarios, como MOT17 y MOT20, y colocarlos en la estructura de directorios adecuada. Dado que en el repositorio no existe la carpeta `path_to_dataset`, es necesario crearla manualmente.

```
path_to_dataspace
├── MOT17
│   ├── train
│   └── test
└── MOT20
    ├── train
    └── test
```

3. El siguiente paso implica descargar archivos adicionales necesarios desde un repositorio en Google Drive con la siguiente estructura:

```
path_to_dataspace
├── AFLink_epoch20.pth # checkpoints for AFLink model
├── MOT17_ECC_test.json # CMC model
├── MOT17_ECC_val.json # CMC model
├── MOT17_test_YOLOX+BoT # detections + features
├── ...
├── MOT20_ECC_test.json # CMC model
├── MOT20_test_YOLOX+BoT # detections + features
└── MOT20_test_YOLOX+simpleCNN # detections + features
```

4. Tras ubicar todos los conjuntos de datos y archivos necesarios en sus respectivas carpetas, es fundamental configurar adecuadamente las rutas en el archivo `opts.py`, incluyendo: `root_dataset`, `path_AFLink`, `dir_save`, `dir_dets`, `path_ECC`.

   Para minimizar la posibilidad de confusiones y errores durante la configuración del sistema, es altamente recomendable revisar el archivo "AuxiliaryTutorial.md", ubicado en la carpeta `others`. Este documento contiene una serie de instrucciones detalladas y paso a paso, diseñadas para guiar al usuario en el proceso de configuración y asegurar que se realice de manera correcta y eficiente. El seguimiento cuidadoso de estas directrices facilitará una experiencia de implementación más fluida y exitosa.

Una vez que se hayan completado los pasos previos, resulta imprescindible proceder con la descarga de ciertos paquetes adicionales. Estos son esenciales para prevenir potenciales inconvenientes futuros. Aunque es posible que estos paquetes ya se hayan instalado en implementaciones anteriores, es crucial asegurarse de que las versiones correctas estén presentes para el correcto funcionamiento de StrongSORT. Los paquetes necesarios incluyen:

- pytorch
- opencv
- scipy
- sklearn

Para instalar estos paquetes, los usuarios deben ejecutar los siguientes comandos en la terminal:

```
conda create -n strongsort python=3.8 -y
conda activate strongsort
pip3 install torch torchvision torchaudio
pip install opencv-python
pip install scipy
pip install scikit-learn==0.19.2
```

Estos comandos crean un entorno conda nuevo y específico para StrongSORT,

activan dicho entorno, e instalan las dependencias requeridas. Este procedimiento asegura que el entorno de trabajo contenga todas las herramientas necesarias para la operación exitosa del algoritmo.

Una vez completados los pasos preliminares, se procede a la fase de seguimiento utilizando diferentes comandos adaptados a varios modelos. A continuación, se describen los comandos específicos para cada variante de seguimiento:

- Para iniciar el seguimiento con DeepSORT en el conjunto de validación MOT17, se debe ejecutar el siguiente comando:

```
python strong_sort.py MOT17 val
```

- Para ejecutar el seguimiento con StrongSORT en el conjunto de validación MOT17, se utiliza el comando:

```
python strong_sort.py MOT17 val --BoT --ECC --NSA --EMA --MC --woC
```

- Para aplicar una variante mejorada, StrongSORT++, en el mismo conjunto de validación MOT17, se emplea:

```
python strong_sort.py MOT17 val --BoT --ECC --NSA --EMA --MC --woC --AFLink --
    GSI
```

- La evaluación de StrongSORT++ en el conjunto de prueba MOT17 se realiza mediante:

```
python strong_sort.py MOT17 test --BoT --ECC --NSA --EMA --MC --woC --AFLink --
    GSI
```

- Finalmente, para llevar a cabo el seguimiento con StrongSORT++ en el conjunto de prueba MOT20, se utiliza:

```
python strong_sort.py MOT20 test --BoT --ECC --NSA --EMA --MC --woC --AFLink --
    GSI
```

Estos comandos permiten a los usuarios ejecutar diversas variantes de seguimiento, adaptándose a distintos conjuntos de datos y algoritmos, completando así la implementación de StrongSORT en el entorno local.

7.5. Configuración e Inicialización de Trackers en un Modelo de Predicción YOLO para Seguimiento de Objetos

```
def on_predict_start(predictor):
    predictor.trackers = []
    predictor.tracker_outputs = [None] * predictor.dataset.bs
    predictor.args.tracking_config = \
    ROOT / 'boxmot' / opt.tracking_method / 'configs' / (opt.tracking_method + '.yaml')
    for i in range(predictor.dataset.bs):
        tracker = create_tracker(
            predictor.args.tracking_method,
            predictor.args.tracking_config,
```

```
10              predictor.args.reid_model,
11              predictor.device,
12              predictor.args.half
13          )
14          predictor.trackers.append(tracker)
```

A continuación se proporciona la explicación del código anterior:

- **Inicialización de Trackers**:La función on_predict_start recibe un objeto predictor. Se inicializa predictor.trackers como una lista vacía, que almacenará los rastreadores individuales. predictor.tracker_outputs se inicializa como una lista de None del tamaño del lote de datos (predictor.dataset.bs), preparándose para almacenar las salidas del rastreador para cada elemento del lote.
- **Configuración del Rastreador**:'predictor.args.tracking_config' se establece como la ruta al archivo de configuración YAML que contiene los parámetros para el rastreador. Esta configuración es específica para el método de seguimiento seleccionado ('opt.tracking_method') y se encuentra en la estructura de carpetas descrita.
- **Creación del Rastreador**:Dentro del bucle, se crea un rastreador para cada elemento del lote utilizando la función 'create_tracker'. Esta función configura cada rastreador con el método y la configuración especificados, el modelo de re-identificación ('predictor.args.reid_model'), el dispositivo en uso ('predictor.device') y si se utiliza precisión mixta ('predictor.args.half'). Cada rastreador creado se añade a la lista 'predictor.trackers'.

Dentro de la función principal, se inicia el proceso creando una instancia del modelo YOLO, especificada por el argumento `yolo_model`. A continuación, se obtienen las configuraciones destinadas al predictor vinculado a dicho modelo. Se procede a fusionar los argumentos predeterminados del predictor con los personalizados, otorgando prioridad a los últimos, que incluyen detalles como la fuente de datos y el tamaño de las imágenes.

Una vez configurado el modelo y determinada la fuente de datos, se verifica la compatibilidad del tamaño de las imágenes con el modelo. Además, se establece un directorio para almacenar los resultados, creándolo si no existe previamente.

Antes de comenzar con la fase de predicción, se realiza un proceso de "calentamiento" del modelo utilizando un tamaño de imagen específico, lo cual es crucial para asegurar la preparación y respuesta rápida del modelo al procesamiento de imágenes.

En este contexto se introduce y ejecuta la función `on_predict_start`, responsable de inicializar los rastreadores antes del inicio del proceso de predicción. Para cada imagen del conjunto de datos, se instaura y configura un rastreador, basado en la información proporcionada por su respectivo archivo de configuración YAML, detallando los parámetros y especificaciones del método de seguimiento empleado.

Justo antes de proceder con las predicciones mediante el modelo YOLO, se activan los callbacks asociados al inicio de la predicción, garantizando que todos los preparativos estén listos y que los rastreadores estén preparados para funcionar en conjunto con el modelo durante las etapas de detección y seguimiento.

Es crucial destacar que la salida del detector debe estar en el formato $N \times$

$(x, y, x, y, \text{conf}, \text{cls})$. Para obtener más detalles, se recomienda revisar el ejemplo proporcionado en el repositorio original [1].

7.6. Detalles de la Estructura de Datos COCO y Formato YOLO

En las secciones anteriores, especialmente en la de implementación, ya se realizó la adaptación de los conjuntos de datos al formato COCO. Esta información se organizó en archivos JSON, estructurados como diccionarios con cuatro componentes principales, lo que permitió una organización y acceso claros y sistemáticos a los datos:

- **images**: El componenete "images" tienen una lista de diccionarios. Cada palabra representa información sobre una única imagen del conjunto de datos COCO con pares clave-valor como:
  - "**file_name**": Una cadena de texto que especifica el nombre del archivo correspondiente a la imagen.
  - "**id**": Un identificador único asignado a cada imagen, representado por un número entero.
  - "**frame_id**": Un identificador único para cada cuadro de la imagen, similar al id, también representado por un número entero.
  - "**prev_image_id**": El identificador único de la imagen anterior en la secuencia, facilitando la navegación entre imágenes relacionadas.
  - "**next_image_id**": El identificador único de la imagen siguiente, permitiendo una secuenciación ordenada en el análisis de frames sucesivos.
  - "**video_id**": Un identificador que enlaza la imagen a un video específico, en caso de que la imagen forme parte de una secuencia de video.
  - "**height**": Un número entero que indica la altura de la imagen, proporcionando una dimensión crucial para el procesamiento de la misma.
  - "**width**": Un número entero que señala la anchura de la imagen, otro parámetro esencial para su análisis y procesamiento.
- **annotations**: El componenete "annotations" en el archivo JSON contiene una lista detallada de diccionarios, donde cada diccionario representa la anotación de un objeto individual dentro de una imagen del conjunto de datos COCO. Cada anotación proporciona información vital para la identificación y seguimiento de objetos en las imágenes. A continuación se detallan los elementos que componen cada anotación:
  - "**id**": Un identificador único asignado a cada anotación dentro del conjunto de datos, permitiendo la distinción y referencia específica a cada objeto anotado.
  - "**category_id**": Un número que designa la categoría a la que pertenece el objeto anotado, facilitando la clasificación y el reconocimiento de tipos de objetos dentro del conjunto de datos.

- "**image_id**": Un identificador que vincula la anotación con la imagen específica en la que el objeto está presente, asegurando la correspondencia precisa entre la anotación y su contexto visual.
- "**track_id**": Un identificador único asignado a cada objeto o trayectoria a lo largo de una secuencia de video, esencial para tareas de seguimiento que requieren la identificación consistente de objetos a través de múltiples frames.
- "**bbox**": Una lista que detalla las coordenadas del cuadro delimitador que encierra al objeto en la imagen, especificadas en el formato [x, y, width, height]. Estas coordenadas son cruciales para localizar el objeto dentro de la imagen.
- "**conf**": Un valor decimal que indica el nivel de confianza o certeza de la anotación, sirviendo como un indicador de la fiabilidad de la identificación del objeto.
- "**iscrowd**": Un indicador que señala si la anotación representa a un grupo de objetos. Un valor de 0 indica una anotación individual, mientras que un valor de 1 indica que la anotación comprende un grupo o conjunto de objetos.
- "**area**": El área del cuadro delimitador, calculada multiplicando el ancho por el alto. Este dato proporciona una medida de la escala o tamaño del objeto dentro de la imagen.

- **videos**: El componente "videos" es una lista de diccionarios donde cada diccionario representa un video individual o una secuencia dentro del conjunto de datos. Los componentes esenciales para cada entrada de vídeo suelen ser los siguientes:
  - "**id**": Un identificador único asignado a cada video o secuencia. Este ID facilita la referencia específica y el acceso a los datos asociados con cada secuencia de video dentro del conjunto de datos.
  - "**file_name**": El nombre del archivo o la designación de la secuencia, que generalmente se utiliza para localizar y acceder al video o conjunto de imágenes correspondiente dentro del conjunto de datos.
- **categories**: El componenete "categories" se deine para organizar y clasificar los tipos de objetos presentes en el conjunto de datos, es una lista de daiccionarios, donde cada diccionario define una categoría específica:
  - "**id**": Un identificador único para cada categoría. Este ID es crucial para vincular anotaciones específicas a su categoría correspondiente, permitiendo una clasificación precisa de los objetos detectados.
  - "**name**": El nombre de la categoría, que describe el tipo de objeto que la categoría representa. Este nombre es esencial para la interpretación humana y la referencia en análisis posteriores.

Para entrenar los modelos, es esencial convertir los datos al formato YOLO, un tipo de estructura de datos diseñada para la configuración de conjuntos de datos. Este formato especifica la ubicación base del conjunto de datos e incluye las rutas

relativas a los directorios que contienen las imágenes de entrenamiento, validación y prueba, o bien a archivos *.txt que listan las rutas de las imágenes. Además, incluye un diccionario que asigna nombres a las diferentes clases:

```
path: /tmp/dataset # dataset root dir
train: /tmp/dataset/images/train # train images
val: /tmp/dataset/images/val # val images
test: /tmp/dataset/images/test # test images (optional)
# Classes (80 COCO classes)
names:
    0: Person
nc: 1
```

Dado el archivo de configuración establecido, es necesario reorganizar el conjunto de datos de acuerdo con la estructura especificada a continuación:

```
dataset_YOLO
|------------images
|        |------------images_filename1.jpg
|        |------------images_filenamex.jpg
|------------labels
|        |------------images_filename1.txt
|        |------------images_filenamex.txt
|------------dataset.yaml
```

Este formato se organiza en dos carpetas principales: una destinada a las imágenes y la otra a las anotaciones. Cada imagen corresponde a un archivo de etiquetas, el cual está estructurado de tal manera que cada línea representa un objeto, utilizando el formato:

object-class	x_center	y_center	width	height
0	0.317187	0.567382	0.012537	0.052734
0	0.385937	0.615234	0.012537	0.039062
0	0.573437	0.428710	0.012537	0.044921
0	0.584375	0.477539	0.012537	0.041015
0	0.609375	0.470703	0.012537	0.042968
0	0.617187	0.407226	0.009375	0.041015
0	0.636718	0.450195	0.017187	0.056640
0	0.708593	0.487304	0.014062	0.048828
0	0.690625	0.526367	0.012537	0.041015

Tabla XVIII: Ejemplo de anotaciones de YOLO.

Aquí se detalla que:

- "**object-class**": Corresponde a un número entero que asigna cada objeto a su categoría (por ejemplo, "0" puede representar a "persona", "1" a "automóvil", etc., según la clasificación establecida en el conjunto de datos).
- "**x_center**" y "**y_center**": Son las coordenadas normalizadas del centro del cuadro delimitador del objeto, es decir, están ajustadas entre 0 y 1. Este ajuste se realiza dividiendo las coordenadas en píxeles por el ancho y el alto total de la

imagen, respectivamente.

- "**width**" y "**height**": Indican las dimensiones normalizadas del cuadro delimitador, también entre 0 y 1, obtenidas dividiendo el ancho y el alto del cuadro en píxeles por el ancho y alto total de la imagen, respectivamente.

7.7. Despliegue de YOLO-Tracking

En esta sección se explicará cómo desplegar el repositorio de Boxmot y los comandos utilizados para la evaluación de las métricas de tracking.

1. El primer paso, al igual que con otros despliegues de repositorios en GitHub, es clonar el repositorio. Para ello, utilizamos el siguiente comando:

```
git clone https://github.com/mikel-brostrom/yolo_tracking.git
```

2. El siguiente paso es instalar "Poetry", una herramienta para la gestión de dependencias y entornos en Python. Utilizamos la siguiente línea de comando:

```
pip install poetry
```

3. Una vez instalado Poetry, es necesario instalar las dependencias específicas del proyecto con Poetry:

```
poetry install --with yolo
```

4. Finalmente, es necesario activar el entorno de Poetry cada vez que se realicen las evaluaciones de tracking. Para entrar al entorno virtual creado por Poetry, utilizamos la siguiente línea de comando:

```
poetry shell
```

Una vez completados estos pasos, el entorno estará listo para ejecutar los scripts de evaluación de métricas de tracking en el repositorio de Boxmot.

Ahora se va poner un ejemplo de tracking con un video de Youtube, en este caso se utiliza el comando:

```
python tracking/track.py --source 'https://www.youtube.com/watch?v=ATK7gAaZTOM'
```

En caso de que no se quiera utilizar un video de YouTube, existen varias otras opciones para las fuentes de entrada. La tabla anterior muestra los diferentes tipos de entradas soportadas y sus descripciones. Para utilizar una fuente diferente, simplemente cambie el valor que sigue a la opción `–source`.

Para evaluar una combinación de un detector, un método de tracking y un modelo de ReID en un dataset estándar de MOT o en uno personalizado, se deben seguir los pasos a continuación:

Tipo de Entrada	Descripción
0	Webcam
`img.jpg`	Imagen
`vid.mp4`	Video
`path/`	Directorio
`path/*.jpg`	Glob (conjunto de archivos)
`'https://youtu.be/Zgi9g1ksQHc'`	YouTube
`'rtsp://example.com/media.mp4'`	RTSP, RTMP, HTTP stream

Tabla XIX: Tipos de entradas soportadas

Figura 7.2: Video sin tracking

Figura 7.3: Video con tracking

1. **Guardar detecciones y embeddings**: El siguiente comando guarda las detecciones y embeddings en el directorio **./runs/dets_n_embs**, separado para cada modelo YOLO y de ReID seleccionado.

```
python tracking/generate_dets_n_embs.py --source ./assets/MOT17-mini/train --
    yolo-model yolov8n.pt yolov8s.pt --reid-model weights/osnet_x0_25_msmt17.pt
```

2. **Generar resultados en formato MOT Challenge**: Se deben utilizar las detecciones y embeddings pre-generados para generar resultados en el formato del MOT Challenge basado en un método de tracking específico.

```
python tracking/generate_mot_results.py --dets yolov8n --embs osnet_x0_25_msmt17
     --tracking-method botsort
```

3. **Generar métricas MOT utilizando TrackEval**: Este comando utiliza TrackEval para generar métricas MOT para los resultados de tracking y los guarda en el directorio **./runs/mot/<dets+embs+tracking-method>**.

```
python tracking/val.py --benchmark MOT17-mini --dets yolov8n --embs
    osnet_x0_25_msmt17 --tracking-method botsort
```

Para la optimización de hiperparámetros del tracker, se utiliza un algoritmo genético multiobjetivo rápido y elitista. Por defecto, los objetivos son: HOTA, MOTA e IDF1. Se deben seguir los pasos a continuación para ejecutar esta evolución:

1. **Guardar detecciones y embeddings**: El siguiente comando guarda las detecciones y embeddings en el directorio **./runs/dets_n_embs**, separado para cada modelo YOLO y de ReID seleccionado.

```
python tracking/generate_dets_n_embs.py --source ./assets/MOT17-mini/train --
    yolo-model yolov8n.pt yolov8s.pt --reid-model weights/osnet_x0_25_msmt17.pt
```

2. **Evolucionar parámetros para un método de tracking específico**: Se deben utilizar las detecciones y embeddings generados en el paso anterior para evolucionar los parámetros del método de tracking especificado.

```
python tracking/evolve.py --benchmark MOT17-mini --dets yolov8n --embs
    osnet_x0_25_msmt17 --n-trials 9 --tracking-method botsort
```

Estas instrucciones permiten evaluar y optimizar diferentes combinaciones de detectores, métodos de tracking y modelos de ReID, proporcionando una manera sistemática de mejorar el rendimiento de los sistemas de tracking en datasets estándar como MOT.

7.8. Conversión del Dataset a Formato COCO(YOLOv8)

En esta sección se describen los pasos para transformar el dataset a formato COCO, con el fin de crear un modelo personalizado y utilizarlo en la comparación. Este proceso involucra la importación de paquetes necesarios, la definición de rutas y la creación de las carpetas necesarias para almacenar las imágenes y las etiquetas en el nuevo formato.

- El primer paso es importar los paquetes necesarios que se utilizarán en este proceso:

```
import os
import numpy as np
import json
import cv2
import shutil
import matplotlib.pyplot as plt
from PIL import Image
import random
import matplotlib.patches as patches
```

- El siguiente paso es definir las rutas de los datos y las carpetas donde se almacenarán las imágenes y etiquetas transformadas:

```
SPLITS = ['train', 'test']
DATA_PATH = 'D:/UPM/TFM/dataset/MOT17'
OUT_IMG_PATH = os.path.join('D:/UPM/TFM/dataset/MOT17', 'images')
OUT_LABEL_PATH = os.path.join('D:/UPM/TFM/dataset/MOT17', 'labels')
```

- Finalmente, se utiliza el siguiente código para crear las carpetas necesarias y transformar los datos al formato COCO. Este código copia las imágenes y procesa las anotaciones para adaptarlas al formato requerido:

```
for split in SPLITS:
    data_path = os.path.join(DATA_PATH, split)
    seqs = os.listdir(data_path)
    OUT_IMG_SAVE_PATH = os.path.join(OUT_IMG_PATH, split)
    OUT_LABEL_SAVE_PATH = os.path.join(OUT_LABEL_PATH, split)
```

```
6
7             if not os.path.exists(OUT_IMG_SAVE_PATH):
8                 os.makedirs(OUT_IMG_SAVE_PATH)
9             if not os.path.exists(OUT_LABEL_SAVE_PATH):
10                os.makedirs(OUT_LABEL_SAVE_PATH)
11
12            for seq in sorted(seqs):
13                seq_path = os.path.join(data_path, seq)
14                img_path = os.path.join(seq_path, 'img1')
15                det_path = os.path.join(seq_path, 'det/det.txt')
16                images = sorted(os.listdir(img_path))
17                annotations = np.loadtxt(det_path, delimiter=",")
18
19                for frame in images:
20                    frame_id = int(frame.split('.')[0])
21                    new_name = seq + '_' + frame
22                    IMG_new_path = os.path.join(OUT_IMG_SAVE_PATH, new_name)
23                    IMG_path = os.path.join(img_path, frame)
24                    shutil.copy(IMG_path, IMG_new_path)
25
26                    # Obtener la altura y la anchura de la imagen
27                    with Image.open(IMG_path) as img:
28                        img_width, img_height = img.size
29
30                    processed_annotations = []
31                    frame_annotations = annotations[annotations[:, 0] == frame_id]
32                    for ann in frame_annotations:
33                        class_id = 0  # Asumiendo una sola clase, cambiar si es
                              necesario
34                        x, y, w, h = ann[2:6]
35                        # Convertir el formato del bbox de (x, y, w, h) a formato YOLO (
                              x_center, y_center, w, h) y normalizar
36                        x_center = (x + w / 2) / img_width
37                        y_center = (y + h / 2) / img_height
38                        w = w / img_width
39                        h = h / img_height
40                        processed_annotations.append([class_id, x_center, y_center, w, h
                              ])
41
42                 ruta_archivo = os.path.join(OUT_LABEL_SAVE_PATH, new_name.replace(".jpg
                       ", ".txt"))
43                   with open(ruta_archivo, 'w') as f:
44                        for ann in processed_annotations:
45                          f.write(' '.join(map(str, ann)) + '\n')
46        print('finish')
```

Este código realiza las siguientes acciones:

- Recorre las divisiones del dataset ('train' y 'test').
- Crea las carpetas necesarias para almacenar las imágenes y las etiquetas en las nuevas rutas.
- Copia las imágenes a la nueva ubicación y transforma las anotaciones del formato original al formato YOLO, normalizando las coordenadas de los bounding boxes.
- Guarda las anotaciones transformadas en archivos de texto correspondientes a cada imagen.

Con estos pasos, el dataset se transforma correctamente al formato COCO, listo para ser utilizado en la creación de un modelo personalizado y su posterior comparación.

7.9. Conversión del Dataset a Formato COCO(YOLO-NAS)

En esta sección se describe el proceso para convertir un dataset en formato COCO al formato compatible con YOLO-NAS. A continuación se presentan los pasos necesarios para realizar esta conversión, incluyendo la carga de las anotaciones, el copiado de las imágenes y la creación de los archivos de etiquetas en el formato requerido.

- Importación de bibliotecas necesarias:

```
import json
import cv2
import os
import matplotlib.pyplot as plt
import shutil
import numpy as np
```

- Definición de las rutas de entrada y salida:

```
input_path = "D:/UPM/TFM/dataset/MOT17/"
output_path = "D:/UPM/TFM/MOT17/"
annotations_file = 'D:/UPM/TFM/dataset/MOT17/annotations/train.json'
```

- Carga de las anotaciones desde el archivo JSON:

```
# Load annotations
with open(annotations_file, 'r') as f:
    data = json.load(f)
```

- Función para asegurar la existencia de un directorio:

```
def ensure_dir(directory):
if not os.path.exists(directory):
    os.makedirs(directory)
```

- Función para cargar imágenes desde una carpeta y copiarlas a la carpeta de salida:

```
def load_images_from_folder(folder):
for foldername in os.listdir(folder):
    image_path = os.path.join(folder, foldername, 'img1')
    if not os.path.isdir(image_path):
        continue
    count = 1
    for filename in os.listdir(image_path):
        source = os.path.join(image_path, filename)
        savename = f'{foldername}-img{count}.jpg'
        destination = os.path.join(output_path, 'images/train', savename)
        ensure_dir(os.path.dirname(destination))
        try:
            shutil.copy(source, destination)
            print(f"Archivo {source} copiado exitosamente a {destination}.")
        except Exception as e:
            print(f"Error copiando {source}: {e}")
        count += 1
```

- Función para guardar las etiquetas en el formato YOLO:

```
        def save_labels(data, output_path):
        for ann in data['annotations']:
            image_id = ann['image_id']
            bbox = ann['bbox']
            x, y, w, h = bbox
            x_centre = x + w / 2
            y_centre = y + h / 2

            img_info = next((img for img in data['images'] if img['id'] == image_id),
                None)
            if img_info:
                img_w = img_info['width']
                img_h = img_info['height']
                x_centre /= img_w
                y_centre /= img_h
                w /= img_w
                h /= img_h
                label_path = os.path.join(output_path, f"{img_info['file_name'].replace
                    ('.jpg', '')}.txt")
                ensure_dir(os.path.dirname(label_path))
                with open(label_path, 'a') as file_object:
                    file_object.write(f"0 {x_centre} {y_centre} {w} {h}\n")
```

- Cargar las imágenes y guardar las etiquetas:

```
        load_images_from_folder(os.path.join(input_path, 'train'))
        save_labels(data, os.path.join(output_path, 'labels/train'))
```

- Validación de las rutas y etiquetas:

```
        # Validating paths and labels
        DATA_PATH = 'D:/UPM/TFM/dataset/MOT17/test/'
        OUT_IMG_PATH = 'D:/UPM/TFM/MOT17/images/test/'
        OUT_LABEL_PATH = 'D:/UPM/TFM/MOT17/labels/test/'

        ensure_dir(OUT_IMG_PATH)
        ensure_dir(OUT_LABEL_PATH)
```

- Procesamiento de las secuencias de prueba y guardado de las imágenes y etiquetas correspondientes:

```
        seqs = os.listdir(DATA_PATH)
        for seq in sorted(seqs):
            seq_path = os.path.join(DATA_PATH, seq)
            img_path = os.path.join(seq_path, 'img1')
            det_path = os.path.join(seq_path, 'det/det.txt')

            if not os.path.isfile(det_path):
                continue

            images = sorted(os.listdir(img_path))
            annotations = np.loadtxt(det_path, delimiter=",")
            images_info = os.path.join(seq_path, 'seqinfo.ini')

            with open(images_info) as fichero:
                for line in fichero:
                    line = line.strip()
                    if 'imWidth' in line:
                        img_w = int(line.split('=')[1])
                    if 'imHeight' in line:
                        img_h = int(line.split('=')[1])

            count = 1
```

```
            for i, frame in enumerate(images):
                frame_annotations = annotations[annotations[:, 0] == i + 1]
                source = os.path.join(img_path, frame)
                savename = f'{seq}-img{count}.jpg'
                destination = os.path.join(OUT_IMG_PATH, savename)

                try:
                    shutil.copy(source, destination)
                    print(f"Archivo {source} copiado exitosamente a {destination}.")
                except Exception as e:
                    print(f"Error copiando {source}: {e}")
                    continue

                label_file_path = os.path.join(OUT_LABEL_PATH, f"{savename.replace('.jpg
                    ', '')}.txt")
                ensure_dir(os.path.dirname(label_file_path))
                with open(label_file_path, "a") as file_object:
                    for annotation in frame_annotations:
                        current_category = 0
                        x, y, w, h = map(float, annotation[2:6])
                        x_centre = (x + w / 2) / img_w
                        y_centre = (y + h / 2) / img_h
                        w /= img_w
                        h /= img_h
                        file_object.write(f"{current_category} {x_centre:.6f} {y_centre
                            :.6f} {w:.6f} {h:.6f}\n")

                count += 1

            print('Proceso finalizado para la secuencia:', seq)
```

Con estos pasos, se logra transformar el dataset original en formato COCO a un formato adecuado para YOLO-NAS, permitiendo así la utilización de este modelo para la detección de objetos en las imágenes del dataset.

7.10. Implementación y Entrenamiento del Modelo Personalizado

En esta sección, siguiendo la sección de transformación [7.8 y 7.9], una vez obtenidas las carpetas transformadas, ya se puede crear el modelo personalizado siguiendo los pasos siguientes:

- Instalar el paquete de ultralytics versión 8.0.124, y comprobar si es la última versión. En caso de no serlo, actualizarlo a la última versión:

```
!pip install ultralytics==8.0.124
!pip install --upgrade ultralytics
```

- Comprobar la versión de CUDA del sistema que está utilizando, recomendando utilizar la versión 12.1, ya que las versiones recientes no son compatibles con los paquetes necesarios como 'torch', etc.:

```
!nvcc --version
```

- Comprobar el entorno para determinar si está utilizando GPU o CPU. Se recomienda utilizar el GPU para un procesamiento más rápido:

```
import torch

if torch.cuda.is_available():
    print("CUDA is available. PyTorch is using the GPU.")
    print("GPU device name:", torch.cuda.get_device_name(0))
else:
    print("CUDA is not available. PyTorch is using the CPU.")
```

- Verificar y corregir las etiquetas de los datos para asegurar que están en el formato correcto:

```
import os

def check_and_fix_labels(label_dir):
    for label_file in os.listdir(label_dir):
        if label_file.endswith('.txt'):
            label_path = os.path.join(label_dir, label_file)
            with open(label_path, 'r') as file:
                lines = file.readlines()

            fixed_lines = []
            for line in lines:
                parts = line.strip().split()
                if len(parts) != 5:
                    print(f"Skipping malformed line in {label_file}: {line}")
                    continue
                class_id, x, y, w, h = map(float, parts)
                if 0 <= x <= 1 and 0 <= y <= 1 and 0 <= w <= 1 and 0 <= h <= 1:
                    fixed_lines.append(line)
                else:
                    print(f"Skipping out of bounds coordinates in {label_file}:
                        {line}")

            with open(label_path, 'w') as file:
                file.writelines(fixed_lines)

# Usar la ruta de tu carpeta de etiquetas
check_and_fix_labels('D:/UPM/TFM/dataset/MOT17/labels/train')
check_and_fix_labels('D:/UPM/TFM/dataset/MOT17/labels/test')
```

- Entrenar el modelo personalizado utilizando los datos y etiquetas corregidos:

```
from ultralytics import YOLO

# Cargar el modelo YOLOv8 preentrenado
model = YOLO('yolov8n.pt')

# Iniciar el entrenamiento
model.train(data='D:/UPM/TFM/dataset/MOT17/dataset_1.yaml', epochs=3, imgsz=640,
     workers=8)
```

Siguiendo estos pasos, se logra crear y entrenar un modelo YOLOv8 personalizado utilizando el conjunto de datos transformado. Este proceso asegura que las etiquetas de los datos estén en el formato correcto y que el entorno de entrenamiento esté adecuadamente configurado para utilizar GPU, lo cual optimiza el tiempo de entrenamiento.