UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

EVALUACIÓN DE RENDIMIENTO DE YOLOV5 Y ALGORITMOS DE SEGUIMIENTO EN UNA JETSON NANO 2GB

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

TOMÁS MAURICIO VALLS JIMÉNEZ

PROFESOR GUÍA:
Iván Sipirán Mendoza

MIEMBROS DE LA COMISIÓN:
José Saavedra Rondo
Juan Barrios Nuñez

Este trabajo ha sido financiado por:
Sinantica Spa

SANTIAGO DE CHILE
2023

EVALUACIÓN DE RENDIMIENTO DE YOLOV5 Y ALGORITMOS DE SEGUIMIENTO EN UNA JETSON NANO 2GB

Los sistemas de seguimiento del estado del arte suelen tener como requisito un alto poder de procesamiento. Una respuesta común a esta necesidad ha sido apoyarse en el *cloud-computing*. Sin embargo, distintas características, como sus altos costos, incapacidad de llegar a lugares remotos y problemas de seguridad, hacen que no sea ideal en todos los contextos.

Este estudio, realizado en colaboración con Sinantica Spa, evalúa el rendimiento de YOLOv5 y los algoritmos de seguimiento Centroid, Sort y Deepsort en una Jetson Nano 2GB. Se comparan y contrastan las diferencias en términos de precisión y velocidad entre estas herramientas con el objetivo de establecer una línea de referencia para soluciones desarrolladas en *edge-devices* de bajo costo y baja capacidad como la Jetson Nano 2GB.

Los resultados más relevantes tras el desarrollo del trabajo se presentan en la sección de velocidad de inferencia de los *benchmarks*. Las cuales entregan tasas de FPS para dos modelos de *tracking*, involucrando 24 configuraciones distintas.

Del proyecto se logra extraer la viabilidad de una solución desplegada en el hardware escogido, junto con descartar el uso de modelos más complejos como Deepsort. Adicionalmente, se logra acotar las configuraciones que entregan resultados útiles, así dando una idea de qué parámetros estudiar en futuras iteraciones similares.

A seguir esforzándose,
porque esto esta comenzando.

Agradecimientos

Hay mucha gente a la que debo agradecer, partiendo por mi familia. Mis padres, Gonzalo y Elisa, quienes siempre han estado ahí para mi. Por un lado, siempre me han guiado en la dirección correcta, pero a la vez han dejado que recorra mi propio camino. Sumado a esto, me han dado el lujo de siempre tener un lugar donde me siento seguro.

También me gustaría dar las gracias a mis hermanos, Nicolas, Cristián y Montserrat. Al observarlos y convivir con ellos he aprendido distintas cosas relevantes para mi vida. Primero, que no es necesario dejar de lado ciertos aspectos de la vida en función de otros. También, que con creatividad y pasión, aprender puede brindar grandes emociones. Por último, a siempre intentar no posponer la alegría.

Aprovecho de agradecer a los amigos que me han acompañado a lo largo de distintas etapas. A mis amigos del colegio, quienes son una constante en mi vida y me entregan un lugar en donde nunca faltan las sonrisas. Por otro lado, a mis amigos de la universidad, los FIPES, los cuales al poco conocernos me hicieron sentir incluido y feliz.

Por otro lado, quiero agradecer a Bo Xiao de la Universidad politécnica de Hong Kong. En específico, quisiera mencionar que me brindo su asistencia pese a que el protocolo usual de su universidad es ayudar a gente con la cual existe relación previa. Destacando que la base de datos que compartió conmigo fue clave para el entrenamiento de redes neuronales usadas en este proyecto.

En adición a estos, me gustaría reconocer a la gente de Sinantica. Partiendo por Joaquín, quien al darme la oportunidad de desarrollar mi segunda práctica en su empresa me permitió conocer y aprender más de esta área. En adición a esto, brindarme la oportunidad de hacer este trabajo. Por otro lado, me gustaría agradecer a Jorge, el cual me asistió y guío en el proceso.

Finalmente, me gustaría mencionar al profesor Ivan, quien invirtió tiempo en ayudarme a pesar de que el tema de trabajo no fue una iniciativa de él. Por otro lado, destacar que su experiencia y conocimientos en el tema de desarrollo fueron herramientas claves para el progreso del trabajo. Por último, destacar que su consejo al momento de tener que tomar decisiones clave fue imprescindible para poder terminar el proyecto.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación

El gran volumen de tráfico vehicular en ciudades de todo el mundo lo convierte en una situación de interés para diferentes entidades. El monitoreo del flujo de vehículos dentro y fuera de una ciudad, junto con la captura y detección de accidentes y congestión vehicular, pueden ser sumamente relevantes. Dichas acciones pueden ayudar a tomar decisiones potencialmente efectivas al momento de prevenir y actuar frente a eventos de este contexto.

Hay una variedad de soluciones en la academia e industria, las cuales usualmente consisten en combinar algoritmos de inteligencia artificial y visión computacional. Su precisión, rendimiento y escalabilidad, hacen que las herramientas de estas áreas sean la primera opción al momento de implementar sistemas de monitoreo automatizados. Sin embargo, para su despliegue es esencial un alto poder de procesamiento.

El alza en la complejidad de los algoritmos ha ido de la mano con el desarrollo en distintos frentes por parte de la industria. Por un lado, el desarrollo de procesadores de tipo CPU, GPU y TPU de mayor capacidad, han aumentado las posibilidades de las soluciones diseñadas. Luego, el desarrollo de *edge-devices* especializados en despliegue, ha abierto el abanico de probabilidades para proyectos del área de IOT[1]. Por otro lado, una de las alternativas de uso frecuente por parte de entes privadas son los servicios de *cloud-computing*, los cuales son brindados por grandes proveedores como Amazon, Google y Azure.

Múltiples contextos en donde la información es de contenido sensible, hacen que el *cloud-computing* no sea una opción viable. Asimismo, los altos costos del procesamiento remoto y la incapacidad de llegar a lugares aislados sin acceso a Internet, hacen que este paradigma de procesamiento no sea ideal en algunas situaciones.

El trabajo realizado tiene como objetivo implementar un sistema de análisis de tránsito vehicular que entregue resultados en tiempo real. Diseñando una solución que no utilice procesamiento remoto y que base su funcionamiento en *edge-computing*. Al implementar la totalidad de la solución en una Jetson Nano 2GB, uno de los *edge-devices* más económicamente asequibles y con menor memoria RAM de la compañía NVIDIA, se busca crear cimientos útiles para trabajos que utilicen *edge-devices* de igual o mayor capacidad.

[1] Internet of things.

1.2. Antecedentes

1.2.1. Empresa de trabajo

El proyecto se realiza en conjunto con la empresa Sinantica SPA. El interés de la empresa es determinar el posible alcance de una solución diseñada con las condiciones impuestas en el proyecto. Concretamente, se busca establecer los límites de una implementación que realice el alcance la totalidad del procesamiento en el equipo seleccionado, evitando el uso de procesamiento remoto.

De esta manera, se quiere reutilizar algoritmos y aprendizajes logrados a lo largo del trabajo en futuros proyectos que aborde la compañía. En adición a esto, se busca sacar beneficio de los componentes obtenidos a través de los objetivos secundarios del trabajo. Los que más destacan son la bases de imágenes y anotaciones que se utilizara en el entrenamiento de redes neuronales, junto con los pesos obtenidos en este.

Sinantica SPA es una *start up* basada en Valdivia que se centra en acelerar la transformación digital de distintas organizaciones. Especializándose en soluciones relacionadas con la implementación de algoritmos de *deep learning* y procesamiento de imágenes, con un énfasis en el despliegue de sistemas IOT.

Sus proyectos buscan automatizar, mediante inteligencia artificial y visión computacional, distintos procesos que requieran sus clientes. Pertenecientes a diversos rubros, sus proyectos se desenvuelven en la industria pesquera, sector agrícola y empresas de manufactura.

1.2.2. Lugar de despliegue

El lugar de despliegue es Costa Verde, un camino rural de alto tráfico situado en el sur de Chile. Este cuenta con un flujo compuesto por distintos tipos de vehículos, entre ellos: autos, camiones, motocicletas, bicicletas y buses, en adición al movimiento de peatones. Se planifica el funcionamiento del monitoreo a lo largo de todo el año, durante el día y noche. La imagen de captura se centra en una vía direccional y una calle secundaria que la intersecta de manera perpendicular.

Entre las locaciones en que Sinantica tiene proyectos en curso, Costa Verde se escoge debido a que su contexto es de interés para la empresa. Adicionalmente, se estima que el lugar tiene características que son de utilidad para futuros trabajos. Entre las propiedades que se consideran más relevantes destacan: los tipos de vehículos presentes, región geográfica y ángulo de captura que posee la cámara de vídeo.

Figura 1.1: Lugar de despliegue.

1.2.3. Arquitectura del sistema actual

Actualmente, el sistema instalado en el lugar se divide en dos secciones: local y remota. La parte local de la arquitectura tiene como tarea capturar en vídeo la situación del lugar y mandarla a un sitio lejano. Por otro lado, en la sección remota del sistema ocurre el procesamiento, interpretación, almacenamiento de datos y entrega final de resultados.

En específico, la parte remota se compone por una nube virtual privada o "VPC", la cual es proporcionada por Amazon. Entre los múltiples elementos y funcionalidades que provee, se destaca que sus principales componentes son herramientas de procesamiento, almacenamiento y manejo de datos de manera remota.

Es importante señalar que los objetivos del proyecto se limitan a lo relacionado con el procesamiento y traducción de lo que sucede en el lugar. Por lo que se considera al almacenamiento de datos, junto con la traducción y entrega de resultados, funciones que escapan del alcance del proyecto.

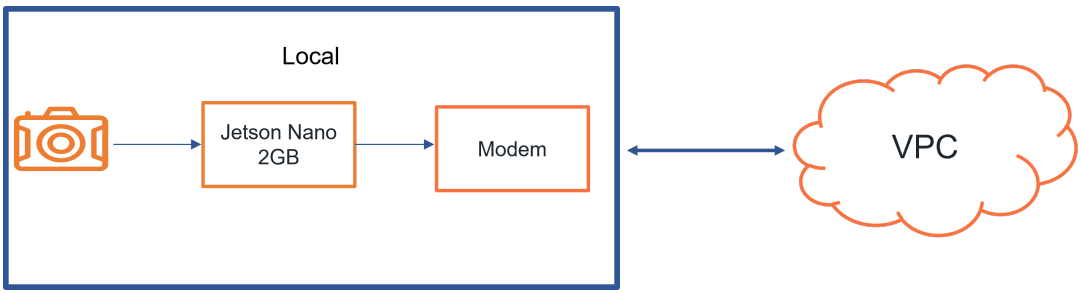

Figura 1.2: Arquitectura instalada en el lugar.

1.3. Equipo a utilizar

1.3.1. Jetson Nano 2 GB

NVIDIA es una compañía de alcance global que se especializa en el desarrollo de circuitos integrados. Estos forman parte de distintas aplicaciones, siendo las más destacadas el *edge-computing* y las unidades de procesamiento gráfico. El año 2014 comenzó el desarrollo de la

línea de productos Jetson, focalizada en placas de computación integradas de bajo consumo energético, cuyo diseño permite la aceleración de aplicaciones de *machine learning*.

En particular, Jetson Nano es una de las alternativas de menor consumo y más económicas de esta línea de productos. Al igual que otros hardware de esta línea, el sistema tiene incorporada una unidad de procesamiento gráfico (GPU). Este circuito integrado permite procesamientos en paralelo, junto con la disminución de tiempo empleado en el despliegue de algoritmos de inteligencia artificial. Adicionalmente, el equipo cuenta con múltiples periféricos útiles para despliegues que involucren visión computacional y redes neuronales.

Más recientemente, se desarrolla una variación de este hardware, llamada "Jetson Nano 2GB". Bajando de cuatro a dos gigas de memoria RAM, se comienza la producción de esta alternativa aún más económica y de menor consumo. Dados sus bajos requerimientos energéticos y precio, este equipo es una opción a considerar para el despliegue de proyectos de inteligencia artificial.

1.3.2. Cámara PTZ

De uso frecuente en vigilancia, las siglas "PTZ" de la cámara utilizada indican las posibilidades de movilidad que tiene el instrumento. Esta es capaz de moverse en múltiples ángulos, pudiendo rotar en un plano horizontal y vertical, además de poder acercar o alejar su visión (*pan-tilt-zoom*).

El modelo utilizado es **DS-2DE2A204IW-DE3** de la compañía Hikvision. El cual provee una máxima resolución de 1920×1080, rangos de movimiento de 355^{o} en giro y 90^{o} de inclinación. Junto con lo mencionado, la cámara admite conexión a Wi-Fi, es resistente al agua y proporciona un rango de tolerancia de radiación suficiente para su uso de noche y día.

Figura 1.3: Cámara PTZ.

1.4. Paradigmas de procesamiento

1.4.1. Cloud computing

El *cloud computing* es un arquetipo de procesamiento basado en la entrega de servicios informáticos de manera remota. En específico, se utiliza una red de servidores conectados a

Internet alejados del lugar de aplicación.

Entre las funciones que entrega este formato, se destaca el almacenamiento. Dicho servicio es imprescindible en aplicaciones que precisen guardar un alto volumen de datos. También se proveen otras prestaciones, tales como: procesamiento, bases de datos y distintos tipos de software.

Es importante señalar que el principal beneficio que entrega este paradigma radica en su alta capacidad, tanto de almacenamiento como procesamiento. En especial, cuando se contrasta la mencionada con la entregada por equipos instalados en el lugar de desarrollo. Por otro lado, se destaca la escalabilidad y elasticidad que brinda este modelo, siendo posible cambiar el alcance o funcionalidad sin tener que realizar grandes cambios de infraestructura.

Dado el alcance que entrega este paradigma, el abanico de ejemplos de su uso es extenso. La computación en la nube participa en actividades como permitir la sincronización de correos electrónicos, proveer almacenes de archivos y hospedar páginas web. Adicionalmente, el *cloud computing* puede ayudar en tareas que requieran de alto poder de procesamiento, como el entrenamiento de redes neuronales.

Vale la pena mencionar que en general los costes asociados a los servicios de *cloud computing* se generan dependiendo de las acciones puntuales que se realicen.

1.4.2. Edge computing

El *edge computing* es un modelo de computación distribuida en la que el procesamiento y/o almacenamiento de datos se acerca al "borde" del problema. En otras palabras, se disminuye la distancia con respecto al lugar del contexto, donde se consume o crea la información de interés.

Este paradigma entrega distintas ventajas, por ejemplo: la disminución en los tiempos de respuesta, ahorro en ancho de banda y aumento en seguridad. Debido a esto, una de las aplicaciones más frecuentes es en vehículos autónomos. Asimismo, un uso frecuente es en sistemas de monitoreo que se beneficien de alertas en tiempo real.

La reciente alza en la popularidad de este arquetipo se justifica con distintos acontecimientos. Uno de ellos es el aumento en el número de aparatos IOT, cuyo número los útlimos años supero a la población mundial. Por otro lado, el desarrollo de algoritmos de inteligencia artificial económicos ha permitido que sea posible su despliegue sin tener altos requerimientos. Sumado a esto, compañías han comenzado a desarrollar hardware dirigido a funcionar con este paradigma en mente, estos se conocen como *edge devices*.

1.5. Objetivos

1.5.1. Objetivo general

Evaluar el rendimiento de YOLO*v5* y los algoritmos de seguimiento Centroid, Sort y Deepsort en tiempo real utilizando una Jetson Nano 2GB. Se pretende comparar y contrastar

las diferencias en términos de precisión y rapidez entre estas herramientas y determinar cuál es la más adecuada para el seguimiento de objetos.

1.5.2. Objetivos específicos

Para lograr el objetivo general de este trabajo de título, se propone cumplir con los siguientes objetivos específicos:

1. Procurar una base de datos de tamaño y variedad suficiente para llevar a cabo un entrenamiento de las arquitecturas Nano y Small de la red de detección de YOLO*v5*. Específicamente, se requiere que la base de datos este constituida por imágenes y etiquetas de las clases: persona, bicicleta, automóvil, motocicleta, bus y camión.

2. Implementar los algoritmos de *tracking*: Centroid, Sort y Deepsort en Jetson Nano 2GB. Dicha implementación debe ser capaz de efectuar su despliegue en combinación con YOLO*v5s* y YOLO*v5n*. Adicionalmente, su procesamiento debe ser realizado mayoritariamente en la GPU del equipo.

3. Generar un *benchmark* de velocidad y calidad de resultados para un despliegue de las versiones *small* y *nano* de YOLO*v5* en Jetson Nano 2GB.

4. Generar un *benchmark* de velocidad y calidad de resultados para un despliegue de los algoritmos: Centroid, Sort y Deepsort en Jetson Nano 2GB.

5. Acotar los parámetros de configuración viables para el despliegue en una situación real de un sistema de seguimiento. Siendo las variables experimentales a estudiar: arquitectura de red de detección, plataforma de despliegue de redes neuronales, algoritmo de *tracking*, precisión de punto flotante y resolución de imagen de ingreso.

Capítulo 2

Marco Teórico

Con el fin de lograr un mayor entendimiento sobre el problema, su contexto y los métodos empleados para llegar a una solución, se presenta una recopilación de antecedentes y consideraciones teóricas en las que se sustenta el proyecto. Con los elementos explorados se busca aclarar la situación que rodea al trabajo, empresa con la que se desarrolla y hardware utilizado. Junto a esto se entregan definiciones básicas necesarias para comprender los enfoques utilizados en el desarrollo del trabajo realizado.

2.1. Detección de objetos

2.1.1. Antecedentes

La detección de objetos es un problema que tiene como objetivo identificar la posición de un objeto de interés dentro de una imagen. Inicialmente, esta labor se enfrentaba utilizando distintas técnicas basadas en la detección de características. La detección de patrones locales, ventanas deslizantes, filtros y gradientes se utilizaban para identificar objetos en distintas escenas. Entre los algoritmos de esta naturaleza que destacan, se encuentran los detectores de patrones binarios locales[1], SIFT[2], histogramas de gradientes orientados[3] y el modelo de partes deformables[4]. Este periodo es conocido como "periodo de detección de objetos tradicional".

Si bien estos algoritmos sirvieron como trabajo basal para futuros modelos, la necesidad de traducir características reales a patrones representables genera que su capacidad de generalización sea limitada. Frente a estos, en búsqueda de técnicas más flexibles y gracias a la disponibilidad de un mayor poder de procesamiento, la detección de objetos se comienza a apoyar más en las redes neuronales convolucionales (CNN). Se puede identificar este periodo de transición entre el 2010 y 2014, siendo uno de los mayores hitos la creación de la red RCNN[5]. Luego, la detección de objetos se comienza a basar en *deep learning*, dando inicio al segundo periodo conocido como "periodo de detección de objetos basado en *deep learning*".

Frente al éxito de este enfoque para el diseño de modelos de detección de objetos, se generaron múltiples redes convolucionales, habiendo distintos tipos de subcategorías dependiendo de su funcionamiento. Una primera clasificación se da según si el proceso de buscar y clasificar objetos se realiza en una o varias etapas, denominándose "*one-stage*" o "*two-stages*", respectivamente. En términos generales, hacerlo en una sola etapa entrega una ventaja en

velocidad, sin embargo, al dividir el proceso en dos se consigue una mayor precisión en los resultados.

Por otro lado, existe otra división que depende del método usado para inferir la posición de los objetos. El primer método, divide la imagen en varios segmentos intentando clasificar a distintas escalas el objeto que se encuentra sobre un punto de interés de la división. Luego, el segundo método consiste en analizar ciertas características de la imagen como bordes, iluminación y colores, con el fin de extraer un punto que describa al objeto. Una vez definidos los puntos suficientes, se infiere el área en el que se encuentra el objeto y se procede a clasificar. La división descrita se puede separar en dos clasificaciones, "*anchor-based*" y "*keypoint-based*", respectivamente.

Este proyecto se desarrolla con dos redes de detección y reconocimiento de objetos del estado del arte:

- **YOLO*v5***: Se utiliza esta arquitectura donde es necesario un entrenamiento de la red. Esto correspondería a la identificación inicial de clases interés para un sitio de tránsito vehicular.
- **Osnet**: Se asigna esta arquitectura cuando se requiere medir la similitud entre dos entidades. Para esto, la red extrae un vector de características descriptivas de los objetos que se desean comparar.

2.1.2. YOLO

La arquitectura YOLO, es presentada por primera vez el 2015, en el artículo escrito por Joseph Redmon llamado "*You Only Look Once: Unified, Real-Time Object Detection*"[6]. Su uso se masificó con gran velocidad por su alta exactitud y su capacidad de correr en tiempo real.

Basándose en la arquitectura de Googlenet[7], YOLO comienza por dividir la imagen en una grilla, donde cada celda se encarga de verificar si existe un objeto de interés en su interior. Con este fin, cada celda genera múltiples tamaños de potenciales *bounding boxes*[2], definiendo con que niveles de confianza se hace una predicción de la existencia de un objeto. Posterior a la publicación del artículo, se generan nuevas versiones de la red, habiendo siete versiones a la fecha.

2.1.3. Versiones

Cada iteración de la arquitectura introduce distintas mejoras sobre sus antecesores, sin embargo, se destaca YOLO*v3* por su nivel de impacto. Reemplazando *softmax* por clasificadores logísticos independientes e introduciendo extracción de características, la creación de YOLO*v3* fue disruptiva dado su rendimiento similar a otras redes *one-stage* y velocidad muy superior.

[2] Cuadro delimitador que rodea una zona en específico de una imagen, por lo general es representado por cuatro números.

Posteriormente, YOLO*v4* y YOLO*v5* introducen cambios que generan una mejoría en los *benchmarks*. En particular, el uso de *Mosaic Data Augmentation* en el entrenamiento y la implementación de *Cross Stage Partial Networks* como columna vertebral de las arquitecturas muestran notorias alzas en la precisión de los resultados y en la velocidad de su entrenamiento. Finalmente, se destaca que YOLO*v5* entrega resultados similares a su versión anterior, sin embargo, requiere de un tiempo de entrenamiento aun menor. En adición a esto, el desarrollo de esta arquitectura se implementa nativamente en Pytorch, a diferencia de las anteriores basadas en *Densenet*.

De la misma forma en que YOLO posee diferentes versiones, existen diferentes arquitecturas dentro de una misma versión. Concretamente, para YOLO*v5* existen cinco versiones que se diferencian en el tamaño de parámetros utilizados. Denominados modelos $P5$, estos son: YOLO*v5n* (nano), YOLO*v5s* (small), YOLO*v5m* (medium), YOLO*v5l* (large) y YOLO*v5x* (extra large). En adición a éstas, existen los modelos $P6$, siendo la única diferencia que las anteriores están diseñadas para ser entrenadas con una imagen de resolución 640×640, a diferencia de la P5 cuyo diseño está pensado para resolución 1280×1280. De manera general, el aumento en tamaño provoca una mejora en el nivel de precisión de la red y una disminución en la velocidad de inferencia, por lo que la elección de la arquitectura recae en la importancia de la rapidez.

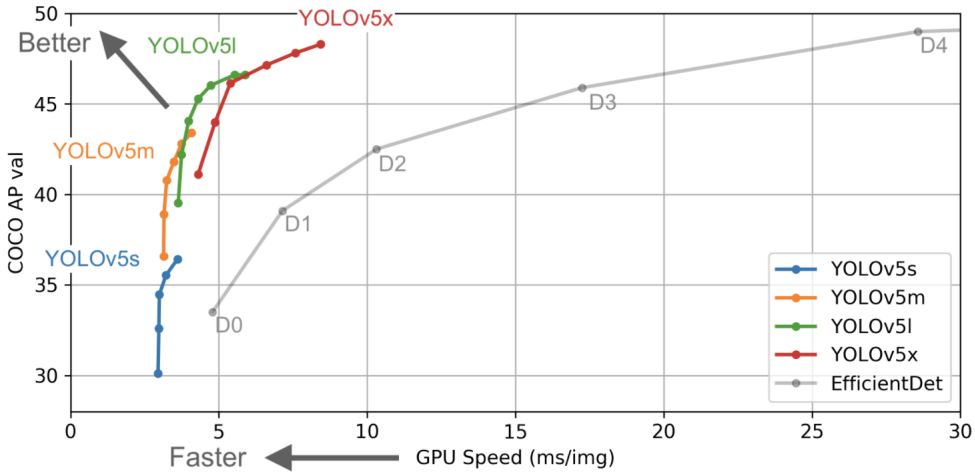

Figura 2.1: Benchmark de YOLOv5[8].

2.1.4. Conceptos clave

2.1.4.1. Intersection over union

También conocido como IOU, el *Intersection over union* corresponde al coeficiente entre la intersección y la unión de las *bounding boxes* de predicción y *ground truth*[3]. En general, el IOU es utilizado como métrica para filtrar si un resultado entregado por un detector es aceptable o no.

[3] Información que se sabe que es real o verdadera, proporcionada por observación o medición. Esta se utiliza para medir la veracidad de una inferencia.

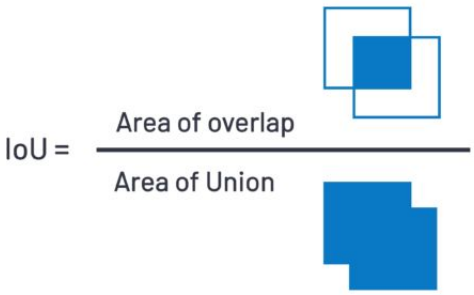

Figura 2.2: Intersection over Union.

2.1.4.2. Non maximal supression

En el caso que exista solapamiento entre múltiples entidades, el uso de *Non maximum Suppression* o NMS, ayuda a llevar a cabo una selección. Más específicamente, en el área de visión computacional se ocupa con el motivo de rebajar el número de *bounding boxes* entregadas por un detector en el caso que exista superposición entre ellas.

Por lo general, el algoritmo sigue los siguientes pasos:

1. Elegir del conjunto de predicciones generadas aquella que tenga mayor grado de confianza, ingresando esta al conjunto que se va a preservar.
2. Comparar con el resto de las predicciones, en el caso que alguna supere un umbral de IOU, se descarta.
3. Si aún quedan predicciones sin descartar, se vuelve al primer paso.

2.1.4.3. Precisión de punto flotante

En computación, la aritmética de punto flotante es una herramienta utilizada para aproximar números reales a través de una fórmula. Como consecuencia de la aproximación, las operaciones que la involucran traen consigo un *trade-off*[4] entre velocidad y precisión. Por esta razón, este tipo de aritmética se utiliza cuando el tiempo empleado en las operaciones es relevante, en especial en sistemas que involucren números reales muy grandes o pequeños.

De la misma forma, la *floating point precision* o FPP, hace referencia al tipo de formato utilizado en la representación de punto flotante. En general, FP32 es el formato más utilizado y viene como predeterminado en la mayoría de los lenguajes de programación. También se conoce como "precisión simple" y es aceptado por todos los ordenadores modernos. Como sugiere su nombre, se caracteriza por ocupar 32 bits en la memoria del procesador.

Por otro lado, también se tiene un formato llamado FP16, el cual se conforma por 16 dígitos. Comúnmente referido como "*half precision*", este tipo de número de punto flotante

[4] El término se utiliza en ingeniería para referirse a un tipo de relación entre variables. En específico, la relación consiste en que el aumento o disminución de una variable conlleva el comportamiento contrario por parte de la otra.

ofrece una mayor velocidad de inferencia que FP32, lo que logra sacrificando precisión. La importancia de estos para la inteligencia artificial radica en la posibilidad de elección entre mayor precisión o menor tiempo de inferencia. Utilizar FP16 se aplica comúnmente para acelerar los procesos de entrenamiento de redes neuronales y aumentar la velocidad que se emplea en generar detecciones. Sin embargo, por lo general su uso viene con la restricción de que la operación objetivo se realice en una GPU o TPU.

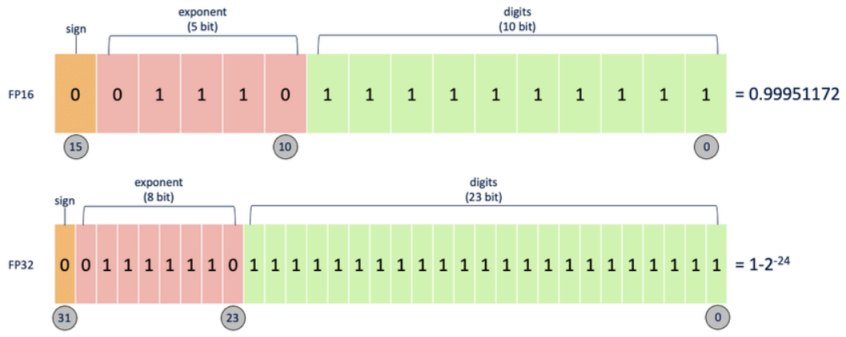

Figura 2.3: Formatos de punto flotante.

2.1.4.4. Pytorch

Pytorch es una librería de código abierto especializada en manejo de tensores. Una de las principales características que hacen de esta librería una herramienta de gran utilidad para los algoritmos de inteligencia artificial, es que tiene integrada en su diseño una plataforma que permite el uso de unidades de procesamiento gráfico de NVIDIA.

Esta plataforma se conoce como CUDA y su relevancia yace en que permite llevar a cabo procesamiento paralelo y acelerar tiempos de inferencia. Junto a esto, al ser una de las librerías más conocidas que permiten el manejo de tensores, lo que facilita distintas aplicaciones que involucren visión computacional y procesamiento de lenguajes naturales.

Habiendo sido mayoritariamente desarrollada por el laboratorio de investigación de inteligencia artificial de Facebook, esta librería es de libre uso y código liberado. A pesar de la interfaz de Python es de uso más común y el foco principal de desarrollo, Pytorch también tiene una interfaz disponible para C++.

2.1.4.5. TensorRT

TensorRT es un kit de desarollo de software o "SDK" creado por NVIDIA, este incluye un optimizador de inferencia y tiempo de ejecución para aplicaciones de *deep learning*. En general, su uso entrega un alza en el rendimiento y una disminución en latencia, por lo que es una herramienta muy útil en las etapas de despliegue.

A continuación, se describen los elementos que permiten la optimización:

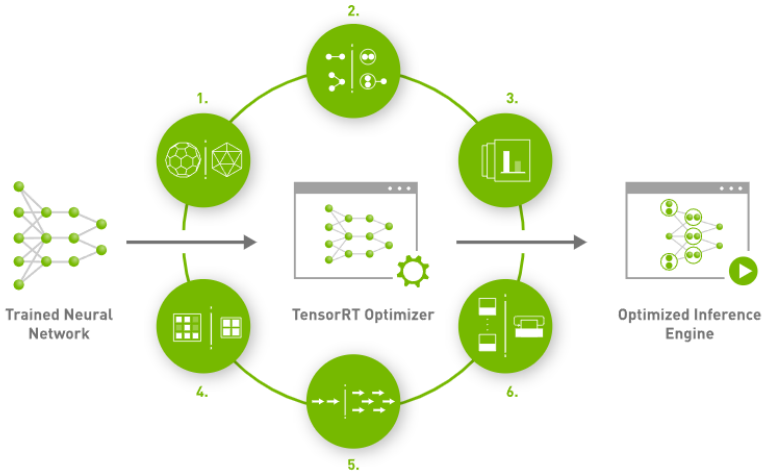

Figura 2.4: Elementos de TensorRT.

1. **Calibración de precisión para pesos y funciones de activación**: Maximiza el rendimiento al quantizar los modelos a INT8[5] sin generar perdida de precisión.
2. **Fusión de capas y tensores**: Optimiza el uso de memoria y ancho de banda de la GPU al usar nodos en el kernel.
3. **Auto-ajuste de kernel**: Selecciona las mejores capas de datos en función de la plataforma GPU destino.
4. **Memoria dinámica para tensores**: Minimiza la huella de memoria y reutiliza la memoria para tensores de manera eficiente.
5. **Ejecución en *multi-stream***: Entrega un diseño escalable para el procesamiento de múltiples entradas en paralelo.
6. **Fusión tensorial**: Optimiza las redes neuronales recurrentes en pasos de tiempo con kerneles generados dinámicamente.

2.2. Multiple Object Tracking

El objetivo de un algoritmo de *tracking* es hacer un seguimiento de objetos en un vídeo. Para esto, se incorpora la información temporal con aquella entregada por un módulo de detección. Posteriormente, se asocian las *bounding boxes* entre fotogramas[6] y se interpretan como un conjunto de trayectorias con alta precisión.

Con el fin de entregar la información generada, los objetos presentes en cada fotograma de una secuencia se suelen representar por tres elementos principales. El primero es una *bounding box*, la cual sirve para indicar la zona de la imagen donde se encuentra un objeto. El segundo corresponde a un ID, el cual se utiliza para distinguir a un ente en específico como

[5] Formato de datos para números enteros compuestos por 8 bits.
[6] Imagen que forma parte de una secuencia para conformar un video.

objeto único a lo largo de un periodo de tiempo. Por último, se suele indicar que fotograma es el que se esta describiendo.

Según los fotogramas que se ocupen en el análisis, los algoritmos de *multiple object tracking* se pueden dividir en dos categorías. Primero, la categoría en que se utiliza información de futuros fotogramas del vídeo para deducir la identidad de un objeto presente en un determinado fotograma se conoce como *Batch tracking*. Por otro lado, existe el *Online tracking*, el cual ocupa información presente y pasada para llegar a conclusiones con respecto a un fotograma determinado. En general, los algoritmos de *batch tracking* entregan mejores resultados, sin embargo, los algoritmos de *online tracking* permiten entregar resultados en tiempo real.

2.2.1. Filtro de Kalman

El filtro de Kalman, introducido por Kalman (1960), consiste en un algoritmo recursivo que provee una solución eficiente al método de mínimos cuadrados ordinarios. Esta solución permite estimar el valor futuro de los estados de las variables[9].

Este filtro se divide en dos partes principales, predicción y corrección, ambos usados para obtener una estimación del comportamiento futuro de las variables. Combinando la información disponible en el momento (t-1) y, posteriormente, actualizando dicho estimador con la información adicional disponible al momento t.

2.2.2. Centroid

El seguimiento de objetos ocupando Centroid se basa en utilizar la distancia euclidiana entre los centroides de los objetos detectados entre dos *frames*[7] consecutivos de un vídeo[10]. Con este fin, primero se generan *bounding boxes* de detección para los objetos de interés en cada *frame*, ya sea utilizando técnicas clásicas o *deep learning*. Luego se calculan los centroides desde estas *bounding boxes* y se prosigue con el algoritmo.

En concreto, los pasos del algoritmo son los siguientes:

1. Se detectan los objetos utilizando un cuadro delimitador para el *frame* en el tiempo t-1.
2. Se calculan los centroides de los objetos detectados para el tiempo t-1.
3. Se generan las *bounding boxes* de detección para el *frame* del tiempo t.
4. Se calculan los centroides de los objetos detectados para el tiempo t.
5. Se calcula la distancia euclidiana entre los centroides de todos los objetos detectados en los fotogramas t-1 y t.
6. Si la distancia entre el centroide del tiempo t-1 y t es menor que un umbral, se determina que es un mismo objeto en movimiento.
7. Si la distancia entre el centroide en el tiempo t-1 y t excede el umbral, se considera como la aparición de un objeto nuevo.

[7] Equivalente al término fotograma.

2.2.3. Simple Online and Realtime Tracking

Un avance en relación a Centroid, *Simple online real-time tracking* o Sort, se basa en contrastar detecciones generadas para cada *frame* con predicciones hechas con el filtro de Kalman[11]. Añadido a esto, se introduce un umbral de detecciones mínimas para empezar a seguir a un objeto y un umbral de tolerancia que indica el número de veces que una asignación entre detección y predicción puede fallar sin reasignar un ID.

Este algoritmo está compuesto por 4 componentes principales:

1. **Detección.**
2. **Estimación.**
3. **Asignación.**
4. **Creación y destrucción de identidades de seguimiento.**

El primer paso es generar *bounding boxes* utilizando un modelo de detección de los objetos a los cuales interesa realizar un seguimiento. Posteriormente, se utiliza el filtro de Kalman con el fin de generar predicciones sobre la posición de un objeto para un momento dado. Posteriormente, se calcula el IOU de esta estimación con las detecciones del mismo fotograma, generando una matriz de costo de asignación. Para resolver el problema de asignación se utiliza el algoritmo húngaro[12]. Finalmente, si el IOU calculado supera un umbral mínimo, se considera un caso positivo, si esto ocurre más que cierto número de veces, se crea un ID. Al mismo tiempo, si un objeto con un ID asignado no cumple con ser un caso positivo por un número consecutivo de fotogramas mayor a un umbral en específico, el ID es eliminado.

2.2.4. Deepsort

Expandiendo la metodología de Sort, Deepsort[13] añade una métrica adicional de similitud entre detección y predicción. Con este motivo, se añade un descriptor de apariencia a la arquitectura del *tracker*, con el cual se extrae la distancia de Mahalanobis y se filtra acorde.

El trabajo se desarrolla utilizando un descriptor de apariencia basado en *deep learning*, en este caso se utiliza la red *Omni-scale*.

2.2.4.1. Omni-scale Network

Inicialmente introducida para la re-identificación de personas, *Omni-scale Network* o Osnet, se compone por redes neuronales convolucionales. Su diseño está pensado para la extracción de características de objetos de interés en múltiples escalas[14].

Específicamente, el trabajo realizado se desarrolla utilizando pesos pre-entrenados. Esto se hace con el fin de extraer características de vehículos producto de la base de datos *ImageNet*. Además, se destaca que la versión utilizada "*Osnet_x1_0*" es la más pequeña, la cual esta compuesta por 200000 parámetros.

2.2.5. Formatos relevantes

Para representar la información utilizada por los distintos módulos, existen distintos formatos aceptados como norma en la academia. En específico, se describen formatos distintos

usados por la red de detección de objetos para representar los objetos que captura, en adición a los usados por los *tracker* para entregar información de los objetos seguidos a lo largo de una secuencia de *frames*.

2.2.5.1. Anotaciones de YOLO

El formato usado para entrenar YOLO*v5* es definido por *ultralytics*. El método consiste en relacionar un archivo de texto con una imagen, donde cada línea del archivo representa un objeto y esta compuesta por cinco números. El primer número, es un entero que indica la clase a la que pertenece el objeto descrito, los siguientes dos describen el centro del objeto, finalmente los últimos dos indican el ancho y altura de la *bounding box*. Es importante destacar que los últimos cuatro valores se normalizan utilizando el tamaño de la imagen sobre la cual generan las detecciones.

```
<object_class> <x> <y> <width> <height>
```

Figura 2.5: Formato de anotaciones de YOLO.

2.2.5.2. Anotaciones de *tracking*

Por otro lado, el formato de anotaciones del *tracker* es el mismo usado en el *MOT challenge 2016*[15]. Análogo al formato mencionado, las anotaciones se presentan en un archivo de texto, donde cada línea representa un objeto único. Sin embargo, las líneas se componen por seis números. El primero, representa el número de *frame* de la secuencia sobre la que se entrega resultados, el segundo corresponde al ID de objeto único y finalmente los últimos cuatro representan la *bounding box* que rodea al objeto anotado.

```
<frame> <id> <bb_left> <bb_top> <bb_width> <bb_height>
```

Figura 2.6: Formato de anotaciones de tracking

Capítulo 3

Estado del Arte

Con el fin de extraer una idea sobre el alcance obtenible bajo las condiciones impuestas, se realiza una revisión de trabajos pasados. En particular, se buscan trabajos donde las soluciones involucren contextos de vigilancia, detección o *tracking* de objetos y procesamiento basado en *edge-computing*. Adicionalmente, se presenta investigación sobre el impacto de la tasa de FPS usada en algoritmos de *tracking* sobre la exactitud de los resultados obtenidos.

3.1. Robot autónomo basado en el reconocimiento de la placa de identificación

En el trabajo mencionado se diseña un robot autónomo que opera dentro de los confines de un hospital y tiene como función reconocer sus distintas habitaciones[16]. La solución se logra diferenciando las habitaciones a través del reconocimiento de su placa de identificación, para esto se emplea YOLO*v4* como modelo de detección. El sistema es conformado por una Jetson Nano 2GB como hardware para el despliegue de algoritmos y un robot de cuatro ruedas para proporcionar movilidad.

Dados los costes humanos generados en el personal de la salud por el COVID-19, una solución robótica que disminuya el contacto humano es de gran utilidad para prevenir el contagio. Debido a los bajos requerimientos energéticos de la Jetson Nano 2GB y su asequibilidad económica, la solución propuesta es costeable por múltiples hospitales y clínicas.

Finalmente, se concluye del trabajo estudiado que bajo las especificaciones mencionadas, al alimentar el sistema con 10W e imponer un umbral de confianza de 0.8 a dos metros de distancia, es posible lograr un *accuracy* de 94.73 % a una tasa de 25.692 FPS.

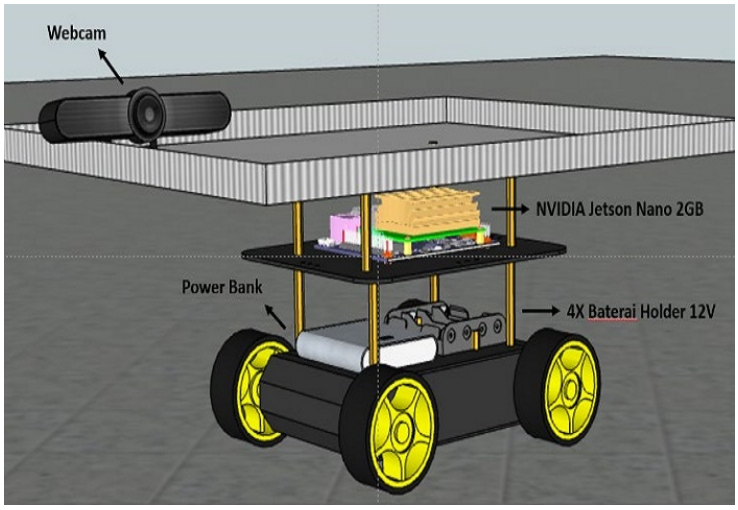

Figura 3.1: Vehículo autónomo[16].

3.2. Detección de objetos en Jetson Nano sobre un paso peatonal

El siguiente trabajo propone la implementación de una red para la detección de pasos peatonales[17]. Esta detección se hace desde un vehículo en movimiento, genera resultados en tiempo real y utiliza un sistema de despliegue basado en *edge-computing*. En concreto, la red utilizada para detección es YOLO*v5* y el *edge-device* donde se realiza la inferencia es una Jetson Nano.

Frente a la necesidad de continua gestión para un largo número de vehículos, es imposible mantener el orden dependiendo exclusivamente de la policía. Por lo cual, cámaras montadas sobre vehículos son una parte importante de los "Intelligent traffic managment systems". La información de detección de pasos peatonales y comportamiento de cruce de vehículos proveen la base para la toma de decisiones de ITMS. La relevancia del trabajo recae en encontrar resultados precisos y en tiempo real pese a las limitaciones de poder de procesamiento ligadas al *edge-computing*.

La solución lograda tras el trabajo se compone por una combinación entre "*Squeeze-and-excitation networks*"[18] y YOLO*v5*. Adicionalmente, se acota la imagen de entrada enfocándose en zonas en específico o "*Regions of interest*". Por último, con el fin de interpretar cuantos cruces peatonales el vehículo ha atravesado, se aplica el algoritmo SSVM[19]. Como conclusión, el modelo propuesto entrega resultados de 33.1 FPS de velocidad y un *F1-score* de 94.8 %.

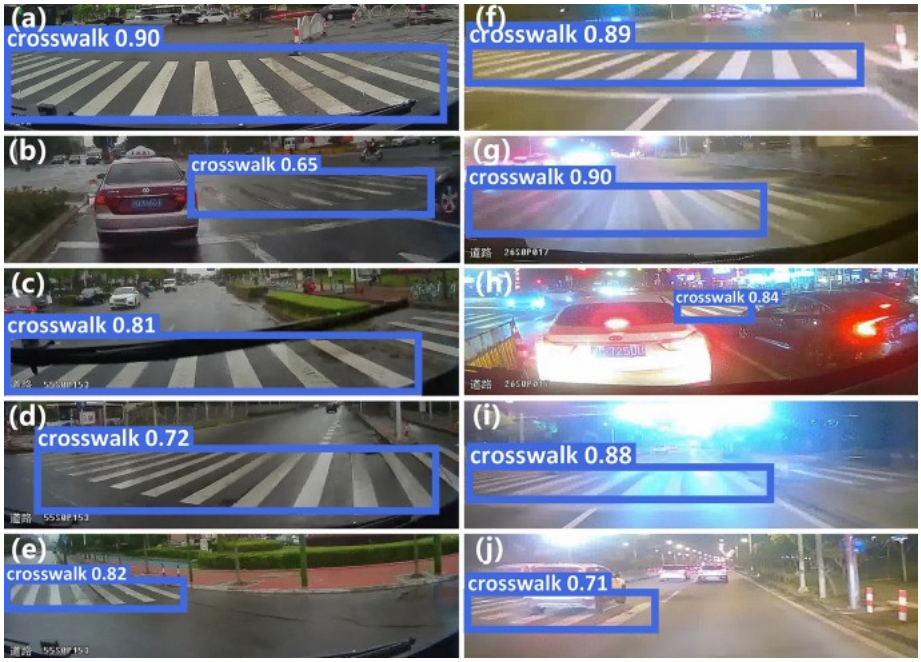

Figura 3.2: Detección de pasos peatonales[17].

3.3. Estudio comparativo de single object tracking sobre Jetson Nano

Este caso de estudio propone el uso de redes siamesas en un equipo de vigilancia montado sobre un vehículo aéreo no tripulado[20]. Dichas redes tienen como función llevar a cabo *tracking* de objetos sobre un UAV. El trabajo consiste en generar un *benchmark* de 14 algoritmos SOT[8], en donde se compare el rendimiento de precisión y velocidad de inferencia de las redes propuestas.

Del trabajo se desprende el alcance obtenible de las soluciones que utilizan redes siamesas. Se destaca que las velocidades de inferencia más altas son obtenidas por LightTrack[21], SiamRPN[22] y SiamCar[23], con tasas de FPS de 8.3, 6.6 y 6.25, respectivamente. Vale la pena notar que gracias a la distancia entre los objetos y el observador, el desplazamiento relativo por *frame* es bajo, lo que permite que tasas de FPS menores a 10 puedan entregar resultados de buena calidad.

3.4. Determinando la tasa de FPS necesaria para multiple object tracking

El trabajo de investigación de Anup Mohan *et al.* titulado "*Determining the Necessary Frame Rate of Video Data for Object Tracking under Accuracy Constraints*", explora el impacto de la velocidad relativa entre el observador y los objetos sobre la tasa de FPS necesaria[24]. En este trabajo se observan 4 contextos: una persona trotando, ciclistas, pelotas de basketball en desplazamiento y movimiento peatonal en una calle, el comportamiento del *trade off* entre

[8] Single object tracker.

tasa de FPS utilizada y exactitud de resultados.

Con este fin, se contrastan las métricas de *precision distance* y *object displacement* para analizar como afectan las variaciones de *frame rate*[9] utilizado a los resultados obtenidos. Puntualmente, la primera se define como la distancia entre los puntos centrales de hipótesis y *ground truth* para un objeto en específico. Por otro lado, la segunda se define como la distancia en pixeles que un objeto abarca entre *frames*.

Para evaluar los resultados de la experimentación realizada, el trabajo considera la *precision rate* como métrica objetivo. La cual se define como el porcentaje de *frames* en los cuales la *precision distance* se encuentra debajo de un umbral.

Finalmente, la investigación descrita logra demostrar que con algunas situaciones es posible disminuir hasta un 80 % la tasa de FPS manteniendo la *precision rate* 60 %.

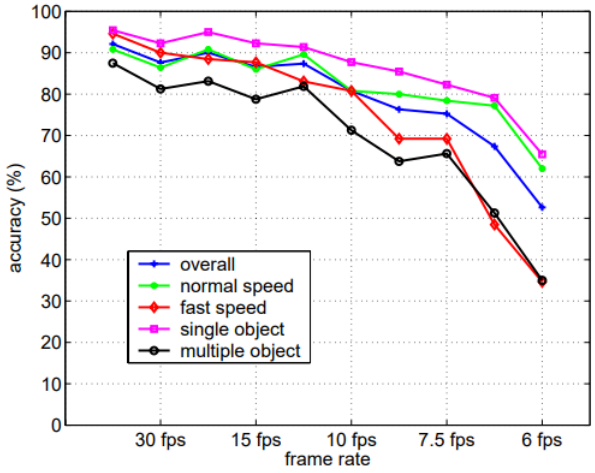

Figura 3.3: Cambio de accuracy según frame rate[24].

[9] Tasa de FPS.

Capítulo 4

Metodología

4.1. Arquitectura e implementación

Considerando los requerimientos del proyecto, se propone un diseño para la solución compuesto por tres módulos distintos: detector de objetos, *multiple object tracker* y extractor de información. Recalcando que los componentes extremos del sistema, o sea la captura y recuperación de datos, escapan del alcance del trabajo.

Se selecciona esta arquitectura debido a la facilidad que provee para localizar problemas en su funcionamiento. Sumado a esto, el extenso alcance en el desarrollo de los elementos responde a la posible necesidad de mejorar el rendimiento del sistema en el futuro. Finalmente, se destaca la extensibilidad de los primeros dos módulos, lo que posibilita su reutilización en futuros proyectos.

4.1.1. Arquitectura

La arquitectura modular de la solución propuesta funciona ingresando un vídeo de la calle a analizar, el cual se transmite y procesa en tiempo real por el sistema. Por último, este entrega anotaciones que describen la situación del lugar. Los elementos que la componen, en conjunto con sus respectivas funciones son las siguientes:

- **Detector de objetos**: Identificar clases objetivo y sus *bounding boxes* en los fotogramas.
- ***Multiple object tracker***: Realizar un seguimiento de los objetos de interés captados en el módulo anterior a lo largo de una serie de fotogramas.
- **Extractor de información**: Interpretar los resultados de los módulos anteriores para detectar eventos de interés y extraer información relevante.

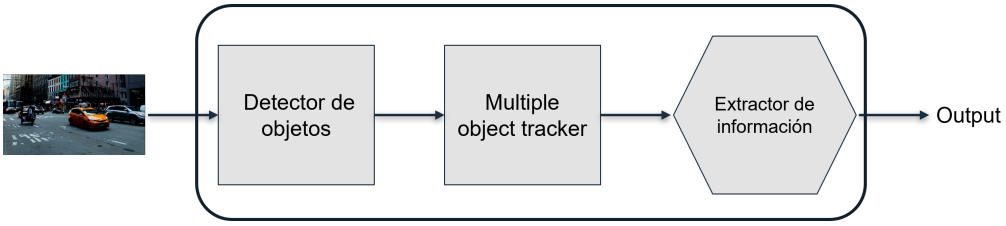

Figura 4.1: Módulos de sistema.

4.1.2. Implementación

Para llegar a una configuración definitiva, se requiere definir un modelo adecuado para cada módulo. Junto a esto, se necesita llevar a cabo una calibración de distintos parámetros del sistema. Con este fin, se propone llevar a cabo distintas pruebas para cada módulo, verificando qué valores entregan los mejores resultados. Los principales puntos de interés de los experimentos son: la calidad de los resultados de una determinada configuración y como esta se comporta en relación al hardware.

Por un lado, la calidad de los resultados busca reflejar la veracidad con que se describe la situación del lugar analizado. Esto se demuestra en la exactitud de las clases y *bounding boxes* que captura el detector por sí solo, junto con la fidelidad que entregue el *tracking* para reconocer a los objetos en una serie de fotogramas como únicos.

Luego, para analizar el comportamiento de los componentes con el hardware, se extrae la velocidad de procesamiento que entregue una determinada configuración. Finalmente, la tarea central de las pruebas es encontrar un punto de operación en que el *trade-off* entre precisión de resultados y velocidad de inferencia entregue los mejores resultados.

Otro punto que se quiere explorar, corresponde al de una posible disminución de *frame rate* para la transmisión y envío del vídeo. Esta búsqueda se realiza bajo la condición de no afectar la calidad de los resultados. La motivación de esto es disminuir el consumo energético y el flujo innecesario de datos.

Cabe destacar que las pruebas se centran en los primeros dos módulos, dado que el funcionamiento del extractor es una consecuencia directa de los resultados entregados por estos. En consecuencia, la evaluación del detector y *tracker* se hace de manera cuantitativa, extrayendo métricas y observando comportamientos que se consideren pertinentes.

En contraste, el último módulo se evalúa a través de una inspección de resultados, enfocándose en casos borde que se consideren atingentes a lo que se busca registrar. El objetivo de esto es verificar que su implementación sea correcta.

4.2. Módulo 1: Detector de objetos

Se propone que las pruebas del detector de objetos estén compuestas por dos secciones. La primera, está dirigida a estudiar la velocidad de inferencia. En particular, se tiene como objetivo registrar el tiempo empleado en realizar el *feed forward* de la red neuronal estudiada. Por otro lado, la segunda se enfoca en medir la calidad de los resultados que se entregan.

4.2.1. Versiones y configuración

Se ocupa el modelo YOLO*v5* como pilar central en la detección. Utilizando este modelo, se varían los parámetros involucrados y se estudian los resultados que se obtienen. Los componentes que se varían son: tamaño de la red a utilizar, motor de inferencia para despliegue, precisión de punto decimal y el tamaño imagen que ingresa al modelo.

La elección de YOLO*v5* como detector recae en la calidad de sus resultados, velocidad de inferencia y bajo tiempo de entrenamiento. En adición a esto, se destaca que existe el suficiente trabajo hecho involucrando la arquitectura, lo que permite contrastar resultados con otros trabajos académicos.

Por otro lado, la elección de las arquitecturas que participan de las pruebas se debe a que son las más pequeñas que proporciona YOLO. Igualmente, las resoluciones utilizadas en la experimentación se deben a que 640 es el tamaño de imagen predeterminado para los modelos P5. Luego, se toman valores menores a este para participar de las pruebas.

4.2.2. Objetos a detectar y Bases de datos

Dado el contexto en que se sitúa el despliegue, el modelo se configura con las clases objetivo: persona, bicicleta, automóvil, motocicleta, bus y camión. Para la procuración de las imágenes necesarias, se utilizaron las bases de datos de ACID[25], COCO, INDIA[26], BIKED[27] y VOC[28]. Asegurando un total de 78883 imágenes.

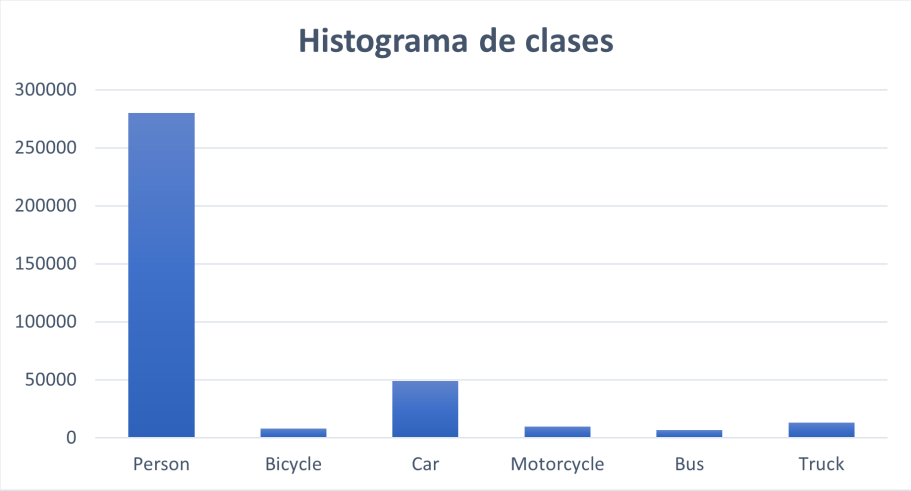

Figura 4.2: Número de etiquetas por clase.

Frente al bajo número de imágenes obtenidas y la diferencia de representación entre las clases, se usan *augmentations* para generar más imágenes. Posteriormente, se hace una división para las funciones de entrenamiento, validación y prueba.

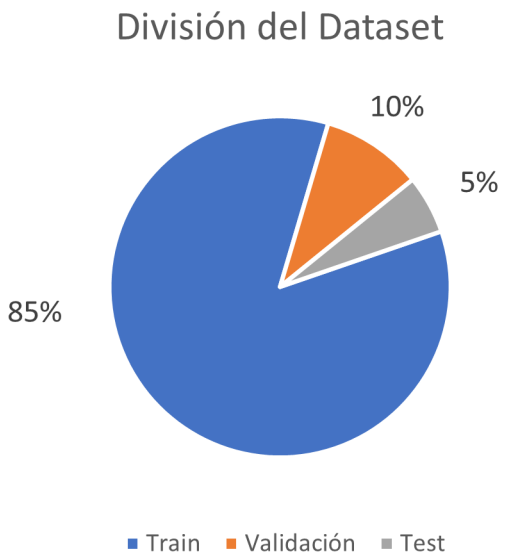

Figura 4.3: División de entrenamiento, validación y prueba.

Tabla 4.1: Composición de clases para conjuntos de entrenamiento, validación y prueba.

Entrenamiento	Validación	Test
267672	31491	15745
7512	884	442
46747	5500	2750
9257	1089	545
6488	763	382
12377	1456	728

4.2.3. Velocidad de inferencia

La unidad de medida utilizada para representar el rendimiento en hardware son los FPS. Experimentalmente, la medición se realiza de igual manera que para medir latencia, dado que el procesamiento se realiza con una sola *batch*.

Se reitera que únicamente se mide el tiempo invertido en la parte *feed forward* del proceso, en otras palabras, no se consideran los tiempos de pre-procesamiento y post-procesamiento.

$$FPS = \frac{1}{\text{frames procesados}} \tag{4.1}$$

4.2.4. Precisión de resultados

El análisis de la calidad de los resultados se centra en el cálculo de su *mean average precision*. El uso de esta métrica se sustenta en su uso frecuente en la academia e industria.

Con el fin de entender dicho valor, se definen primero las siguientes métricas y subproductos:

- ***Precision***: Representa el grado de exactitud que el modelo tiene para detectar objectos relevantes.

$$P = \frac{TP}{TP + FP} \tag{4.2}$$

- ***Recall***: Mide la habilidad que tiene un modelo para detectar todos los *ground truths*.

$$R = \frac{TP}{TP + FN} \tag{4.3}$$

- **Curva *Precision-Recall* o Curva PR**: Gráfico que representa la interacción entre *precision* y *recall* a distintos grados de confianza. En particular, la evaluación de modelos de detección de imágenes suele considerar como confianza el umbral de *intersection over union* utilizado.

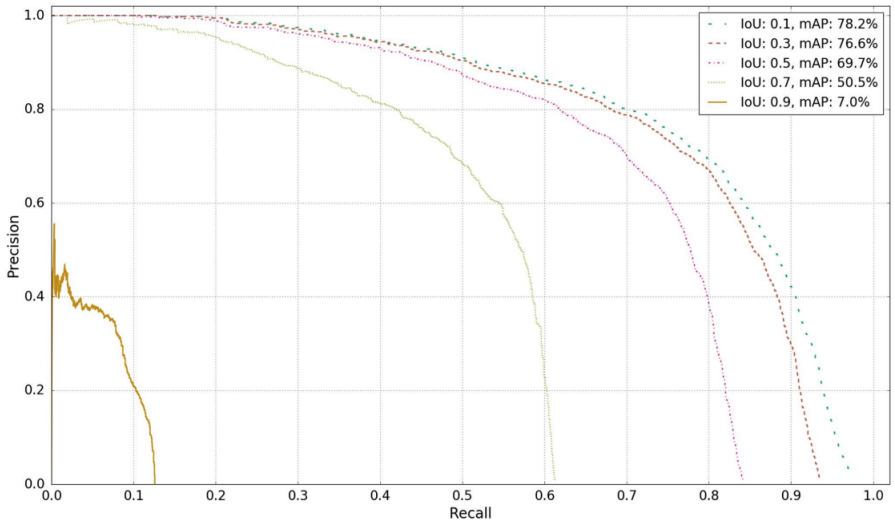

Figura 4.4: Ejemplo de curva PR para distintos umbrales de IOU

- **Precisión promedio o *Average precision***: Área bajo la curva de *Precision-Recall*, bajo un grado de confianza en específico.

$$AP\alpha = \int_0^1 p\left(r\right) \mathrm{d}r \tag{4.4}$$

- **mAP**: Promedio de la *Average precision* sobre todas las clases involucradas.

$$mAP\alpha = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{4.5}$$

Es importante destacar que existen variaciones sobre la aplicación exacta de *mean average precision*. La existencia de variantes se basa en su posible uso para diversas situaciones, sin embargo, todas buscan un enfoque similar. Al momento de escoger una específica, un factor determinante puede ser el impacto que tengan los falsos negativos o positivos sobre el contexto de aplicación. Por ejemplo, en la sección de evaluación del desafío de detección de COCO, la definición de AP y mAP es la misma.

Por otro lado, no existe una convención sobre que valor de IOU es el más apropiado. En general, la variante designada como $\mathbf{mAP}_{[0.5:0.95]}$, se utiliza para medir la calidad de un detector. Este se calcula promediando múltiples mAP para distintos grados de confianza de un intervalo. Específicamente, los mAP se calculan para un grupo de IOU, el cual empieza por 0.5, termina en 0.95 y contiene los valores intermedios tomando una distancia de 0.05 entre cada uno.

Asimismo, existen distintas formas de realizar el cálculo. Uno de los métodos más utilizados para su calculo es una técnica llamada *11-point interpolation*, introducida en el desafío de detección Pascal VOC.

Esta interpolación tiene como objetivo suavizar la curva original y que la evaluación sea menos susceptible para pequeñas variaciones en el ranking. Primero, se parte por calcular 11 valores de *precision* para 11 niveles equidistantes de *recall*, para posteriormente promediar dichos valores. Luego, para cada nivel de *recall*, se remplaza el valor de *precision* usado, con el máximo valor de *precision* a la derecha de ese nivel de *recall*.

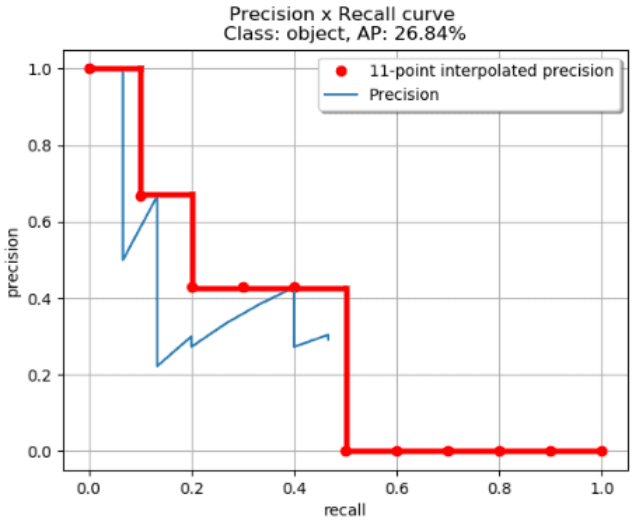

Figura 4.5: *11-Point Interpolation* aplicada en el cálculo de mAP[29].

4.3. Módulo 2: Tracker

Al igual que las pruebas propuestas para el módulo anterior, se plantea que los experimentos del módulo de *tracking* estén enfocados en determinar la velocidad del módulo y calidad de sus predicciones.

Primero, se parte la experimentación por estudiar el comportamiento del *frame rate* empleado por los distintos algoritmos de *tracking* en el procesamiento. El procedimiento consta en observar el rendimiento de los algoritmos al combinarlos con las distintas configuraciones del módulo de detección.

Posteriormente, una vez ya obtenidos los valores de velocidad, se selecciona un subconjunto para la evaluación de calidad de resultados. Para esto, se contrastan las predicciones obtenidas con anotaciones generadas utilizando un *script* de anotación propio de Sinantica.

4.3.1. Modelos explorados

Se utilizan tres modelos distintos de *tracking* para las pruebas: Centroid, Sort y Deepsort. Las pruebas constan de una combinación de los algoritmos seleccionados, junto con las configuraciones del detector anteriormente mencionadas. La elección de los modelos se hace tomando en cuenta su complejidad, calidad de resultados que entregan, junto con las herramientas y limitaciones presentes en el proyecto.

En específico, la elección de Centroid se debe a su simpleza, por lo que sirve como punto base de referencia en cuanto a precisión y velocidad. Adicionalmente, sirve de manera indirecta para observar cómo se desempeña el modulo de detección.

Por otro lado, Sort se escoge por ser uno de los mejores *tracker* que no involucran arquitecturas extras de *deep learning*, lo cual es relevante dado el consumo de memoria RAM asociado a esto.

Finalmente, la elección de Deepsort se sustenta en la calidad de sus resultados, junto con la posibilidad de estudiar el comportamiento de la tasa de FPS entregada por la Jetson Nano 2GB al tener que lidiar con dos redes neuronales a la vez.

Vale la pena mencionar que se evaluaron otros modelos basados en *deep learning* para que participaran de las pruebas. Sin embargo, se determinó con Sinantica que 3 era un número adecuado para el trabajo. Adicionalmente, se limito la selección a los modelos mencionados por los distintos aspectos que cubren con su experimentación. Sumado a esto, se consideraron: simpleza de implementación, extensibilidad y requerimientos de hardware.

Es relevante mencionar que las resoluciones utilizadas anteriormente no son las mismas que las aplicadas en las pruebas de esta sección. Previamente, se utilizó como criterio que las resoluciones elegidas fueran iguales a las dimensiones utilizadas por YOLO. En contraste, esta fase de la experimentación selecciona tamaños de imagen que sean análogos a la resolución que entrega la cámara en el lugar del proyecto.

4.3.2. Velocidad de inferencia

De igual manera que la medición previa de rendimiento en hardware, la unidad objetivo es el *frame rate*, la cual se calcula usando la ecuación 4.1. Sin embargo, el tiempo objetivo de medición abarca a todo el proceso, incluyendo pasos previos y posteriores a la aplicación del *tracker*. Esto debido a la relevancia de estudiar el tiempo real empleado por el proceso completo.

Además, disminuir el rango de estudio a únicamente la inferencia realizada por el *tracker* puede presentar múltiples problemas. Principalmente, esto es a causa de la complejidad de la interacción entre los *trackers* y el módulo de detección. No obstante, intentar aislar al periodo de medición puede generar una perdida de generalidad, al estar potencialmente ligada a un *input* específico.

La experimentación se lleva a cabo utilizando un fragmento de 5 minutos de vídeo, en el cual se captura la calle estudiada en el proyecto. Se destaca que la arquitectura actualmente instalada por Sinantica entrega vídeo en 20 FPS como máximo. Por consecuencia, valores de FPS mayores a este valor no se considera que entreguen un valor aplicable. No obstante, casos como el descrito se registran por su posible utilidad en aplicaciones futuras.

4.3.3. Precisión de resultados

Con el fin de evaluar la calidad de las predicciones hechas por el módulo, se utiliza como marco de evaluación las métricas utilizadas en el *MOT challenge 2016*. Esta decisión se respalda en su uso frecuente en trabajos de investigación.

Cabe aclarar los requisitos para que una predicción se considere un acierto. Primero, de manera similar a la evaluación de un detector de objetos, el IOU calculado entre las *bounding boxes* de hipótesis y *ground truth* debe superar un umbral. Segundo, el ID de objeto único asignado a la predicción debe ser idéntico al presente en el *ground truth*. De no cumplirse una de las condiciones la predicción generada no se considera como correcta. Luego, habiendo definido los requisitos, se definen las variables utilizadas en las métricas:

- ***Ground truth* o GT**: Número de objetos únicos.
- **FP**: Número de falsos positivos.
- **FN**: Número de falsos negativos.
- **IDs**: Número de cambios de ID.

De manera análoga a las métricas: *precision*, *recall* y *F1-score*, tradicionalmente utilizadas en la evaluación de detección de objetos, se definen los siguientes tres valores:

- **IDP**: *Precision* relativa a la asignación de ID.

$$IDP = \frac{TP}{TP + FP} \tag{4.6}$$

- **IDR**: *Recall* relativo a la asignación de ID.

$$IDR = \frac{TP}{TP + FN} \tag{4.7}$$

- **IDF1**: *F1-score* relativo a la asignación de ID.

$$IDF1 = \frac{2 * precision * recall}{precision + recall} \tag{4.8}$$

Por otro lado, las dos métricas consideradas de mayor relevancia en la evaluación de un *tracker* son el *Multiple object tracker accuracy* (MOTA) y *precision* (MOTP). La primera métrica entrega una idea del funcionamiento general del algoritmo, calculando una relación entre el número de errores y número total de objetos únicos capturados en el vídeo. En contraste, la segunda mide la calidad de los *bounding box* de las predicciones que se consideraron correctas. Para el trabajo realizado MOTA es más importante en la evaluación que MOTP.

A continuación, se entregan las fórmulas de estas:

- ***Multiple object tracker accuracy* o MOTA**: Mide la exactitud de tanto el *tracker* como la detección.

$$MOTA = 1 - \frac{\sum FN + FP + IDS}{\sum GT} \tag{4.9}$$

- ***Multiple object tracker precision* o MOTP**: Mide la exactitud de localización de las *bounding boxes* detectadas.

$$MOTP = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{4.10}$$

Donde

d_t: Distancia entre el output de detección y ground truth

c_t: Total de coincidencias hechas entre el output de detección y ground truth

Vale la pena mencionar que evaluar *multiple object tracking* se considera notoriamente difícil en la academia. En particular, se ha explorado en distintos estudios la existencia de problemas en el uso de MOTA, llegando a la conclusión de que no siempre refleja de manera correcta el valor de los resultados[30]. No obstante, su valor aún es de relevancia en la evaluación de *tracking*.

Finalmente, se entregan tres valores que dan de manera más simple una noción de la calidad de los resultados:

- **Mostly tracked o MT**: Número de objetos correctamente seguidos por al menos el 80 % de los *frames* en que aparece.
- **Partially tracked o PT**: Número de objetos correctamente seguidos por un número de *frames* entre el 20 % y 80 % del total de su aparición.

- **Mostly lost o ML**: Número de objetos correctamente seguidos por un número de *frames* de no más del 20 % del total de su aparición.

4.3.3.1. Subconjunto experimental

Dado que la calidad de las predicciones puede ser impactada por la tasa de FPS, es necesario llevar a cabo múltiples pruebas para valores de FPS en específico. Con este fin, se plantea que la experimentación se realice para *frame rates* de valor 8, 10, 12, 14, 16, 18 y 20.

La selección de este grupo se debe a que se busca limitar la cantidad de pruebas necesarias, considerando que el conjunto mencionado es suficiente para retratar el comportamiento del sistema. Específicamente, 8 FPS se escoge como cota inferior dado a su tendencia a ocupar este lugar en proyectos similares. Por otro lado, 20 FPS se propone como cota máxima al igualar el *frame rate* de mayor valor posible entregado por el sistema.

Asimismo, se impone que la experimentación se haga para un grupo limitado de configuraciones. Dicho conjunto objetivo se determina según los resultados obtenidos en las pruebas de velocidad del módulo, junto a elementos relevantes que deben participar de las pruebas. Se considera como mínimo que las pruebas realizadas consten de la participación de los tres *trackers* mencionados en las secciones anteriores, utilizando las dos arquitecturas de YOLO*v5* estudiadas.

Finalmente, para determinar los tres parámetros faltantes de cada configuración: motor de inferencia, tamaño de imagen de ingreso y *floating point precision*, se siguen los siguientes pasos:

1. Filtrar variables por velocidad, descartando aquellas configuraciones que no cumplan con los *frame rates* objetivos.
2. Seleccionar entre las configuraciones restantes aquella que haya entregado mejores valores de mAP.

4.3.3.2. Aspectos de calibración

Dado que el funcionamiento del módulo de *tracking* va ligado a la calidad de resultados que entregue el detector de objetos, parte de la calibración se hace en relación a este. Junto a esto, se debe tener en cuenta que los FPS con que se procese el vídeo afecta la distancia que abarca un objeto de interés por fotograma, por lo tanto, es necesario asignar los parámetros de los *trackers* en coordinación con los resultados anteriormente obtenidos.

4.3.3.2.1. Parámetros de detección

- **Clases**: Indica qué objetos debe enfocar la búsqueda el detector.
- **Class Agnostic**: Configura que la búsqueda ignore el tipo de clase de los objetos o no.
- **IOU**: Umbral de *Intersection over union* utilizado al aplicar *Non max supression*.
- **Confidence**: Umbral de confianza utilizado en la detección.

4.3.3.2.2. Parámetros de tracking

- **Minimum hits**: Número de detecciones consecutivas para que el algoritmo de *tracking* comience a realizar el seguimiento.
- **Max age**: Número de *frames* de tolerancia, para que no se realice el seguimiento de un objeto previo a borrarlo.
- **Max distance**: Máxima IOU entre las *bounding boxes* generadas por el detector y filtro de Kalman para que un objeto sea considerado como seguido.
- **Max diference**: Máxima diferencia entre el vectores de *features* para llevar a cabo una asociación entre *frames*.

Se destaca que se calibran todos los valores relacionados detección para los tres *trackers*. Sin embargo, los parámetros de *tracking* no forman parte de la calibración de todos los modelos. Particularmente, Deepsort utiliza todos los valores, por otro lado, en Sort se utilizan únicamente los primeros tres. Finalmente, en Centroid no aplica ninguno.

4.3.3.2.3. Restricted zones

Con el fin de disminuir las falsas detecciones y aislar las predicciones a una zona, se crean áreas en donde si existe una detección, esta no se introduce al *tracker*. Se distingue esta metodología de la clásica selección de áreas de interés (ROI) por su mayor precisión al momento de seleccionar áreas que no presentan información relevante para un sistema.

Frente a esto, la selección de zonas se hace a través de una inspección de resultados, de la cual se extraen áreas del video donde falsos positivos o negativos ocurran frecuentemente.

Figura 4.6: Áreas restringidas

4.3.4. Conjuntos de validación y prueba

Para la calibración y evaluación se utilizan dos fragmentos de vídeo que capturan el lugar de despliegue del proyecto. El primero, se utiliza para definir los parámetros relacionados al funcionamiento del detector y módulo de *tracking*. Por otro lado, el segundo busca obtener métricas numéricas que retraten la calidad de los resultados obtenidos.

Se destaca que la elección de ambos se justifica porque retratan el funcionamiento normal de la zona estudiada. Igualmente, estas entregan un número de vehículos suficiente para el estudio del comportamiento de los *trackers*. En adición a esto, vale la pena mencionar que el motivo de la división de los conjuntos es disminuir un posible sesgo causado en la calibración.

Tabla 4.2: Conjuntos de validación para tracking.

Secuencia de Frames	Tiempo[segundos]	Número de objetos
Evaluación	17.3	6
Prueba	12.1	6

4.3.4.1. Etiquetador tracker bbox

Para utilizar el conjunto de prueba, es necesario poseer anotaciones que representen el funcionamiento presentado en la secuencia de *frames*. Con este motivo, se utiliza una librería desarrollada por Sinantica llamada "*etiquetador tracker bbox*", el cual se modifica para este propósito. Su funcionamiento se basa en anotar utilizando el cursor y teclado las *bounding boxes* presentes en cada *frame*, en conjunto con su ID de objeto único.

Los comandos para utilizar la librería se detallan a continuación:

- q: Salir del programa y guardar anotaciones a archivo JSON.
- n: Pasar al siguiente *frame*.
- s: Guardar anotaciones a archivo JSON.
- d: Borrar *bounding box* según ID de objeto único.
- i: Ingresar siguiente ID de objeto único a anotar.
- l: Borrar último objeto anotado.
- r: Borrar todos los objetos en pantalla.
- x: Mostrar u ocultar los IDs en pantalla.
- c: Mostrar u ocultar las *bounding boxes* en pantalla.

Vale la pena mencionar que el vídeo se ingresa al programa ya fragmentado en imágenes. En adición a esto, también se destaca que la librería posee un *tracker* propio para asistir con las anotaciones.

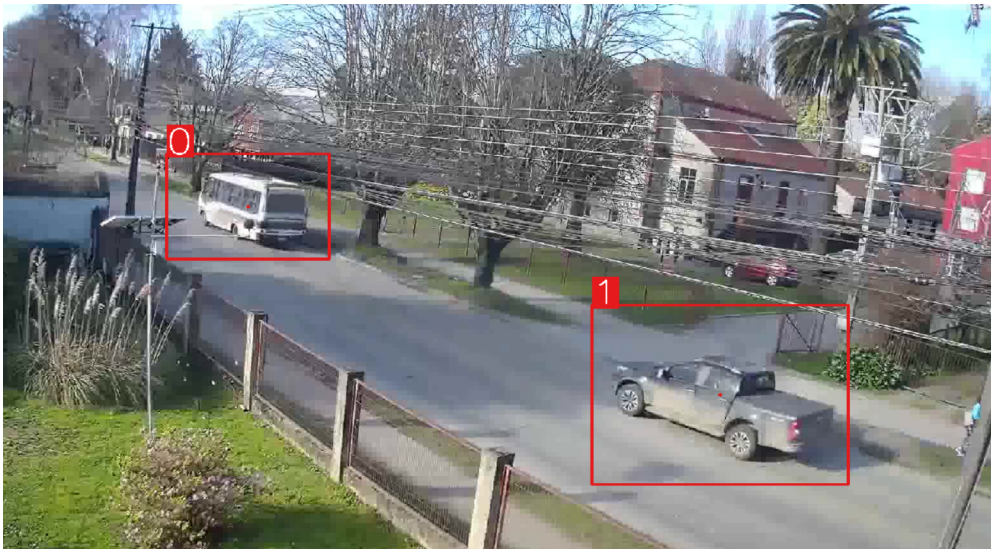

Figura 4.7: Etiquetador para validación de *tracking*.

4.4. Módulo 3: Detección de eventos y extracción de información

4.4.1. Conteo de vehículos

Se busca llevar un registro de los vehículos capturados en cámara, desglosados según tipo y hora de aparición. Para esto se utiliza el historial de objetos seguidos por el módulo de *tracking*, junto con su hora de aparición y clase.

4.4.2. Dirección de vehículos

En conjunto con el registro de vehículos, se desea catalogar la dirección en la que estos se mueven. Esto se logra calculando los centroides de las *bounding boxes* entregadas por el *tracker*. Posteriormente, se calcula un vector de desplazamiento usando la diferencia de coordenadas de los centroides de un mismo objeto, para *frames* consecutivos. Finalmente, se asigna "IZQ" o "DER" dependiendo del valor del delta calculado.

4.4.3. Peatón en zona de peligro

En adición al resto, se busca alertar cuando un peatón entra a la calle por un lugar no permitido. Para esto, se utiliza el algoritmo Ray-Casting[31], el cual verifica que un punto no se encuentre dentro de una zona. Se parte por representar la calle con un polígono que indique sus límites. Posteriormente, se filtran las *bounding boxes* de un fotograma para que resten únicamente las que involucran a personas. Finalmente, de los centroides de las detecciones que permanezcan, se aplica el algoritmo y se levanta una advertencia en el caso que sea pertinente.

Figura 4.8: Peatón en zona de peligro

4.4.4. Congestión vehicular

Se desea detectar si existe congestión vehicular en la calle observada, ya sea por poco movimiento de vehículos o un gran número de ellos. Con este fin, se verifica si el número de vehículos en pantalla supera un umbral, en el caso de que esto ocurra se registra el caso como positivo. Adicionalmente, se revisa si la velocidad de los vehículos presentes es menor a un umbral determinado, en el caso de que esto ocurra, también se registra como positivo. Con este propósito, se calcula la velocidad mediante una división entre el vector de desplazamiento y el número de *frames* en que ocurre. En el caso de que la velocidad de un vehículo sea menor a un umbral, se registra la existencia de una congestión.

4.5. Formato de resultados

Para representar los resultados que entrega el sistema se utiliza un archivo de texto. Es importante destacar que esta modalidad se debe a que la interfaz existente tiene como único objetivo posibilitar la evaluación del funcionamiento del sistema. Puesto que implementar una interfaz para la recolección de datos se escapa del alcance del trabajo.

Este se presenta a continuación:

Código 4.1: Formato de anotación de resultados.

```
# Formato propio
vehicle_format={'frame_id':frame_id,'objects':[],'congestion':0,'pedestrians':[],'count':0}

object={'bbox':[(x1,y1),(x2 ,y2)],'id':ID,'vehicle':1,'direc':'-'}

pedestrian={'in danger':endangered,'bbox':[(x1,y1),(x2 ,y2)]}
```

Donde “vehicle_format” es un diccionario en donde se encuentran los resultados completos y se conforma por cuatro elementos y dos arreglos.

Estos son:

1. **Frame id**: Número de *frame* relativo a la secuencia de imágenes que se utiliza.
2. **Objects**: Arreglo de vehículos detectados.
    a) **Bbox**: *Bounding box* del vehículo seguido.
    b) **Id**: ID de objeto único del vehículo seguido.
    c) **Direc**: Dirección en la que viaja el vehículo seguido.
3. **Congestion**: Variable que toma valores de 1 o 0, dependiendo de la existencia o falta de congestión vehicular.
4. **Pedestrians**: Arreglo que entrega información sobre los peatones y si estos se encuentran en la zona de peligro o no.
    a) **In danger**: Variable de valor 1 o 0 dependiendo de sí el peatón seguido esta en la zona de peligro o no.
    b) **Bbox**: *Bounding box* del peatón seguido.
5. **Count**: Conteo de los vehículos observados

Capítulo 5

Resultados Obtenidos

Este capitulo tiene como finalidad presentar los resultados de la experimentación realizada. Cabe mencionar que esta parte del informe se limita a exponer de manera gráfica los valores que se consideran más relevantes para el proyecto. Luego, se exhibe una apertura total de los datos numéricos obtenidos en el anexo.

5.1. Módulo 1: Detector de objetos

En la siguiente sección se presentan los productos experimentales obtenidos por las pruebas del detector de objetos. Adicionalmente, se indican detalles de las pruebas realizadas que se consideren relevantes.

5.1.1. Velocidad de inferencia

En la siguiente sección se presentan los resultados obtenidos para la experimentación de velocidad de inferencia. Para esto, la tasa de FPS es calculada a través de un promedio de los fotogramas por segundo obtenidos para 300 iteraciones.

Es necesario señalar que el *input* a procesar es un arreglo numérico generado al azar, el cual se diseña para que cumpla con las dimensiones necesarias. Sumado a esto, es importante mencionar que las resoluciones utilizadas en esta y las siguientes pruebas, se escogen de acuerdo al método que utiliza YOLOv5. En específico, las transformaciones aplicadas sobre las imágenes buscan que el número de pixeles sea un múltiplo de 32, esto debido a que de esta manera se espera que la velocidad de inferencia sea mayor.

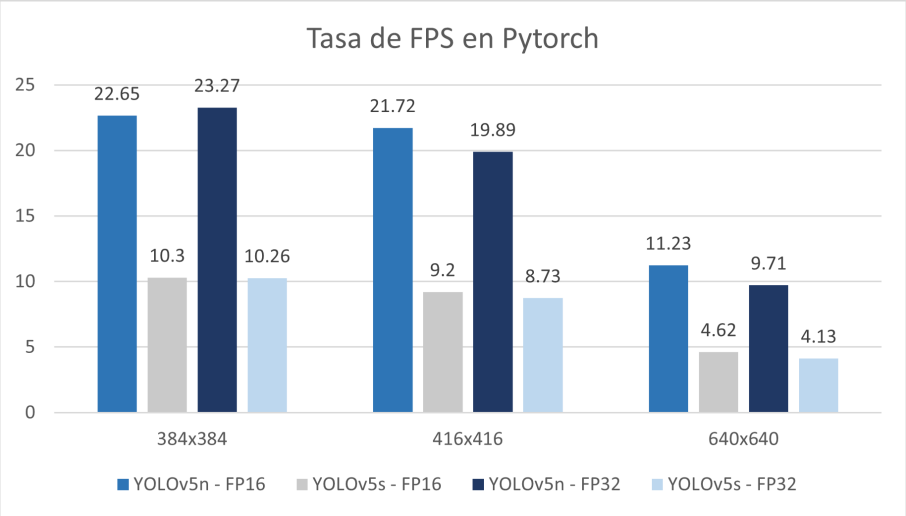

Figura 5.1: Tasa de FPS en módulo de detección Pytorch.

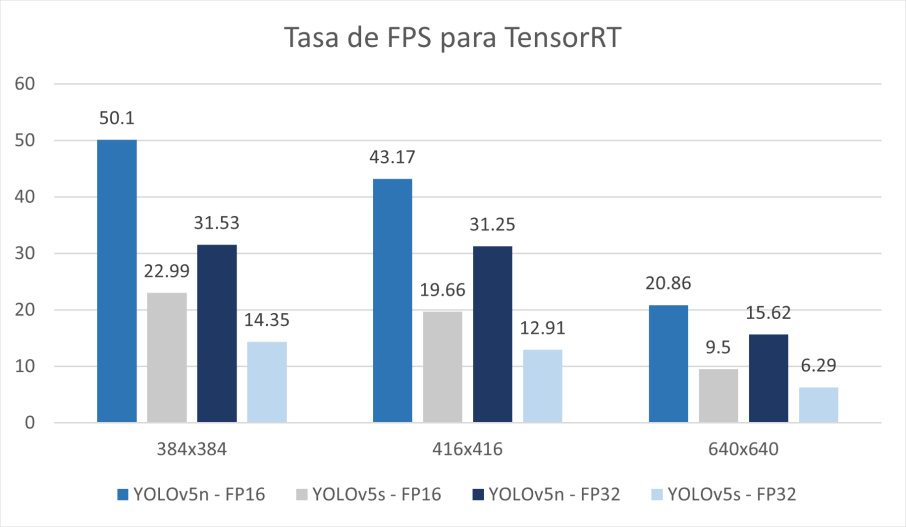

Figura 5.2: Tasa de FPS en módulo de detección TensorRT.

Se desprende de la figura 5.2, que la mayor velocidad de inferencia lograda es de 50.1 FPS, utilizando YOLOv5n con una imagen de tamaño 384×384, precisión FP16 y motor de inferencia TensorRT.

Por otro lado, es digno de mención que la mayor tasa lograda utilizando Pytorch es de $23.27FPS$, para el mismo tamaño de imagen y precisión. Finalmente, de ambas figuras se aprecia que el mejor rendimiento para YOLOv5s es de 22.99 FPS.

5.1.2. Rendimiento del modelo

Se analizan los modelos sobre un conjunto de evaluación de 700 imágenes. Para realizar el proceso se usan dos variaciones de mAP, una proveniente de la definición usada en COCO y otra que utiliza la parametrización definida por Pascal VOC. Específicamente, las variaciones mencionadas son $\mathbf{mAP}_{[0.5:0.95]}$ y $\mathbf{mAP}_{0.5}$.

Si bien se considera como estándar de medición para la calidad de un detector de objetos a la primera métrica, para fines del proyecto el valor de mAP usado por Pascal VOC se considera más importante. Se respalda esta preferencia en que se busca enfrentar la susceptibilidad de los algoritmos de *tracking* a los falsos negativos.

Para llevar a cabo la evaluación de los resultados de inferencia de TensorRT y Pytorch, se utiliza el *toolkit* proporcionado en "*review object detection metrics*"[32]. Se utiliza esta herramienta debido a que los pesos de formato *engine*, proporcionados por TensorRT, no son compatibles con la evaluación nativa de la librería de YOLO*v5*.

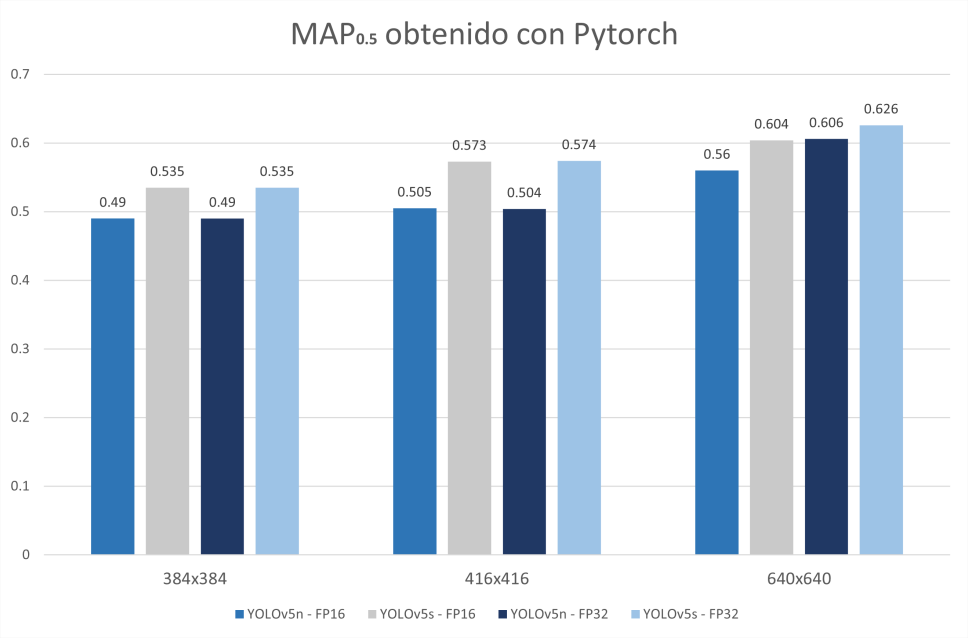

Figura 5.3: Mean average precision obtenida en Pytorch.

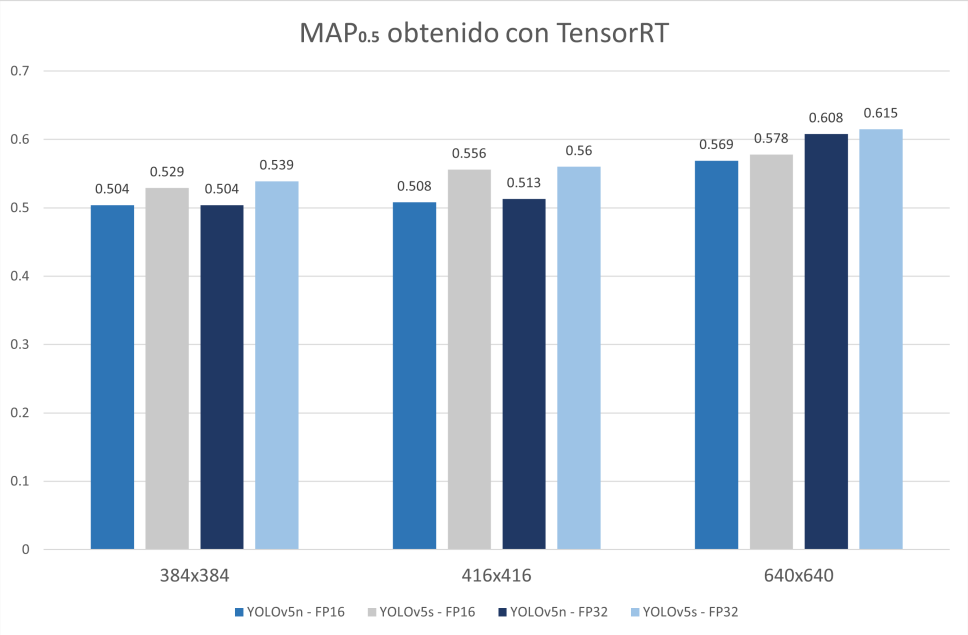

Figura 5.4: Mean average precision obtenida en TensorRT.

Se aprecia de la figura 5.3, que el mayor valor de mAP es de 0.626. Este máximo se logra desplegando YOLOv5s en Pytorch como motor de inferencia. Además, se observa en la figura 5.4 que el mayor **mAP** logrado utilizando TensorRT es de 6.15, nuevamente logrado con la arquitectura YOLOv5s.

Por último, se desprende de ambas figuras que el mejor resultado para YOLOv5n es de 0.608, al generar la inferencia con TensorRT. Se destaca que los tres valores mencionados se obtienen al utilizar una imagen de tamaño 640×640 y precisión FP32.

5.1.3. Relación entre exactitud y velocidad

A continuación, se combinan los resultados de las secciones anteriores para presentar el comportamiento del *mean average precision* en relación a la velocidad de inferencia.

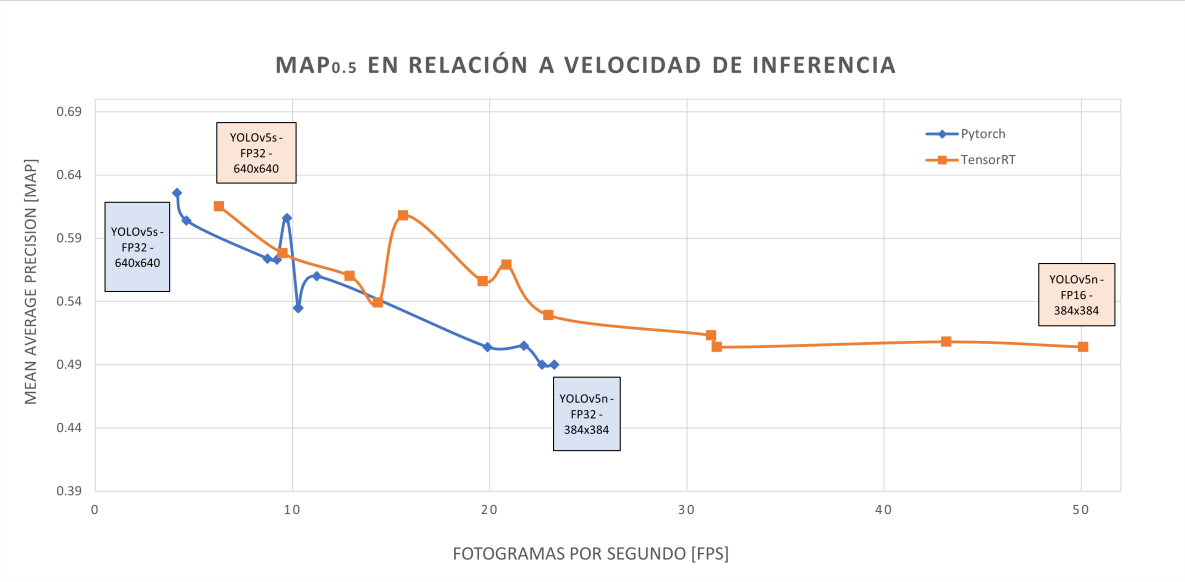

Figura 5.5: *Mean average precision* del módulo de detección según tasa de FPS obtenida. Se destacan los valores máximos de mAP y FPS para cada motor de inferencia.

Se desprende del gráfico que las velocidades de inferencia utilizando TensorRT son mayores. Por otro lado, se observa que la diferencia entre valores máximos y mínimos de mAP alcanzados entre Pytorch y TensorRT es despreciable.

En adición a esto, se concluye de la figura que el mAP obtenido entre 30 y 50 FPS ocupando TensorRT se mantiene relativamente constante. No obstante, se puede observar que el mAP de ambas arquitecturas tiende a bajar entre 0 y 20 FPS.

5.2. Módulo 2: Tracker

5.2.1. Velocidad de inferencia

La velocidad de inferencia de esta sección se extrae del promedio de tiempo de procesamiento sobre 300 imágenes. Asimismo, la resolución de estas imágenes se obtienen de una transformación que respeta la metodología de YOLO y las proporciones originales otorgadas por la cámara.

Es relevante aclarar que la medición de tiempo abarca: tiempo de pre-procesamiento, tiempo que consume el algoritmo y post-procesamiento.

5.2.1.1. Centroid

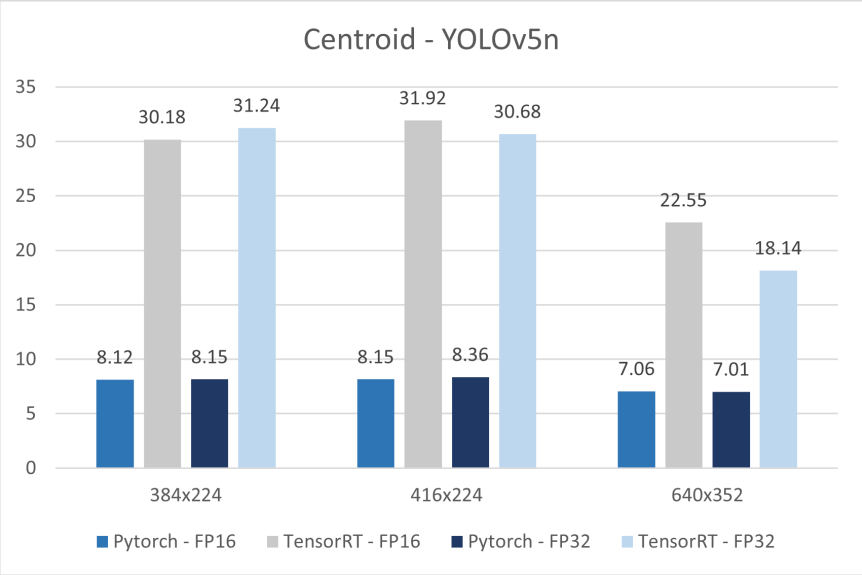

Figura 5.6: Tasa de FPS para Centroid usando YOLO*v5n*.

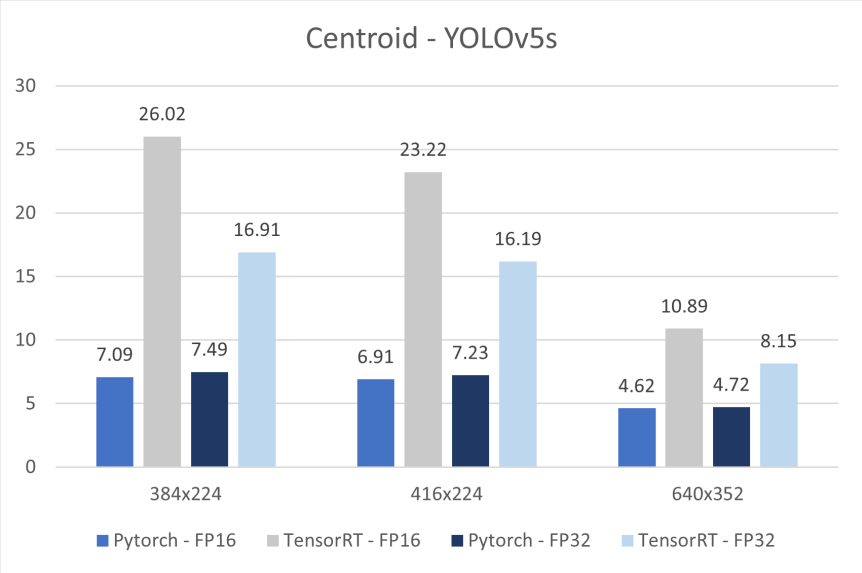

Figura 5.7: Tasa de FPS para Centroid usando YOLO*v5s*.

Se puede observar de ambas figuras que los máximos de cada configuración los entrega TensorRT. En particular, se desprende que el valor más grande de esta sección es 31.92 FPS. Este límite superior se encuentra para la arquitectura YOLOv5n, configurada con *half precision* y un tamaño de imagen de ingreso equivalente a 416×224.

Por otra parte, se extrae de los gráficos que el mejor resultado para la arquitectura YOLO*v5s* es de 26.02 FPS. Este valor se logra utilizando TensorRT, *half precision* y la menor resolución de imagen entre las configuraciones.

Finalmente, se señala que el rango de valores observado en las figuras usando Pytorch es de 4.62 y 8.12 FPS. Específicamente, se puede desprender que el mínimo es alcanzado por YOLO*v5s* y el máximo por YOLO*v5n*.

5.2.1.2. Sort

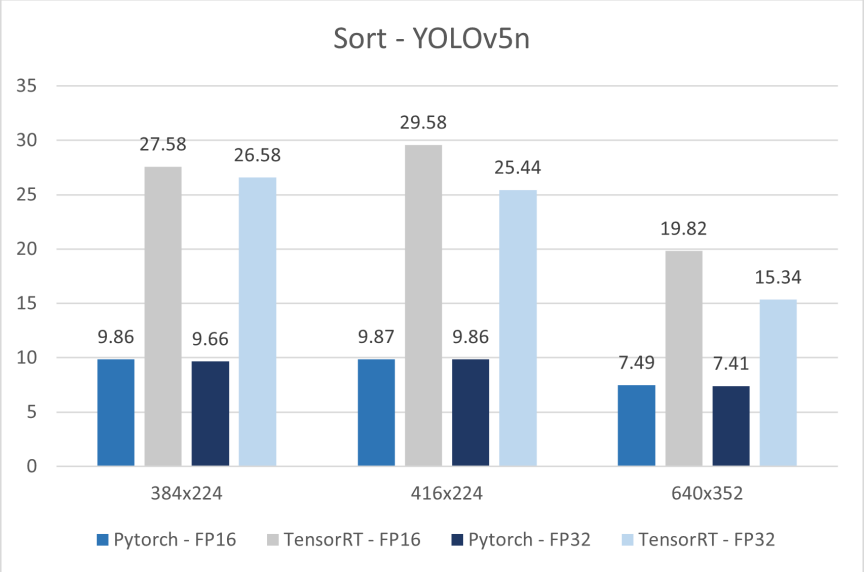

Figura 5.8: Tasa de FPS para Sort usando YOLO*v5n*.

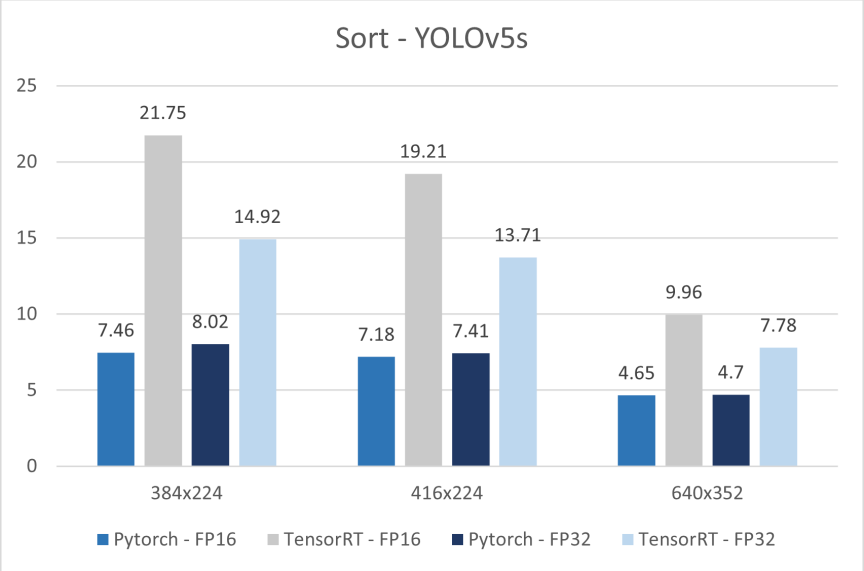

Figura 5.9: Tasa de FPS para Sort usando YOLO*v5s*.

Se observa en los gráficos que los resultados son similares a los anteriormente explorados. Pudiendo diferenciar en las figuras una clara diferencia de escala para lo entregado por TensorRT y Pytorch. Asimismo, se destaca que el máximo encontrado toma un valor de 29.58 FPS. Este extremo se alcanza con YOLO*v5n*, FP16, TensorRT y un tamaño de imagen igual a 416×224.

Adicionalmente, se puede recalcar de la figura 5.9 que el mayor *frame rate* encontrado para YOLOv5s es de 21.75. Esto se obtiene realizando la inferencia en TensorRT, en conjunto con el uso de *half precision* y una resolución equivalente a 384×224.

Por último, se observa en las figuras que los máximos valores de YOLO*v5n* y YOLO*v5s* para Pytorch son de 9.87 y 8.02, respectivamente. Para YOLO*v5n*, esto se logra con FP16 y una resolución de 416×224. En contraste, YOLO*v5s* utiliza FP32, en conjunto con un tamaño de imagen igual a 384×224.

5.2.1.3. Deepsort

Luego de realizar las pruebas, se observa que el uso de GPU para el despliegue del *tracker* no es posible. Es relevante mencionar que el factor que imposibilita el uso de la unidad de procesamiento gráfico, es la memoria RAM consumida.

Frente a esto, no es posible utilizar *half precision* ni la plataforma TensorRT. Luego, las pruebas se realizan únicamente para las configuraciones posibles, destinando exclusivamente el procesamiento requerido a la CPU.

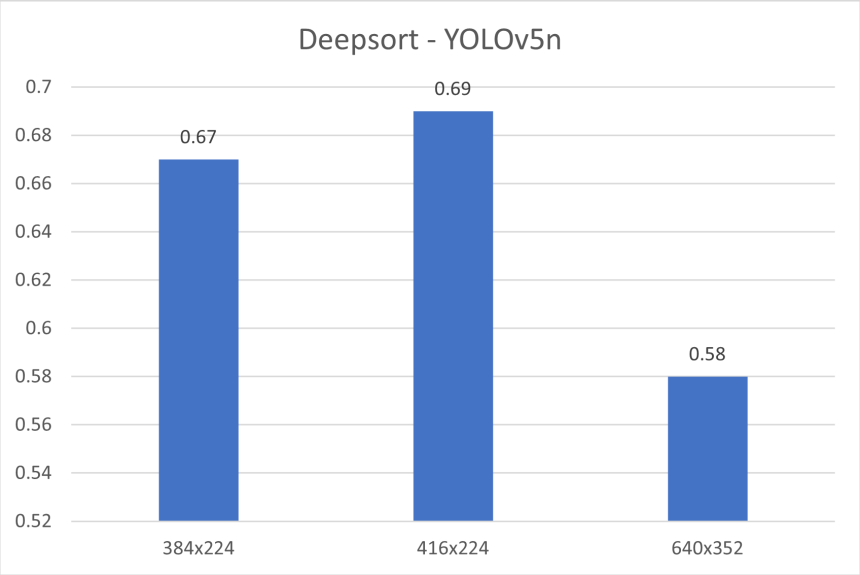

Figura 5.10: Tasa de FPS para Deepsort usando YOLO*v5n*.

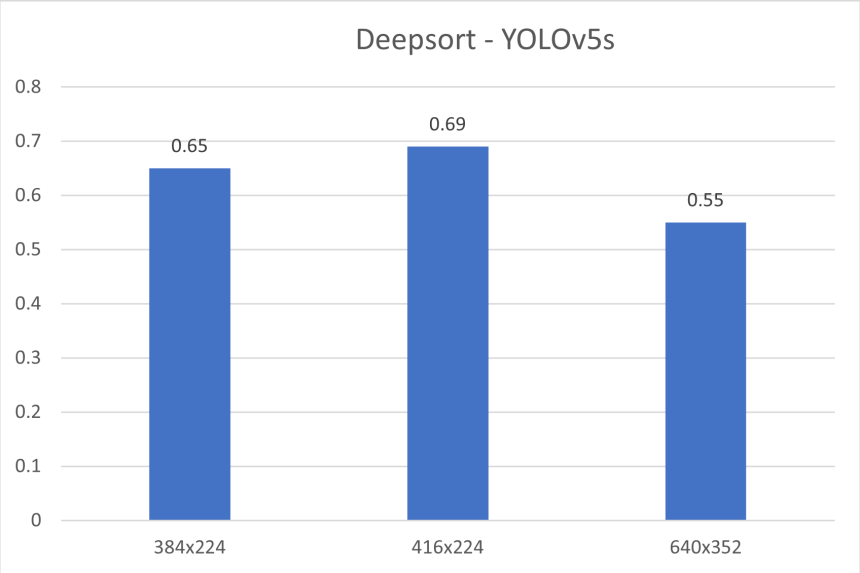

Figura 5.11: Tasa de FPS para Deepsort usando YOLO*v5s*.

Se destaca de ambas figuras que ninguna configuración alcanza un *frame rate* de 1 FPS. Adicionalmente, se desprende de los gráficos que la resolución que entrega mejores resultados es igual a 416×224

5.2.2. Selección de configuraciones

Frente a los resultados de velocidad de inferencia entregados por Deepsort, se determina que no es factible su uso. Por lo tanto, se descarta el modelo para lo que resta de experimentación.

La siguiente sección se aborda usando los resultados y el método mencionado anteriormente. Con esto, se acota el conjunto de combinaciones para las cuales se realizan la pruebas. En particular, se parte por descartar el uso de Pytorch, ya que se puede observar en las figuras de la sección de velocidad de *tracking*que ninguna configuración supera los 20 FPS.

Posteriormente, se continua el proceso para determinar los valores para la configuración conformada por Centroid y YOLOv5n. Primero, se parte por analizar los resultados de velocidad, donde se distingue que la única combinación de parámetros que no supera la cota superior propuesta es la compuesta por FP32 y resolución de 640×352. Luego, al analizar los valores de *mean average precision*, se escogen las variables: FP16 y 640×352.

Asimismo, se realiza un proceso análogo para los pares “Centroid - YOLOv5s” y “Sort - YOLOv5n”, llegando a una configuración específica para cada uno. Por último, para la configuración compuesta por por Sort y YOLOv5s , se determinan sus valores de manera directa. Esto se debe a que solo hay un par que consigue valores alcancen los 20 FPS. Finalmente, se determina que las configuraciones a utilizar se conforman de la siguiente manera:

Tabla 5.1: Configuraciones seleccionadas.

Arquitectura	Tracker	FPP	Resolución
YOLOv5n	Centroid	FP16	640 × 352
	Sort	FP32	416 × 224
YOLOv5s	Centroid	FP16	416 × 224
	Sort	FP16	384 × 224

5.2.3. Precisión de resultados

En la siguiente sección, se muestran los resultados relacionados a la exactitud que alcanzan las predicciones de los distintos modelos. Para obtener los resultados de la evaluación, se utiliza la librería *py-motmetrics*[33].

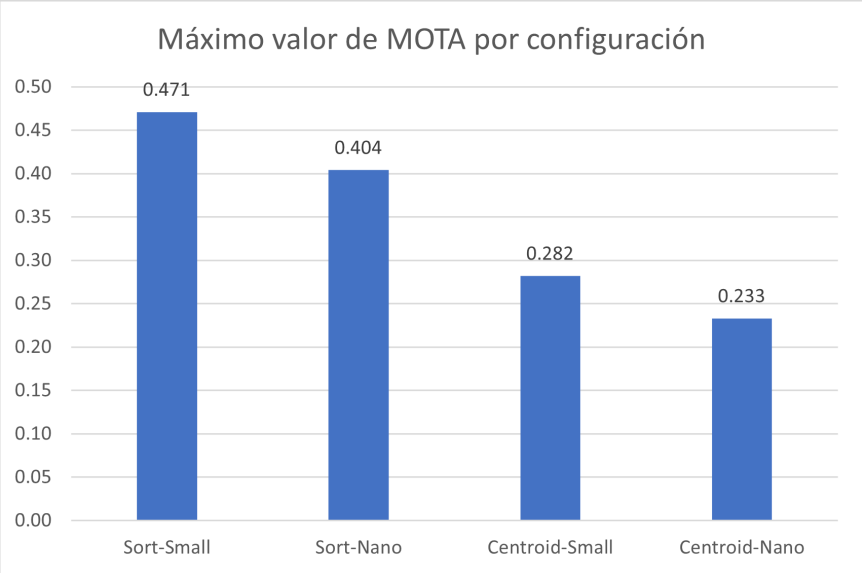

Figura 5.12: Mejor valor de MOTA para cada modelo implementado.

Se desprende de las figuras que el valor máximo obtenido en las pruebas se logra usando Sort como *tracker* y YOLO*v5s* como arquitectura de detección. También, se destaca que el mayor valor de MOTA logrado es de 0.471.

Asimismo, se observa en las figuras que el peor valor es equivalente a 0.233 %, este se logra utilizando Centroid en conjunto con YOLO*v5s*.

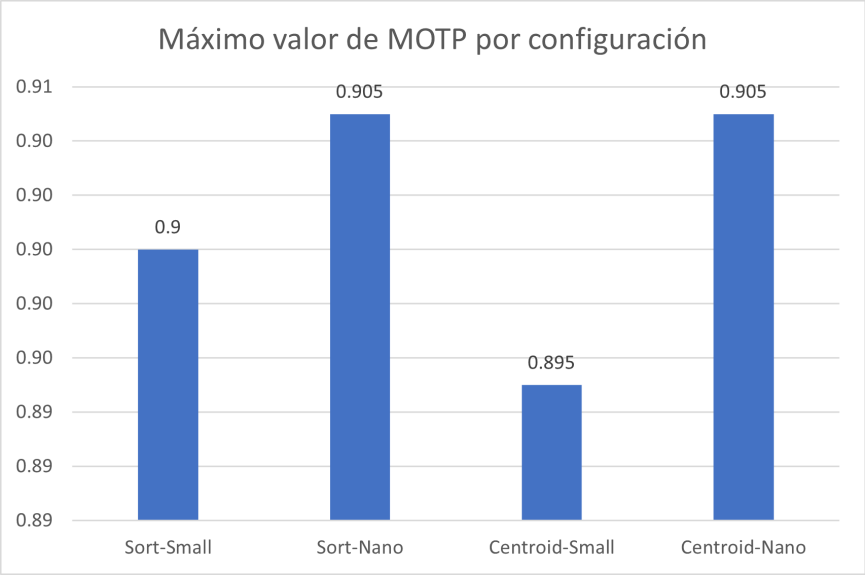

Figura 5.13: Mejor valor de MOTP para cada modelo implementado.

Por otro lado, se observa en el gráfico 5.13 que la diferencia entre los mejores resultados de MOTP no supera a 0.01. Igualmente, al examinar la figura se aprecia que YOLO*v5n* entrega los valores de mayor valor para cada *tracker*.

5.2.4. Relación entre exactitud y velocidad

A continuación, se presenta el comportamiento de *multiple object tracking accuracy* en relación a la velocidad de inferencia. Específicamente, se observa el comportamiento para las 4 configuraciones estudiadas el comportamiento de MOTA en función de las tasas. Junto a esto, se muestra el valor máximo de MOTA para cada algoritmo, desglosándolo según la tasa en que se obtuvo.

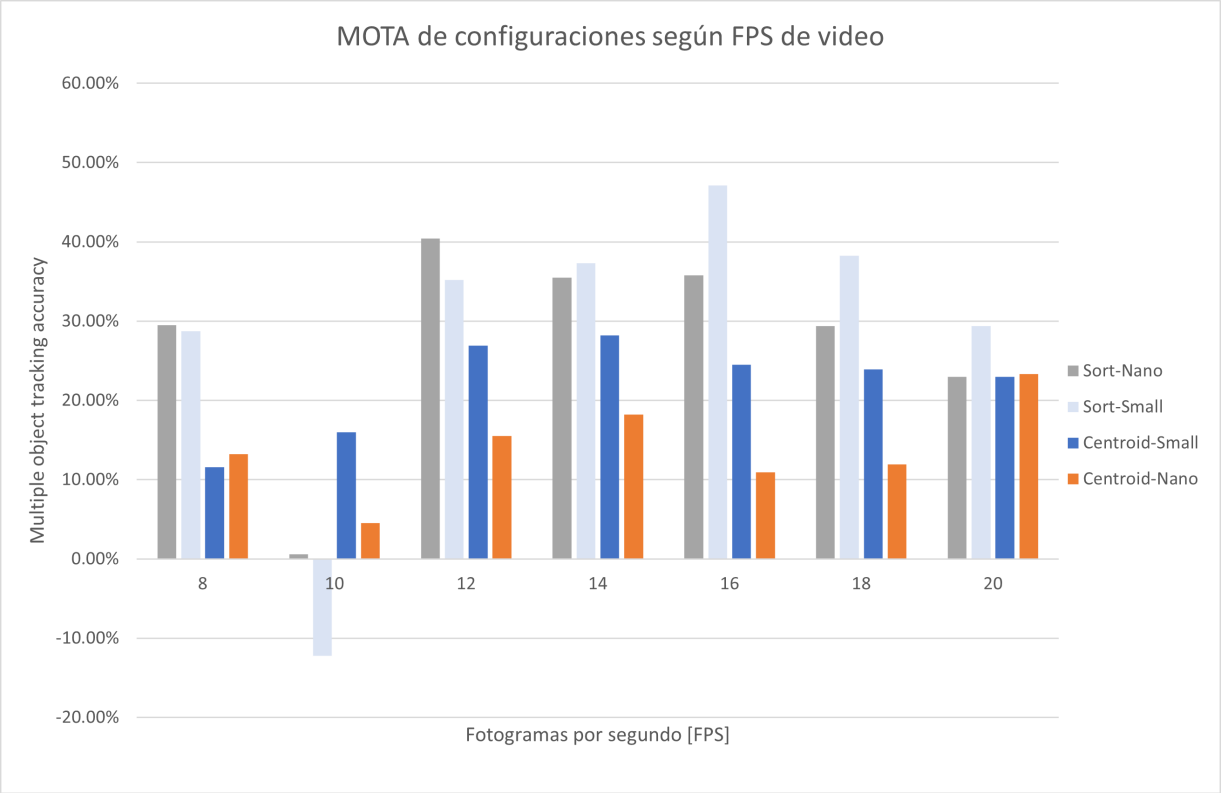

Figura 5.14: Valores de MOTA para las configuraciones seleccionadas.

Se observa en la figura que los peores resultados para todos los *trackers* ocurren al utilizar una tasa de 10 FPS. Sumado a esto, se destaca del gráfico que la configuración compuesta por Sort y YOLO*v5s* alcanza un valor negativo para esta tasa.

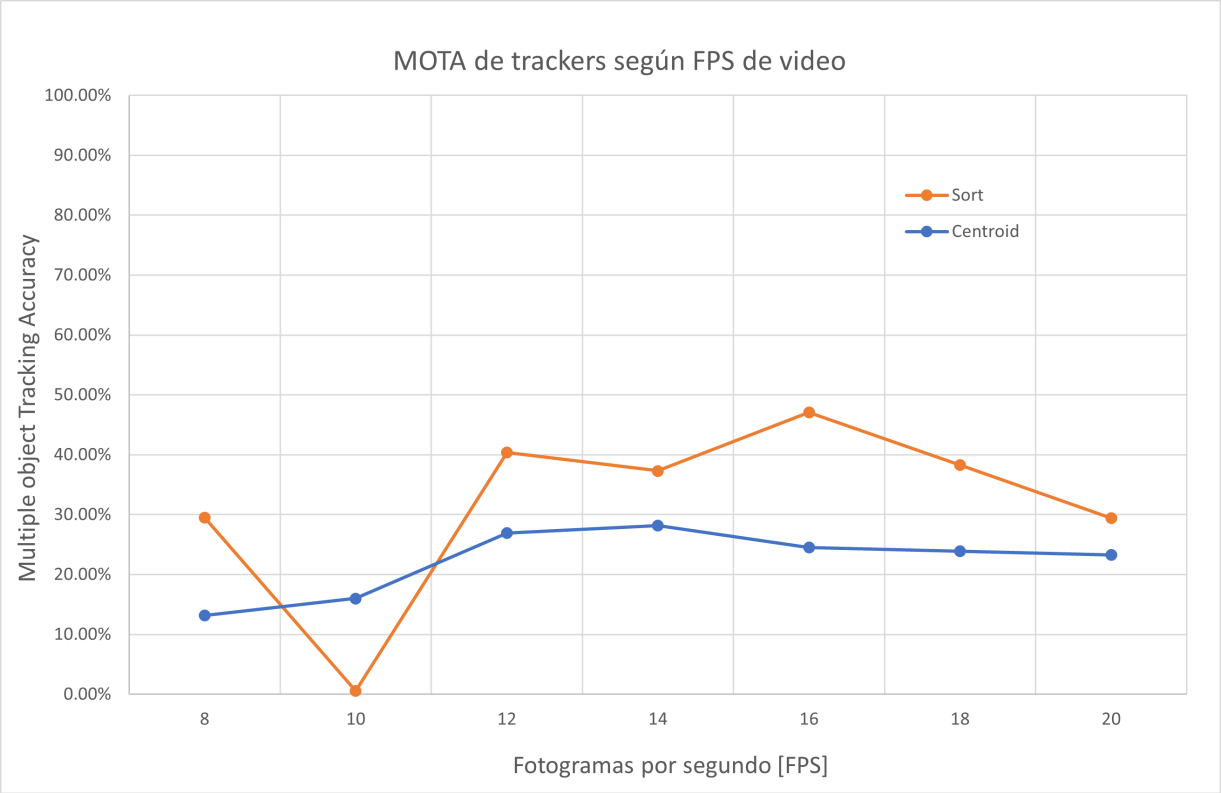

Figura 5.15: Comportamiento del mejor MOTA obtenido por cada algoritmo según la tasa de FPS.

Al analizar la figura, se puede extraer que Sort provee mejores resultados de MOTA para todos los *frame rates* probados, exceptuando 10 FPS. Al mismo tiempo, se aprecia que el mejor resultado que entrega este ocurre al utilizar una tasa de 16 FPS.

De igual manera, se observa poca variabilidad al utilizar tasas entre 12 y 20 FPS al usar Centroid. No obstante, se puede apreciar una caída en la magnitud de los valores al usar tasas menores. Por otro lado, se desprende del gráfico que el algoritmo entrega su máximo resultado cuando se usa un *frame rate* equivalente a 14.

5.3. Módulo 3: Extractor de información

5.3.1. Conteo de vehículos

El conteo de vehículos depende directamente de los resultados entregados por el módulo de *tracking*. En específico, la métrica más relevante para su funcionamiento es el *multiple object tracking accuracy*. Por consecuencia, el porcentaje de error de este se encuentra ligado al MOTA obtenido.

5.3.2. Dirección de vehículos

En general, se puede observar un alto porcentaje de acierto en las anotaciones de la dirección de viaje de los vehículos. Siendo un caso borde el uso de la vía secundaria que se encuentra en el camino observado. Adicionalmente, se destaca que se observa cierta mejoría

al aplicar un número de fotogramas de validación antes de confirmar una dirección.

5.3.3. Peatón en zona de peligro

Dada la escasez de casos para esta evaluación, se dificulta validar el funcionamiento de esta alerta. Frente a esto, se extiende el polígono que representa la calle para cubrir una zona de la vereda en específico y se observan los acontecimientos.

5.3.4. Congestión vehicular

Al igual que el conteo de vehículos, la congestión vehicular causada porque el número de autos exceda el umbral seleccionado depende directamente del módulo de *tracking*. Debido a esto, con analizar los valores de MOTA es suficiente.

Por otro lado, la congestión causada por que la velocidad de un vehículo presente sea baja, no se valida mediante una métrica. Por lo cual, se decide variar el umbral utilizado y observar los resultados obtenidos.

Capítulo 6

Análisis y discusión de resultados

6.1. Módulo 1: Detector de objetos

6.1.1. Velocidad de inferencia

Relativo a la velocidad de inferencia del detector, se puede extraer de las figuras 5.1 y 5.2 que los factores que más influyen en el rendimiento son motor de inferencia y la resolución de la imagen usada. Luego, al analizar los datos de la tabla B.1, se desprende que TensorRT cumple con su cometido, dado que la tasa de FPS que se observa para todas las configuraciones es de mayor magnitud que las entregadas por Pytorch.

En adición a esto, se resalta que el impacto causado por la resolución de la imagen, posiblemente se sustenta en que el volumen de datos que se procesa varía en relación al tamaño de la imagen. En otras palabras, se requieren de tiempos proporcionales a la dimensión de los datos ingresados.

Por otro lado, se destaca que el uso de *half precision* para ambos motores entrega mejoras en velocidad, bajando su incidencia a medida que el tamaño de imagen utilizado es menor. Esto posiblemente es causado porque la cantidad de bits ahorrados en el cambio de precisión se hace menor a medida que la imagen es de menor tamaño.

6.1.2. Rendimiento del modelo

En relación a los valores experimentales de precisión, se advierte que el mayor impacto lo genera la arquitectura del modelo de detección. Al mismo tiempo, se señala como segundo factor de mayor incidencia, la resolución de la imagen que se ingresa. Igualmente, se resalta de los resultados que el motor de inferencia y el *floating point precision* no afectan en mayor medida.

Es importante señalar que el comportamiento de la precisión y sus factores de incidencia siguen lo esperado. Ya que, las distintas arquitecturas de YOLO*v5* tienen como objetivo original priorizar o sacrificar velocidad de inferencia usando como intercambio la calidad de sus predicciones. En segundo lugar, la disminución de la resolución que se ingresa a una red neuronal lleva consigo un coste en la información que provee, por lo que es esperable que detecciones generadas sobre imágenes más pequeñas tiendan a ser menos exactas.

6.1.3. Relación entre exactitud y velocidad

De manera general, se puede desprender de los resultados de la experimentación la existencia de un *trade-off* entre precisión y velocidad de inferencia.

Esto responde a lo esperado según la teoría y se puede evidenciar en la figura 5.5. En esta se aprecia una caída en los valores de mAP a medida que el *frame rate* logrado aumenta.

Adicionalmente, los valores extremos de ambos motores de inferencia son los esperados, exceptuando al valor que consigue la máxima velocidad para Pytorch. Este valor anómalo difiere del esperado por la precisión usada, ya que por teoría se esperaba que el uso de *half precision* fuera parte de esta configuración. Según lo mencionado en la sección anterior, en adición a la baja diferencia de esta configuración con la que si usa FP16, se desestima la existencia de un error.

6.2. Módulo 2: Tracker

6.2.1. Velocidad de inferencia

Referente a los resultados de velocidad de *tracking* obtenidos, se extraen distintas conclusiones. Primero, al analizar los datos presentados en la tabla B.3, se desprende que el rendimiento de los módulos de *tracking* sigue lo sugerido por la teoría. En concreto, como Centroid es un *tracker* relativamente simple y directo, es de esperar que entregue una mayor tasa de FPS. En contraste, como Sort es un algoritmo más moderno y posee una mayor complejidad, es lógico que entregue menores valores para los *frame rates* obtenidos.

Es importante notar que los resultados tienden a responder de acuerdo al *benchmark* de detección. Este comportamiento se considera esperable, dado que el tiempo de medición estudiado abarca el tiempo de inferencia del detector. En otras palabras, se obtienen menores tiempos de inferencia al aplicar *tracking* usando TensorRT para el despliegue y YOLO*v5n* como arquitectura de detección.

Sin embargo, al examinar las figuras 5.6, 5.7, 5.8 y 5.9, se advierten valores de FPS imprevistos. Específicamente, se espera por sustento teórico que las configuraciones en las que se use *half precision* presenten una mejoría en la velocidad de inferencia.

No obstante, en la tabla B.3 no se atisba una diferencia significativa en los valores obtenidos con Pytorch. Posiblemente, esto se debe a que la mejoría en la tasa de FPS se ve opacada por el tiempo invertido en cambiar la precisión de las imágenes que se ingresan. Es importante aclarar que la lectura de imágenes se hace con la librería OpenCV, la cual utiliza precisión FP32 como predeterminada.

Finalmente, se descarta el uso de Deepsort para este *hardware*. Esto debido a los bajos FPS logrados y la incapacidad de utilizar la GPU en la Jetson Nano 2GB. Una de las causas más probables detrás de este suceso, es la escasez de memoria RAM. Esto posiblemente

causado por la necesidad de cargar dos redes neuronales para el uso de este *tracker*.

6.2.2. Precisión de resultados

Con respecto a la validación de las predicciones, se distingue de la figura 5.12 que el *tracker* y arquitectura que entregan mejores resultados de MOTA, son Sort y YOLOv5s, respectivamente.

Por otro lado, se aprecia del gráfico de la figura 5.13 que no existe mayor diferencia entre las configuraciones para los valores de MOTP obtenidos. Luego, vale la pena mencionar que a causa de la definición de MOTP y del comportamiento de los valores, es difícil sacar observaciones útiles de los modelos.

Dicho de otra manera, debido a que la calidad de los *bounding boxes* se calcula sobre las predicciones acertadas, no se pueden extraer conclusiones claras con esta métrica. Un ejemplo de esto es que la configuración que alcanza el valor más alto de MOTA no alcanza el valor más alto de MOTP.

6.2.3. Relación entre exactitud y velocidad

Por lo que se refiere a la interacción entre calidad de resultados y tasa utilizada, se observa de la figura 5.14 que los mejores resultados se centran alrededor de 12 y 18 FPS. Adicionalmente, es relevante recalcar que no se presenta una proporcionalidad clara entre la calidad de los resultados y el *frame rate* usado para ninguna de las configuraciones.

La falta de una tendencia clara en los resultados puede ser ocasionada por múltiples cosas. Sin embargo, la presencia u ausencia de casos bordes se considera como un punto de contención de gran impacto para la evaluación. Esto debido a que la extensión del conjunto de prueba es baja y que casos borde pueden generar efectos en cadena que afecten los valores de MOTA obtenidos.

Existen distintos casos bordes, uno de los más esperables es que un objeto de interés aparezca en un ángulo extraño, lo que cause que el primer módulo no sea capaz de llevar a cabo de manera correcta su detección. Otro posible caso, es que en un fotograma un objeto obstruya la visión de otro, dificultando el funcionamiento del primer módulo y por consiguiente el del *tracker*. Es importante aclarar que frente a un ajuste de tasa de FPS, un caso borde presente en un fotograma puede o no aparecer en la secuencia de *frames* final.

Por otra parte, se muestra en la figura 5.15 que Centroid es menos susceptible a la variación de *frame rate*. Probablemente, esto se puede justificar con la forma en que funciona Sort y los elementos que lo componen. En concreto, Sort espera un número de detecciones en específico antes de empezar el seguimiento de un objeto. Además, este algoritmo provee un umbral de tolerancia, el cual indica el número de fotogramas sin detección que espera antes de borrar un objeto seguido.

En particular, la aplicación de los valores mencionados anteriormente, *min hits* y *max age*,

respectivamente, puede verse afectada por la tasa de FPS del video entrante. Esto debido a que su utilidad está directamente ligada a los pixeles que recorre un objeto específico entre *frames*. Concretamente, frente a una baja tasa de FPS el desplazamiento entre pixeles de un objeto es mayor. Por consiguiente, esperar a un número de detecciones antes de registrar un objeto u aguardar una cantidad de veces para eliminar un objeto que no aparece puede no ser posible.

6.3. Módulo 3: Extractor de información

Al realizar una inspección de los resultados obtenidos por el extractor, se puede concluir que el conteo de vehículos se mantiene igual al número de objetos seguidos. Esto indica que su implementación es correcta.

Adicionalmente, al analizar la detección de un peatón en peligro se observan errores para casos borde. Específicamente, las faltas se presentan en forma de falsos positivos. Concretamente, esto sucede en casos en que un peatón esta al extremo de la calle observada.

Con respecto al registro de dirección en que viajan los vehículos, se aprecia que los resultados tienden a ser correctos. Sin embargo, existe un margen de error para casos en que el vehículo observado ingresa a la calle principal a través de la calle perpendicular.

Por último, no es posible determinar con exactitud el grado de éxito de la captación de congestión vehicular. Pese a esto, se confirma en cierta medida que la implementación se hace de manera correcta. Esto debido a que el sistema responde de manera apropiada al modificar el umbral máximo de vehículos en calle.

Es importante mencionar que las observaciones del comportamiento de este módulo se hacen a través de una inspección de funcionamiento. Por consecuencia, las conclusiones a las que se puede llegar son limitadas.

6.4. Elección de parámetros finales

Con el fin de llegar a una configuración final, se analizan los resultados obtenidos a través de la experimentación. En adición a esto, se inspeccionan los resultados a través de una interpretación gráfica. En otras palabras, se dibujan los datos de las anotaciones obtenidas sobre el video de inferencia.

En particular, para el análisis cuantitativo se considero MOTA como métrica determinante para elegir los valores finales. No obstante, también se observan las otras variables para descartar la existencia de un comportamiento irregular.

Tabla 6.1: Parámetros finales de configuración.

Parámetro	**Valor seleccionado**
Arquitectura de Red	YOLOv5s
Tracker	Sort
Resolución	384×224
FPP	FP16
Motor de Inferencia	TensorRT

Es relevante mencionar que el mejor resultado de MOTA se logra utilizando una tasa de FPS igual a 16. Sin embargo, el hecho de que con esta configuración se obtienen los mejores resultados para los 4 valores más altos de *frame rate*, sustenta que la elección es robusta para otras tasas de FPS.

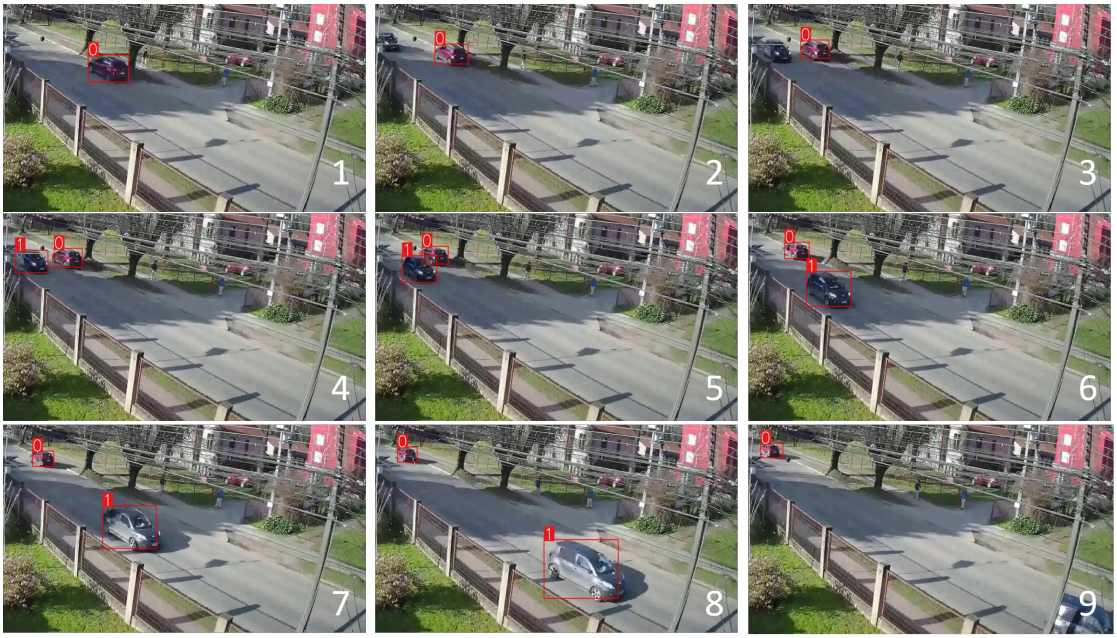

Figura 6.1: Resultados de tracking sobre video.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

Analizando el trabajo realizado, se desprenden distintas conclusiones relacionadas al objetivo general y los objetivos específicos propuestos para lograrlo. De manera general, se observa que los resultados del proyecto si permiten un primer acercamiento a una implementación comercial de un proyecto de *tracking* en un contexto de tránsito vehicular basado en arquitectura de *edge computing*. De manera más específica, se observa de los objetivos específicos planteados al inicio del trabajo se cumplen con distintos grados de éxito.

Por un lado, la procuración de datos propuesta objetivo 1 logra formar una base de tamaño y variedad suficiente para el entrenamiento de las redes seleccionadas. Asegurando un total de 78883 imágenes y 360000 anotaciones de objetos de interés.

En relación a la implementación de *trackers* propuesta, se concluye el trabajo habiendo alcanzando parcialmente el objetivo. En específico, se logra desplegar con éxito dos de los tres modelos propuestos. Siendo posible utilizar la GPU de la Jetson Nano para Centroid y Sort con ambas arquitecturas de detección. En contraste, el uso de la tarjeta gráfica no fue posible para ninguna arquitectura de YOLO*v5* al usar Deepsort como modelo de *tracking*. Frente a este evento, se pueden apreciar las limitaciones del hardware usado y la dificultad de implementar soluciones que involucren más de una red neuronal.

Por otro lado, al analizar el trabajo realizado se desprende que la generación de *benchmark* propuesta en el objetivo 3 es exitosa. En particular, se logra generar métricas de velocidad y calidad de resultados al realizar un despliegue en Jetson Nano 2GB de las dos arquitecturas de YOLO*v5* propuestas. Los resultados obtenidos abarcan múltiples configuraciones, compuestas por diversas variables experimentales relevantes para el trabajo. Como resultado del trabajo se obtiene un *benchmark* que estudia: dos plataformas de despliegue, tres resoluciones de imagen distintas y dos posibles precisiones de punto flotante, logrando resultados para un total de 24 configuraciones posibles.

Adicionalmente, se puede concluir que se logra una conformación satisfactoria la sección de velocidad de inferencia del *benchmark* de modelos de *tracking*. Habiendo generado resultados para los dos modelos de *tracking* funcionales, en combinación con las dos arquitecturas de detección y 24 configuraciones estudiadas en el objetivo anterior. Específicamente, se logra

generar métricas de velocidad 48 configuraciones posibles. El espectro de variables estudiado en la sección de calidad de resultados del *benchmark* es de igual magnitud, sin embargo, el valor que se puede extraer de este es menor. Esto debido a la compleja naturaleza de evaluar un MOT, junto con su baja generalidad para otros contextos.

En cuanto a determinar los parámetros de configuración más apropiados, se consigue acotar entre las opciones un subconjunto viable. Partiendo por descartar de manera definitiva a Deepsort como modelo de *tracking* y concluir que TensorRT es la plataforma más conveniente para un despliegue. Por otro lado, el resto de los parámetros no se puede descontar ni priorizar de manera concluyente. Sin embargo, se puede extraer sobre algunas configuraciones cierta inclinación a ser descartadas o preferidas. Por ejemplo, el uso de una resolución de 640×352 al utilizar YOLO*v5s* queda mayoritariamente descartado, esto a causa de su velocidad de inferencia. Otro caso se presenta al poner como umbral mínimo de funcionamiento del sistema 15 FPS, lo que descartaria el uso de precisión $FP32$ al usar YOLO*v5s*.

Por último, determinar con que grado se cumple el objetivo 6 es complejo. Por un lado, haciendo una inspección de su funcionamiento, se desprende que la implementación logra entregar de manera coherente la información objetivo. Sin embargo, no se puede realizar una evaluación con métricas precisas dada la ausencia de un conjunto de validación.

Finalmente, es importante recalcar que desde el enfoque experimental del trabajo realizado, el objetivo principal si se cumple. Habiendo extraído aprendizajes relevantes para futuras implementaciones similares. Siendo los valores de FPS obtenidos una herramienta útil para otros proyectos, dado que entregan una idea de las limitaciones de hardware. En contraste, si bien las métricas relacionadas a la calidad de resultados pueden ser útiles, estas son menos extrapolables para otros contexto. La razón de esto es su carencia de generalidad, causada por aspectos como el entrenamiento realizado, contexto del proyecto y parámetros aplicados.

7.2. Trabajo futuro

Como se mencionó antes, un punto importante a mejorar es el entrenamiento de las redes neuronales empleadas. A pesar de que los resultados obtenidos sirven para un contexto experimental, es necesario un mayor entrenamiento si se quiere efectuar un despliegue real. En relación a esto, procurar un mayor número de imágenes intentando disminuir la desigualdad de clases podría ser beneficioso para el proyecto.

Al mismo tiempo, un potencial problema con la validación de los *trackers* es la carencia de generalidad. En otras palabras, debido a las dificultades señaladas previamente, es posible que exista un sesgo en los valores logrados. Esto se debe al menudo tamaño del conjunto de evaluación y prueba empleados. Para solucionar esto, es necesario extender los *datasets* usados en la validación. Adicionalmente, anotaciones que retraten el funcionamiento de la calle a múltiples horas y estaciones entregaría robustez a la evaluación generada.

Sumado a esto, en la validación realizada falta analizar posibles situaciones que pudiesen potencialmente afectar el funcionamiento del sistema. Un posible evento es la aparición de clases no pertenecientes a las clases utilizadas en el entrenamiento, por ejemplo: animales,

maquinaria y carretas. De igual manera, el conjunto de validación usado en el análisis carece de distintas situaciones climáticas y condiciones de iluminación.

Por otro lado, una alternativa a considerar para mejorar el trabajo es expandir el conjunto de variables usadas en la experimentación. Pese a que el número de alternativas se determino en conjunto a Sinantica, extender la cantidad de combinaciones podría ayudar a encontrar una combinación que entregue mejores resultados.

En adición a esto, falta desarrollar aspectos relacionados al sistema en el lugar, los cuales del alcance del trabajo. Para empezar, resta examinar el comportamiento de la energía consumida por el Jetson Nano 2GB para distintas situaciones. Asimismo, falta por probar una solución que involucre la movilidad que provee la cámara.

Finalmente, es necesario estudiar cómo se desenvuelve la solución diseñada al ser desplegada en el lugar estudiado. Por un lado, el funcionamiento mismo del sistema y los parámetros seleccionados. También resta ajustar el sistema con la interfaz de recopilación de datos que ya se encuentra en funcionamiento.

Bibliografía

[1] T. Ojala, M. Pietikainen, and D. Harwood, "Performance evaluation of texture measures with classification based on kullback discrimination of distributions," in *Proceedings of 12th International Conference on Pattern Recognition*, vol. 1, pp. 582–585, 1994.

[2] D. Lowe, "Object recognition from local scale-invariant features," *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999.

[3] R. K. McConnell, "Method of and apparatus for pattern recognition," 1 1986.

[4] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, 2014.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[8] Ultralytics, "Ultralytics/yolov5: Yolov5 in pytorch & gt; onnx & gt; coreml & gt; tflite."

[9] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, p. 35–45, 1960.

[10] J. Nascimento, A. Abrantes, and J. Marques, "An algorithm for centroid-based tracking of moving objects," in *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, vol. 6, pp. 3305–3308 vol.6, 1999.

[11] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3464–3468, 2016.

[12] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, p. 83–97, 1955.

[13] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 3645–3649, IEEE, 2017.

[14] K. Zhou, Y. Yang, A. Cavallaro, and T. Xiang, "Omni-scale feature learning for person

re-identification," 2019.

[15] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler, "Mot16: A benchmark for multi-object tracking," 2016.

[16] M. P. D. Cahyo and F. Utaminingrum, "Autonomous robot system based on room nameplate recognition using yolov4 method on jetson nano 2gb," *JOIV : International Journal on Informatics Visualization*, 2022.

[17] Z.-D. Zhang, M.-L. Tan, Z.-C. Lan, H.-C. Liu, L. Pei, and W.-X. Yu, "CDNet: A real-time and robust crosswalk detection network on Jetson Nano based on Yolov5," *Neural Computing and Applications*, vol. 34, no. 13, p. 10719–10730, 2022.

[18] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2017.

[19] S. Vishwanathan and M. Narasimha Murty, "Ssvm: a simple svm algorithm," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, vol. 3, pp. 2393–2398 vol.3, 2002.

[20] A. A. Kareem, D. A. Hammood, A. A. Alchalaby, and R. A. Khamees, "A performance of low-cost nvidia jetson nano embedded system in the real-time siamese single object tracking: A comparison study," *Communications in Computer and Information Science*, p. 296–310, 2022.

[21] B. Yan, H. Peng, K. Wu, D. Wang, J. Fu, and H. Lu, "Lighttrack: Finding lightweight neural networks for object tracking via one-shot architecture search," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[22] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[23] D. Guo, J. Wang, Y. Cui, Z. Wang, and S. Chen, "Siamcar: Siamese fully convolutional classification and regression for visual tracking," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6268–6276, 2020.

[24] A. Mohan, A. S. Kaseb, K. W. Gauen, Y.-H. Lu, A. R. Reibman, and T. J. Hacker, "Determining the necessary frame rate of video data for object tracking under accuracy constraints," *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, 2018.

[25] B. Xiao and S.-C. Kang, "Development of an image data set of construction machines for deep learning object detection," *Journal of Computing in Civil Engineering*, vol. 35, no. 2, 2021.

[26] K. K, E. Venugopal, and P. G, "Indian vehicle dataset," 2022.

[27] L. Regenwetter, B. Curry, and F. Ahmed, "Biked: A dataset for computational bicycle design with machine learning benchmarks," 2021.

[28] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," 2010.

[29] A. Sharma, "Mean average precision (map) using the coco evaluator," Jul 2022.

[30] J. Luiten, A. Osep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taixé, and B. Leibe, "Hota: A higher order metric for evaluating multi-object tracking," *International Journal of Computer Vision*, vol. 129, pp. 1–31, 02 2021.

[31] S. D. Roth, "Ray casting for modeling solids," *Computer Graphics and Image Processing*, vol. 18, no. 2, pp. 109–144, 1982.

[32] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva, "A comparative analysis of object detection metrics with a companion open-source toolkit," *Electronics*, vol. 10, no. 3, 2021.

[33] C. Heindl and J. Valmadre, "Benchmark multiple object trackers (mot) in python," 2017.

Anexos

Anexo A. Conceptos útiles

A.1. Conceptos Base

- **Siniestro de tránsito**: Suceso que ocurre cuando un individuo queda expuesto a múltiples riesgos físicos y/o psicológicos relacionados involucrados a un vehículo.
- **Fotogramas por segundo o FPS**: Número de imágenes que un algoritmo en específico es capaz de procesar en un segundo.
- ***Random access memory* o RAM**: Memoria de corto plazo de los ordenadores utilizada para almacenar datos y procesos de ejecución de manera temporal.
- ***Internet of things* o IOT**: Proceso que permite conectar elementos físicos cotidianos al Internet
- ***Intelligent traffic managment systems* o ITMS**: Conjunto de soluciones tecnológicas de las telecomunicaciones y la informática diseñadas para mejorar la operación y seguridad del transporte terrestre.
- **Vehículo aéreo no tripulado o UAV**: Aeronave que vuela sin tripulación, la cual ejerce su función remotamente.
- **Overfitting**: Efecto negativo causado por sobreentrenar un algoritmo de inteligencia artificial.

A.2. Distancias

- **Euclidiana**:

$$DIST(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{A.1}$$

- **Mahalanobis**:

$$DIST(X,Y) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})} \tag{A.2}$$

A.3. Métricas de evaluación

Medir el rendimiento de un detector de objetos se reduce en determinar si una detección es correcta o no. Para esto se definen los siguientes conceptos:

- **Verdad basal o Ground Truth (GT)**: Objetivo correcto de la detección utilizado para contrastar con las detecciones generadas por el modelo.
- **Verdadero positivo o True Positive (TP)**: Detección correcta realizada por el modelo.
- **Falso positivo o False Positive (FP)**: Detección incorrecta realizada por el modelo.
- **Falso negativo o False Negative (FN)**: Verdad basal no detectada por el modelo.
- **Verdadero negativo o True Negative (TN)**: Área de fondo correctamente no detectada por el modelo.

Para determinar que una detección es correcta o no se compara la métrica IOU en relación a un umbral escogido.

- IOU: Grado de superposición entre el ground truth y predicción.

$$IOU = \frac{\text{Area}\,(GT \cap PD)}{\text{Area}\,(GT \cup PD)} \tag{A.3}$$

Anexo B. Resultados experimentales

B.1. Resultados experimentales módulo de detección

Tabla B.1: Tasa de FPS para módulo de detección.

Arquitectura	Plataforma	FPP	Resolucion	FPS
YOLOv5s	Pytorch	FP32	384 × 384	10.26
			416 × 416	8.73
			640 × 640	4.13
		FP16	384 × 384	10.3
			416 × 416	9.2
			640 × 640	4.62
	TensorRT	FP32	384 × 384	14.35
			416 × 416	12.91
			640 × 640	6.29
		FP16	384 × 384	22.99
			416 × 416	19.66
			640 × 640	9.5
YOLOv5n	Pytorch	FP32	384 × 384	23.27
			416 × 416	19.89
			640 × 640	9.71
		FP16	384 × 384	22.65
			416 × 416	21.72
			640 × 640	11.23
	TensorRT	FP32	384 × 384	31.53
			416 × 416	31.25
			640 × 640	15.62
		FP16	384 × 384	50.1
			416 × 416	43.17
			640 × 640	20.86

Tabla B.2: Resultados de mean average precision.

Arquitectura	**Plataforma**	**FPP**	**Resolución**	**mAP0.5**	**mAP0.5-0.95**
YOLOv5s	Pytorch	FP32	384 × 384	0.535	0.378
			416 × 416	0.574	0.419
			640 × 640	0.626	0.405
		FP16	384 × 384	0.535	0.379
			416 × 416	0.573	0.423
			640 × 640	0.604	0.422
	TensorRT	FP32	384 × 384	0.539	0.382
			416 × 416	0.56	0.405
			640 × 640	0.615	0.394
		FP16	384 × 384	0.529	0.373
			416 × 416	0.556	0.406
			640 × 640	0.578	0.396
YOLOv5n	Pytorch	FP32	384 × 384	0.49	0.333
			416 × 416	0.504	0.349
			640 × 640	0.606	0.385
		FP16	384 × 384	0.49	0.334
			416 × 416	0.505	0.355
			640 × 640	0.56	0.378
	TensorRT	FP32	384 × 384	0.504	0.347
			416 × 416	0.513	0.358
			640 × 640	0.608	0.387
		FP16	384 × 384	0.504	0.348
			416 × 416	0.508	0.358
			640 × 640	0.569	0.387

B.2. Resultados experimentales módulo de tracking

Tabla B.3: Velocidad de inferencia Trackers.

Tracker	Red	Plataforma	FPP	Resolucion	FPS
YOLOv5n	Centroid	Pytorch	FP32	640 × 352	7.01
				416 × 224	8.36
				384 × 224	8.15
			FP16	640 × 352	7.06
				416 × 224	8.15
				384 × 224	8.12
		TensorRT	FP32	640 × 352	18.14
				416 × 224	30.68
				384 × 224	31.24
			FP16	640 × 352	22.55
				416 × 224	31.92
				384 × 224	30.18
	Sort	Pytorch	FP32	640 × 352	7.41
				416 × 224	9.86
				384 × 224	9.66
			FP16	640 × 352	7.49
				416 × 224	9.87
				384 × 224	9.86
		TensorRT	FP32	640 × 352	15.34
				416 × 224	25.44
				384 × 224	26.58
			FP16	640 × 352	19.82
				416 × 224	29.58
				384 × 224	27.58
YOLOv5s	Centroid	Pytorch	FP32	640 × 352	4.72
				416 × 224	7.23
				384 × 224	7.49
			FP16	640 × 352	4.62
				416 × 224	6.91
				384 × 224	7.09
		TensorRT	FP32	640 × 352	8.15
				416 × 224	16.19
				384 × 224	16.91
			FP16	640 × 352	10.89
				416 × 224	23.22
				384 × 224	26.02
	Sort	Pytorch	FP32	640 × 352	4.7
				416 × 224	7.41
				384 × 224	8.02
			FP16	640 × 352	4.65
				416 × 224	7.18
				384 × 224	7.46
		TensorRT	FP32	640 × 352	7.78
				416 × 224	13.71
				384 × 224	14.92
			FP16	640 × 352	9.96
				416 × 224	19.21
				384 × 224	21.75

Tabla B.4: Exactitud de resultados de tracking.

Modelo	FPS	IDF1	IDP	IDR	Rcll	Prcn	MOTA	MOTP
Sort-Small	20	55.80 %	61.50 %	51.10 %	56.30 %	67.70 %	29.40 %	0.803
	18	60.75 %	66.25 %	56.10 %	61.40 %	72.55 %	38.25 %	0.828
	16	65.70 %	71.00 %	61.10 %	66.50 %	77.40 %	47.10 %	0.853
	14	65.60 %	67.10 %	64.10 %	66.80 %	70.00 %	37.30 %	0.774
	12	62.70 %	59.90 %	65.80 %	73.10 %	66.50 %	35.20 %	0.849
	10	37.20 %	34.00 %	41.00 %	55.10 %	45.70 %	-12.20 %	0.9
	8	61.00 %	54.60 %	69.00 %	78.30 %	62.00 %	28.70 %	0.858
Sort-Nano	20	61.20 %	61.60 %	60.80 %	60.80 %	61.60 %	23.00 %	0.821
	18	57.90 %	58.40 %	57.45 %	63.85 %	64.95 %	29.40 %	0.8435
	16	54.60 %	55.20 %	54.10 %	66.90 %	68.30 %	35.80 %	0.866
	14	67.70 %	67.70 %	67.70 %	67.70 %	67.70 %	35.50 %	0.819
	12	70.60 %	69.70 %	71.50 %	71.50 %	69.70 %	40.40 %	0.862
	10	41.30 %	40.90 %	41.70 %	51.30 %	50.30 %	0.60 %	0.905
	8	55.90 %	52.00 %	60.50 %	72.90 %	62.70 %	29.50 %	0.859
Centroid-Small	20	59.50 %	63.00 %	56.30 %	56.30 %	63.00 %	23.30 %	0.797
	18	58.55 %	64.60 %	53.65 %	53.65 %	64.60 %	23.90 %	0.818
	16	57.60 %	66.20 %	51.00 %	51.00 %	66.20 %	24.50 %	0.839
	14	59.10 %	68.70 %	51.80 %	51.80 %	68.70 %	28.20 %	0.759
	12	58.20 %	68.10 %	50.80 %	50.80 %	68.10 %	26.90 %	0.831
	10	50.90 %	61.30 %	43.60 %	43.60 %	61.30 %	16.00 %	0.895
	8	51.70 %	57.00 %	47.30 %	47.30 %	57.00 %	11.60 %	0.822
Centroid-Nano	20	54.60 %	56.60 %	52.80 %	53.40 %	57.30 %	12.90 %	0.784
	18	54.00 %	56.30 %	51.90 %	52.40 %	56.85 %	11.90 %	0.808
	16	53.40 %	56.00 %	51.00 %	51.40 %	56.40 %	10.90 %	0.832
	14	57.30 %	59.90 %	55.00 %	55.50 %	60.40 %	18.20 %	0.777
	12	54.20 %	56.80 %	51.80 %	53.90 %	59.10 %	15.50 %	0.842
	10	49.70 %	52.90 %	46.80 %	46.80 %	52.90 %	4.50 %	0.905
	8	52.10 %	58.10 %	47.30 %	47.30 %	58.10 %	13.20 %	0.822

Anexo C. Especificaciones de Hardware

C.1. Jetson Nano 2GB

Tabla C.1: Especificaciones técnicas Jetson Nano 2GB.

Componente	**Especificación**
GPU	128-core NVIDIA Maxwel
CPU	Quad-core ARM® A57 @ 1.43 GHz
Memory	2 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (Card not included)
Video Encode	4Kp30 | 4x 1080p30 | 9x 720p30 (H.264/H.265)
Video Decode	4Kp60 | 2x 4Kp30 | 8x 1080p30 | 18x 720p30 (H.264/H.265)
Connectivity	Gigabit Ethernet, 802.11ac wireless
Camera	1x MIPI CSI-2 connector
Display	HDMI
USB	1x USB 3.0 Type A,2x USB 2.0 Type A, USB 2.0 Micro-B
Others	40-pin header (GPIO, I2C, I2S, SPI, UART)
	12-pin header (Power and related signals, UART)
	4-pin Fan header
Mechanical	100 mm x 80 mm x 29 mm

C.2. Cámara PTZ

Tabla C.2: Especificaciones técnicas cámara Hikvision

ESPECIFICACIONES DS-2DE2A204IW-DE3	
Material	Aluminum alloy, PC, PC+ABS
Dimensions	ϕ 130.7mm × 101.7 mm (ϕ 5.15"× 4.00")
Approx.	0.53 kg (1.17 lb)
Protection Level	IP66 Standard, IK 10, Surge Protection, ±2kV Line to Gnd, ±1kV Line to Line, IEC61000-4-5
Power	12 VDC (Max. 10.9 W, including Max. 4.2 W for IR)
	PoE 802.3af (Max. 12.2 W, including Max. 4.2 W for IR)
Working Temperature	-20ºC to 60ºC (-4ºF to 140ºF)
Working Humidity	≤ 90 %
Movement Range (Pan)	0º to 355º
Movement Range (Tilt)	From 0º to 90º
Max. Resolution	1920 × 1080
Web Browser	IE 8 to 11,Chrome 31.0+, Firefox 30.0+, Edge 16.16299+

Tabla C.3: Precios de la línea Jetson 2022.

Producto de la línea Jetson	**Precio [usd]**
Jetson Nano	99
Jetson Nano 2GB Developer Kit	59
Jetson TX2 NX2	149
Jetson Xavier NX	399