



## **Control de Ingresos y Egresos de Vehículos en Patios de Distribución mediante Redes Neuronales Convolucionales y RFID en Raspberry Pi con Unidad Hailo-8L**

**Profesora Titular: Ing. Grettel Barceló Alonso, PhD**

### **Equipo #20**

#### **Avance 4. Modelos alternativos**

- A01794188 - Francisco Xavier Bastidas Moreno
- A01794653 - Raúl Jesús Coronado Aguirre
- A01794327 - Juan Sebastián Ortega Briones

## Contents

Introducción .....	3
Modelo 1 Yolo 6.....	5
Modelo 2 Yolo 11 .....	5
Modelo 3 Detección de movimiento con sustracción de fondo en un área específica .....	7
Modelo 4 Detección de movimiento con sustracción de fondo y seguimiento por centroides.....	8
Modelo 5 (Entrenado con yolov9c).....	11
Modelo 6 (Flujo Optico) .....	16
Comparativo de modelos YOLO estándares y entrenado con nuestro Dataset.....	19
Comparar el rendimiento de los modelos obtenidos. ....	20
Seleccionar los dos modelos que proporcionen el mejor rendimiento. ....	22
Elegir el modelo individual final.....	22
Desempeño del modelo:.....	22
Ventajas de este modelo: .....	22
Desventajas de este modelo .....	23

# Introducción

En el marco de nuestro proyecto sobre el control de ingresos y egresos de vehículos en patios de distribución, hemos explorado una variedad de modelos basados en tecnologías de redes neuronales convolucionales y RFID implementados en Raspberry Pi con la unidad Hailo-8L. Este documento compara y evalúa seis modelos alternativos, cada uno utilizando diferentes algoritmos y técnicas. El objetivo es determinar el modelo que proporciona el mejor rendimiento en términos de precisión, eficiencia y adaptabilidad a diferentes condiciones ambientales y operativas.

## Desarrollo de Modelos Alternativos

Nos enfocamos en los siguientes modelos

1. **Modelo Basado en YOLOv6:** Este es nuestro modelo base, que utiliza YOLOv6 para la detección de vehículos. Ha demostrado ser eficiente, pero con margen de mejora en detección bajo ciertas condiciones. Usando una raspberry pi con potenciado Hailo
2. **Modelo Basado en YOLOv11:** Se hizo un entrenamiento de Yolo 11 para comparar.
3. **Detección de Movimiento con Sustracción de Fondo:** Utilizando técnicas de procesamiento de imágenes para identificar vehículos mediante cambios en el fondo. Es sensible a variaciones ambientales que pueden afectar su eficacia.

4. **YOLOv9c con Google Colab:** Una iteración adicional del modelo YOLOv9c, utilizando recursos de Google Colab y Roboflow para mejorar las detecciones de los carros con imágenes propias y específicas del feed de la cámara,
5. **Flujo Óptico:** El último modelo incorpora técnicas de flujo óptico para seguir el movimiento de los vehículos, lo que permite una detección robusta con menos recursos computacionales utilizando Yolov5
6. **Comparativo de modelos YOLO estándar y entrenado:** Comparativo de modelos Yolo estándar con nuestro dataset y con un modelo de YOLO entrenado con nuestro dataset.

## Modelo 1 Yolo 6

Este Modelo es el de la actividad anterior, en este tenemos detecciones de entradas y salidas

Matriz	de	Confusión	para	'Salidas':
Verdaderos		Positivos	(TP):	7
Falsos		Negativos	(FN):	4
Recall (Sensibilidad): 0.64				

Matriz	de	Confusión	para	'Entradas':
Verdaderos		Positivos	(TP):	2
Falsos		Negativos	(FN):	3
Recall (Sensibilidad): 0.40				

Matriz	de	Confusión	para	'Detección	General':
Verdaderos		Positivos		(TP):	13
Falsos		Negativos		(FN):	3
Recall (Sensibilidad): 0.81					

## Modelo 2 Yolo 11

```
Custom Training

%cd {HOME}

!yolo task=detect mode=train model=yolo11s.pt data={dataset.location}/data.yaml epochs=10 imgsz=640 plots=True

Class      Images  Instances  Box(P  R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00, 4.95it/s]
all         18         31         0.807  0.942  0.948    0.782

10 epochs completed in 0.041 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 19.2MB
Optimizer stripped from runs/detect/train/weights/best.pt, 19.2MB

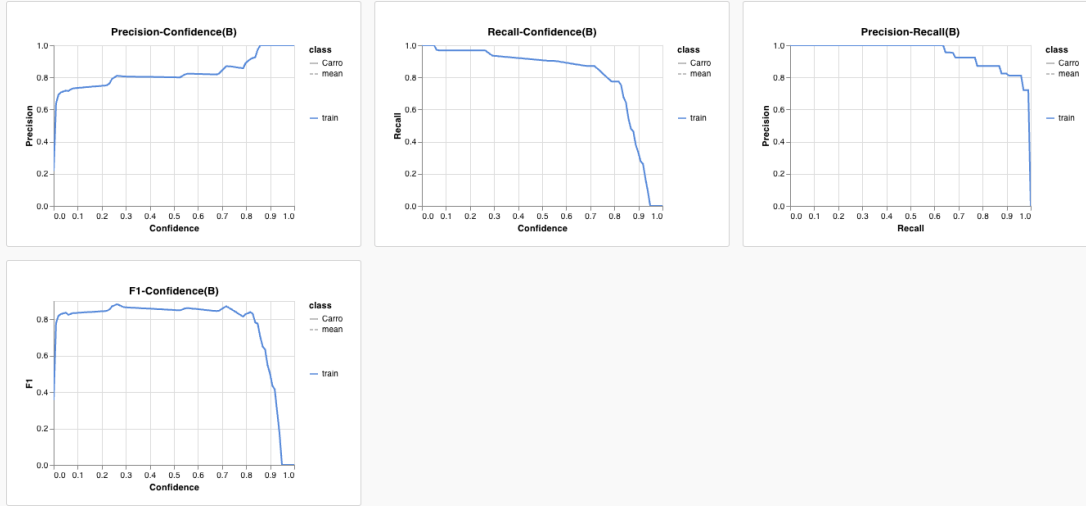
Validating runs/detect/train/weights/best.pt...
Ultralytics 8.3.18 Python-3.10.12 torch-2.4.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
YOLO11s summary (fused): 238 layers, 9,413,187 parameters, 0 gradients, 21.3 GFLOPs
Class      Images  Instances  Box(P  R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00, 5.74it/s]
all         18         31         0.806  0.938  0.948    0.782

Speed: 0.2ms preprocess, 4.4ms inference, 0.0ms loss, 0.9ms postprocess per image
```

## Validate fine-tuned model

```
!yolo task=detect mode=val model={HOME}/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml
```

Ultralytics 8.3.18 Python-3.10.12 torch-2.4.1+cu121 CUDA:0 (Tesla T4, 15102MiB)  
YOLO11s summary (fused): 238 layers, 9,413,187 parameters, 0 gradients, 21.3 GFLOPs  
val: Scanning /content/datasets/Proyecto\_Equipo20-2/valid/labels.cache... 18 images, 0 backgrounds, 0 corrupt: 100% 18/18 [00:00<?, ?it/s]  
Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 [00:01<00:00, 1.49it/s]  
all 18 31 0.807 0.942 0.948 0.792  
Speed: 0.2ms preprocess, 28.5ms inference, 0.0ms loss, 30.3ms postprocess per image





**Modelo 3 Detección de movimiento con sustracción de fondo  
en un área específica**

En este modelo, utilizamos librerías de OpenCV, NumPy, y imutils, junto con diferentes técnicas de preprocesamiento de imágenes, como sustracción de fondo y operaciones morfológicas para eliminar sombras y unificar los objetos cuando hay cortes en la detección de movimiento.

Determinamos un área sobre la cual nos centraremos al detectar el movimiento, ignorando todo lo que suceda fuera de esta, y dentro de esta área agregamos una línea que nos servirá para detectar cuando un objeto la cruza aumentando el contador. En la siguiente imagen podemos apreciar lo anteriormente mencionado:



Logramos detectar los vehículos correctamente, pero el contador se incrementa más de lo real debido a cuenta cada fotograma que detecta mientras el vehículo va entrando. En la siguiente imagen se muestra el resultado de la entrada de un vehículo:



Pueden encontrar el código utilizado en el repositorio de GitHub con el nombre de Counter.py en la carpeta de Avance4.

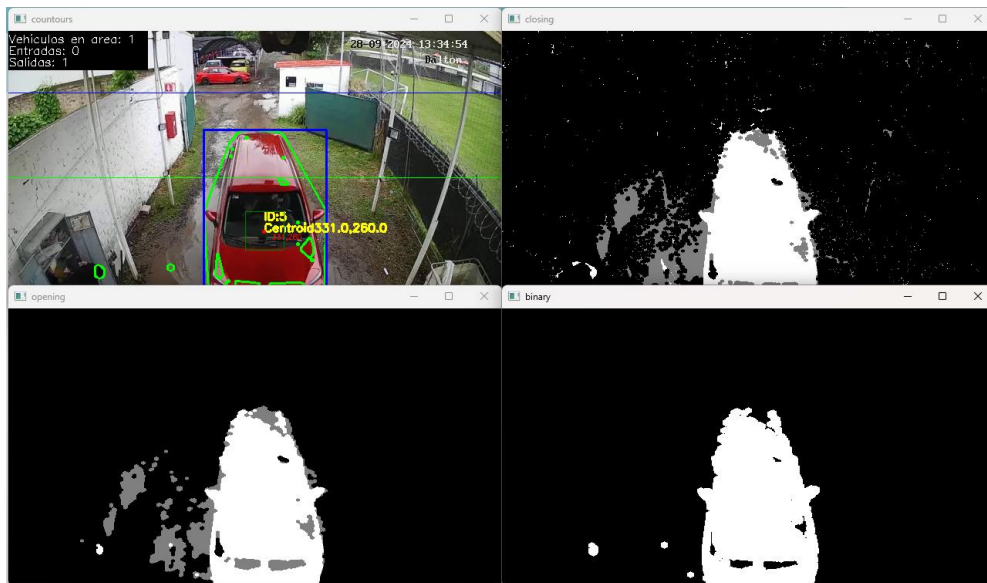
## **Modelo 4 Detección de movimiento con sustracción de fondo y seguimiento por centroides**



En este modelo, utilizamos librerías de OpenCV, NumPy, y Pandas, junto con diferentes técnicas de preprocesamiento de imágenes, como la transformación a escala de grises, sustracción de fondo y operaciones morfológicas.

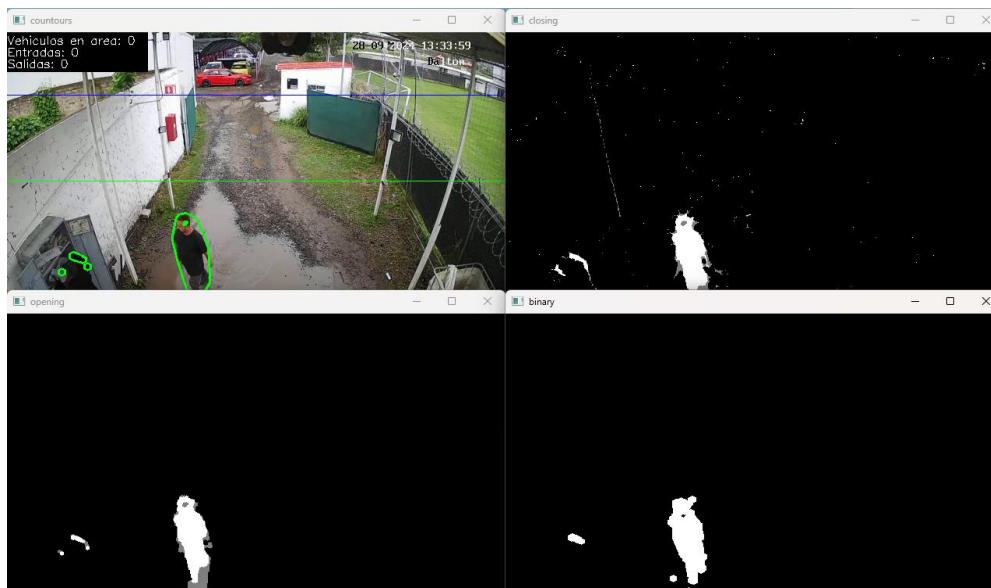
Con la sustracción de fondo comparamos cada cuadro del video con un modelo del fondo estático, restando la imagen actual de este modelo. Los cambios resultantes resaltan los objetos en movimiento, permitiendo su identificación y seguimiento. Después aplicamos operaciones morfológicas de erosión, dilatación y apertura – cierre:

- Erosión: Elimina píxeles de los bordes de los objetos, ayudando a reducir el ruido.
- Dilatación: Agrega píxeles a los bordes de los objetos, aumentando su tamaño.
- Apertura y cierre: Combinaciones de erosión y dilatación que ayudan a eliminar pequeños objetos o rellenar huecos.



Una vez aplicadas estas transformaciones tenemos un mejor seguimiento de los objetos detectados. Dibujamos contornos alrededor de las detecciones y dos líneas horizontales en la imagen, una azul y una verde, cualquier cosa por encima de la línea azul no se rastrea, pero cualquier cosa por debajo sí. La segunda línea verde se utiliza para hacer seguimiento de si el coche se mueve hacia arriba o hacia abajo, indicando si entra o sale de las instalaciones.

También se crearon variables para las áreas mínima y máxima permitidas para que un contorno obtenga un id y pueda ser contado como entrada o salida. Esto es importante para no tomar dentro del seguimiento otros objetos que no sean vehículos, como personas, perros o aves que aparecen en las detecciones.



Como podemos observar en la imagen superior detecta una persona entrando y cruzando la línea horizontal verde que realiza el conteo, pero al no tener el área suficiente no se le asigna un id y no cuenta como entrada.

Al terminar de ajustar los parámetros y realizar pruebas obtuvimos un buen resultado de detección con algunos errores por corregir, en algunos casos cuando entran o salen dos vehículos muy cerca uno del otro les asigna un mismo id y cuentan como solo una entrada o salida según sea la dirección de cruce.

Además, existen factores que afectan las detecciones como lo son charcos que reflejan el cielo y al pasar los carros crean más movimiento generando ruido al momento de dibujar los contornos del objeto.

El código utilizado para este modelo de detección se encuentra en el repositorio de GitHub en la carpeta de Avance4 con el nombre de Counter\_background\_subtractor.py

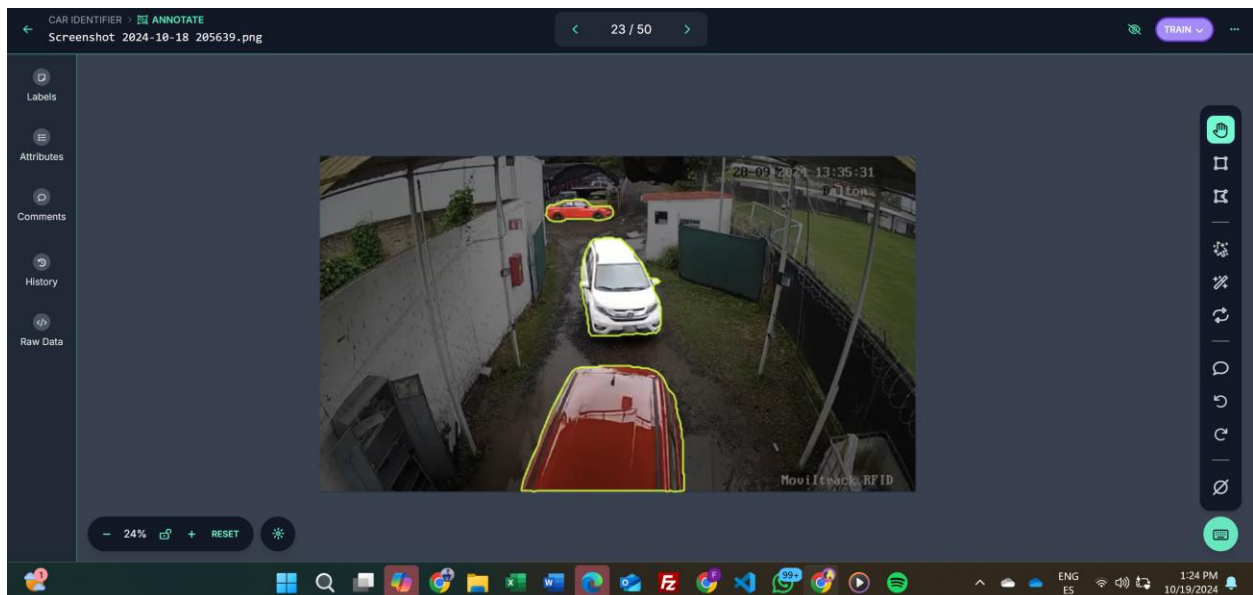
## Modelo 5 (Entrenado con yolov9c)

Tras identificar irregularidades en la detección continua de vehículos utilizando el modelo preentrenado de YOLOv6n, decidimos entrenar un nuevo modelo basado en YOLOv9 con imágenes específicas del feed de nuestra cámara. Esta decisión se tomó luego de observar que el modelo anterior perdía de vista los vehículos cuando estos pasaban la mitad de su longitud o realizaban movimientos repentinos, afectando la precisión en la detección.

Para mejorar los resultados, utilizamos Roboflow para generar anotaciones poligonales precisas en 100 imágenes seleccionadas, lo que proporcionó mayor precisión y detalle al modelo. Estas imágenes cubren una variedad de escenarios, como uno o dos vehículos en el campo visual, vehículos en el fondo y algunas sin vehículos, lo que nos permitió abarcar situaciones reales en el entorno de monitoreo.

Dado que las cámaras se encuentran en ubicaciones fijas, los ángulos de captura son consistentes. Esto facilitó el proceso de anotación, permitiéndonos enfocarnos en capturar imágenes a diferentes horas del día, de modo que el modelo pueda adaptarse a los cambios de iluminación, ya sea luz solar o artificial.

Además, algunas imágenes fueron etiquetadas para incluir solo partes de los vehículos, lo que ayuda a que el modelo mantenga la detección, aunque el vehículo esté parcialmente fuera del campo de visión. Esta estrategia asegura que, a diferencia del modelo anterior, el nuevo modelo YOLOv9 pueda seguir detectando el vehículo en todo momento.




Para el primer entrenamiento, utilizamos Roboflow, que no solo facilitó la gestión del dataset, sino también el preprocesamiento mediante aumentaciones de las imágenes, generando escenarios variados como cambios en el contraste, orientación y saturación. Roboflow también recomendó un split para los datos, ajustando la división original de 70/30/10 (entrenamiento/validación/prueba) a un 88/8/4. Sin embargo, notamos que este split podría contribuir al sobreentrenamiento, por lo que planeamos utilizar una división diferente en futuros entrenamientos para evitar este problema.

Este primer modelo dio excelentes resultados. Tras 300 épocas de entrenamiento, las métricas mejoraron notablemente, con un avance significativo después de las primeras 25 épocas y estabilizándose progresivamente hacia el final del entrenamiento. Este progreso es visible en las gráficas de pérdidas y métricas a lo largo del proceso. La precisión y el recall se incrementaron de forma constante, mostrando una mejora clara

240 Total Images

[View All Images →](#)



Dataset Split

TRAIN SET

88%

210 Images

VALID SET

8%

20 Images

TEST SET

4%

10 Images

Preprocessing

Auto-Orient: Applied

Resize: Stretch to 640x640

Augmentations

Outputs per training example: 3

Crop: 0% Minimum Zoom, 20% Maximum Zoom

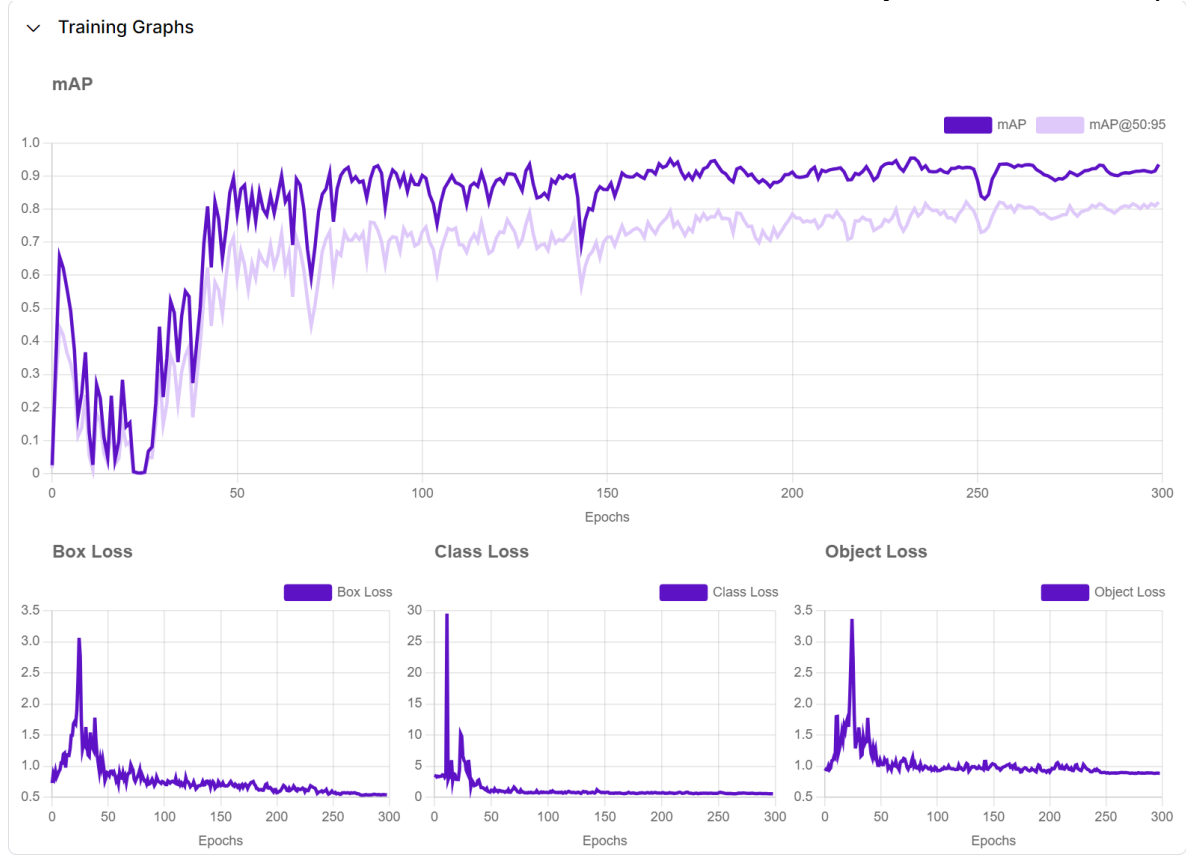
Rotation: Between -15° and +15°

Hue: Between -14° and +14°

Saturation: Between -25% and +25%

Blur: Up to 2.5px

en la detección de vehículos y sus partes.



v1 2024-10-19 4:12am

Generated on Oct 18, 2024

Download Dataset

Edit

car-identfier/1

Model Type: Roboflow 3.0 Object Detection (Fast)

Checkpoint: COCO

mAP 93.6%

Precision 89.8%

Recall 85.7%

Link público al modelo: <https://app.roboflow.com/detecci-de-daos/car-identfier/1>



Podemos ver como en las inferencias los porcentajes de confianza son altos en los autos aun cuando el auto no está completamente en la imagen o si hay uno enfrente del otro.

Luego de esto, entrenamos un segundo modelo utilizando YOLOv9c apoyándonos de la GPU T4 de Google Colab. Se mantuvieron las mismas condiciones de entrenamiento: 300 épocas, tamaño de imagen de 640x640, y batch size de 8.

Aunque los resultados con YOLOv9c fueron ligeramente inferiores a los obtenidos con el modelo de Roboflow, se observaron mejoras significativas comparadas con el modelo preentrenado básico que habíamos utilizado inicialmente.

```

Entrenamiento PI Carro con Roboflow.ipynb
Archivo  Editar  Ver  Insertar  Entorno de ejecución  Herramientas  Ayuda  Guardando...

+ Código  + Texto

[ ] ! pip install ultralytics
Mostrar salida oculta

[ ] !pip install roboflow
Mostrar salida oculta

Aqui en el el dataset que descargues o crees solo copia el link que te da del dataset

!pip install roboflow
from roboflow import RoboFlow
rf = RoboFlow(api_key="m1VyCC3H47AntySf2aZj")
project = rf.workspace("detecci-de-daos").project("car-identifier")
version = project.version(1)
dataset = version.download("yolov9")
Mostrar salida oculta

Selecciona el modelo con el que descargaste el dataset arriba

from ultralytics import YOLO
model = YOLO("yolov9c.pt") # load a pretrained model (recommended for training)

Creating new Ultralytics Settings v0.6 file
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=to/dir'. For help see https://docs.ultralytics.com/quickstart/#ultralytics-settings.
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov9c.pt to 'yolov9c.pt'...
100% [49.4M/49.4M [00:02<00:00, 22.0MB/s]

Fijate bien en el imgs2 de tus fotos y ajusta las epochs y el batch

[ ] # Train the model
# Train the model
results = model.train(data="/content/Car-identifier-1/data.yaml", epochs=300, imgs=640, batch=8)
Mostrar salida oculta

```



Utilizando los mejores pesos del entrenamiento logramos visualizar las inferencias .



Este tipo de entrenamientos puede mejorar significativamente nuestro modelo final, obteniendo detecciones más precisas, lo cual contribuirá a una gestión de inventarios más eficiente para la empresa.

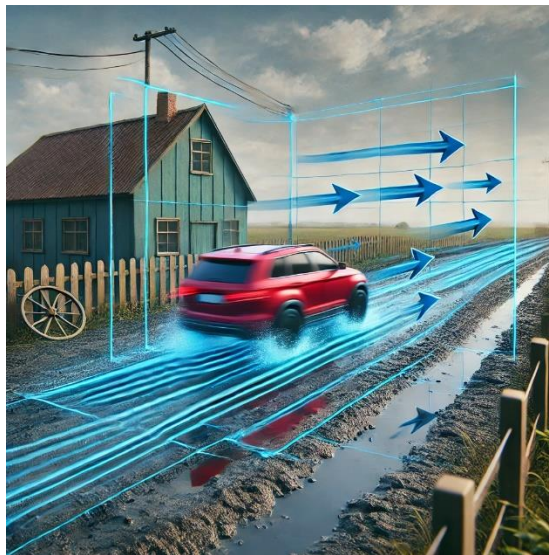
Los resultados del entrenamiento del modelo se pueden encontrar aquí

<https://wandb.ai/fbastidas98-tecnol-gico-de-monterrey/Ultralytics/runs/ngp8bq8d?nw=nwuserfbastidas98>

## Modelo 6 (Flujo Óptico)

Para el último modelo utilizamos una técnica aprendida en la materia de visión computacional llamada flujo óptico, se usa para estimar el movimiento de objetos entre dos cuadros consecutivos de una serie temporal de imágenes. Se basa en el cambio aparente en la posición de los objetos causado por el movimiento del objeto o de la cámara. Es ampliamente utilizado en diversas aplicaciones, como la detección y seguimiento de vehículos, donde es fundamental para identificar la trayectoria y la velocidad de los objetos en movimiento.

En el contexto de nuestro proyecto, el flujo óptico podría ser utilizado para mejorar la precisión del seguimiento de vehículos, permitiendo una detección más robusta a través de diferentes condiciones de iluminación y velocidades variables de los vehículos. A diferencia de otros modelos que pueden requerir una identificación continua de cada vehículo en cada cuadro, el flujo óptico nos permite seguir el movimiento de un vehículo con menos recursos computacionales, haciéndolo ideal para sistemas en tiempo real donde la eficiencia es crucial.



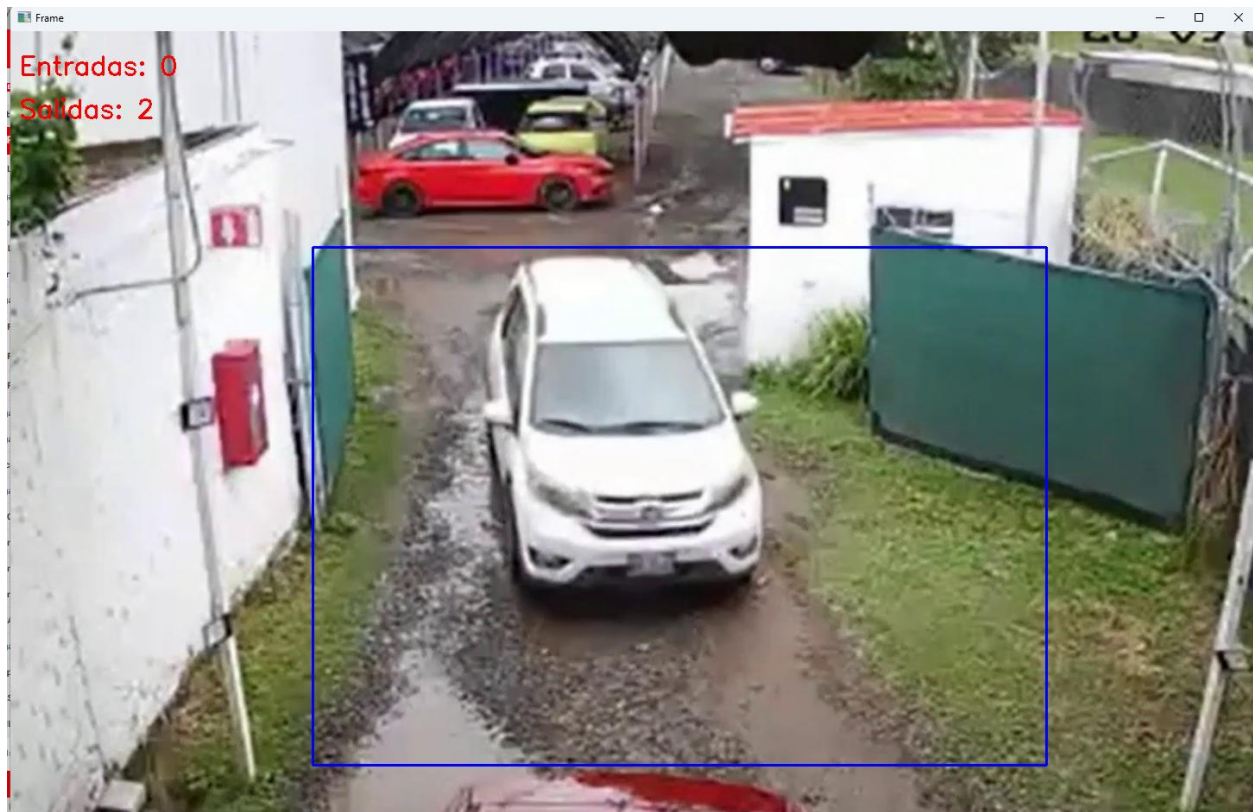


Cargamos YOLOv5, asegurándonos de utilizar una versión preentrenada y optimizada del modelo, lo que garantiza una alta precisión en la detección de vehículos.

**Definición del ROI:** Definimos el ROI manualmente, basándonos en las necesidades específicas de la ubicación de la cámara. Este enfoque nos permite concentrar la detección en áreas clave, minimizando los falsos positivos y maximizando la eficiencia.

**Detección de Vehículos:** Procesamos cada frame del video mediante la función `detect_vehicles`, aplicando el modelo solo dentro del ROI para detectar vehículos específicamente en esta área, lo cual es fundamental para filtrar resultados por clases de interés (vehículos).

**Conteo de Vehículos:** Implementamos una lógica que cuenta los vehículos cada vez que cruzan ciertos umbrales dentro del ROI, utilizando un margen para evitar el conteo múltiple del mismo vehículo mediante el registro de cada vehículo detectado y un intervalo controlado entre conteos para el mismo vehículo.



El proyecto se encuentra en el repositorio de github con el nombre [FlujoOpticoYolov5.p7](#)

Tabla comparative de modelos YOLO

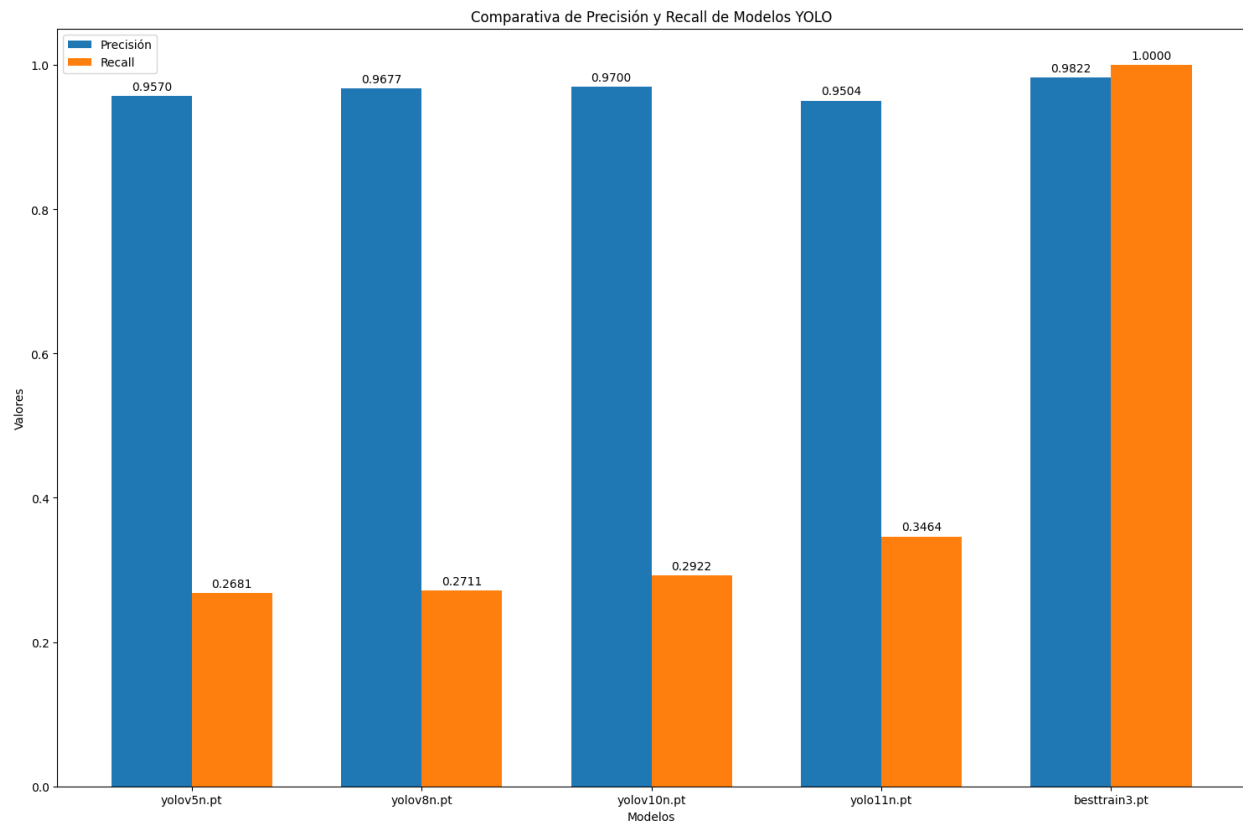
Network Name	mAP	FPS (Batch Size=8)	Input Resolution (HxWxC)	Params (M)
yolov5s_wo_spp	34.79	195	640x640x3	7.85
yolov6n	34.29	355	640x640x3	4.32
yolov7	50.6	35	640x640x3	36.91
yolov7_tiny	37.07	197	640x640x3	6.22
yolov8l	52.44	27	640x640x3	43.7
yolov8m	49.91	60	640x640x3	25.9
yolov8n	37.02	464	640x640x3	3.2
yolov8s	44.58	151	640x640x3	11.2
yolov8x	53.45	14	640x640x3	68.2
yolov9c	52.6	27	640x640x3	25.3
yolox_l_leaky	48.68	27	640x640x3	54.17

## Comparativo de modelos YOLO estándares y entrenado con nuestro Dataset

Se comparo el desempeño de los modelos YOLO estándares y entrenado usando nuestro dataset y estos fueron los resultados:

Modelo	Precisión	Recall
yolov5n.pt	0.957	0.2681
yolov8n.pt	0.9677	0.2711
yolov10n.pt	0.97	0.2922
yolo11n.pt	0.9504	0.3464
besttrain3.pt	0.9822	1

\* besttrain3.pt es el modelo entrenado en Roboflow del [modelo 5 anterior](#).



\*Datos generados con: [https://github.com/SebastianOrtega/cv-hailo-tec/blob/main/Avance4/comparar\\_modelos.ipynb](https://github.com/SebastianOrtega/cv-hailo-tec/blob/main/Avance4/comparar_modelos.ipynb)

## Comparar el rendimiento de los modelos obtenidos.

Los modelos basados en YOLO son extremadamente rápidos y precisos en la detección, pero requieren considerable poder computacional.

RFID, aunque altamente preciso para el seguimiento de vehículos, no ofrece capacidades visuales ni clasificación de entrada o salida del id detectado,

La sustracción de fondo es efectiva en condiciones controladas, pero puede fallar en ambientes dinámicos.

El flujo óptico ofrece una solución menos costosa computacionalmente y es efectivo en el seguimiento de trayectorias, pero puede no ser tan preciso en la identificación de vehículos específicos.

Realizamos un experimento en donde utilizamos un mismo video para comparar el desempeño de los distintos modelos anteriormente mencionados, comparando los resultados finales de cada modelo y el resultado real.

En el video empleado hay 2 salidas y una entrada de vehículos, para lo cual obtuvimos lo siguiente:

### **Modelo flujo óptico**

Detecciones Totales:4

Detecciones correctas: 2

Detecciones incorrectas:2

### **Detección de movimiento con sustracción de fondo en un área específica**

Detecciones Totales:44

Detecciones correctas: 2

Detecciones incorrectas:42



### **Modelo de detección de movimiento con sustracción de fondo y seguimiento por centroides**

Detecciones Totales:3

Detecciones correctas: 3

Detecciones incorrectas:0



## Seleccionar los dos modelos que proporcionen el mejor rendimiento.

Los mejores resultados se obtuvieron con Yolo 11n como segundo lugar y Yolo 9n entrenado como primer lugar

## Elegir el modelo individual final.

El mejor modelo seleccionado para esta tarea fue **Yolo 9**, entrenado, bajo el nombre de archivo besttrain3.pt.

### Desempeño del modelo:

- **Recall: 1.0000** El recall de 1.0000 indica que el modelo ha detectado todos los objetos reales presentes en los datos (es decir, no ha dejado de detectar ningún objeto de interés). Esto asegura que no se omiten vehículos en las imágenes, lo que es crucial para el seguimiento de trayectorias. La ausencia de falsos negativos significa que el modelo garantiza la detección continua y precisa de los vehículos, sin perder ningún objeto a lo largo del video.
- **Precisión: 0.9822** La precisión de 0.9822 (o 98.22%) confirma que la mayoría de las detecciones realizadas fueron correctas. Este alto valor de precisión implica que el número de falsos positivos (detecciones incorrectas) es mínimo, con un margen de error del 1.78%. Este bajo porcentaje de detecciones erróneas no afecta significativamente el desempeño global del sistema.

### Ventajas de este modelo:

**Detección robusta en cada cuadro:** El uso de este modelo nos asegura la detección en todos los cuadros del video. Esto es esencial para mantener un seguimiento puntual y continuo de cada uno de los vehículos dentro de cada cuadro, permitiendo determinar su trayectoria de manera precisa. La consistencia en la detección de vehículos es clave para evitar lagunas en la información sobre el movimiento, lo que nos permitirá un análisis más detallado del comportamiento y dirección de los vehículos.

**Mejora del seguimiento de trayectorias:** Hasta el momento, el algoritmo de detección de trayectorias había mostrado dificultades debido a la pérdida de vehículos en varios cuadros del video, lo que llevaba a un seguimiento intermitente y a la identificación incorrecta de vehículos. Esto impedía obtener trayectorias claras y consistentes. Con el

modelo **besttrain3.pt**, el seguimiento continuo se verá mejorado de forma significativa, ya que no habrá pérdida de vehículos, lo que permitirá identificar correctamente los vehículos que se mueven a lo largo del video y reconstruir trayectorias completas.

**Impacto esperado en la implementación:** Al implementar este modelo junto con el algoritmo de seguimiento de trayectorias, esperamos un aumento considerable en la calidad del seguimiento. Con el recall perfecto y la alta precisión del modelo, las trayectorias serán claras y precisas, eliminando la detección errónea de vehículos en diferentes lugares como instancias distintas. Esto permitirá una mejor toma de decisiones basada en los movimientos reales de los vehículos.

En conclusión, la elección del modelo **besttrain3.pt** no solo responde a su excelente desempeño en términos de recall y precisión, sino también a su capacidad de proporcionar un seguimiento confiable y preciso de los vehículos en el video. Esto será crucial para mejorar la efectividad de nuestro sistema de seguimiento de trayectorias y asegurar la consistencia de las detecciones.

## Desventajas de este modelo

El tiempo de inferencia es 10 veces mayor que el modelo sin entrenar. Esto nos obliga a ocupar GPU o en nuestro caso compilar el modelo para Hailo.