InFo-Cyber. Journal of Cybersecurity and Digital Forensics.
Volumen 1, Número 1. Año 2026. ISSN: en trámite.
https://revistas.ufasta.edu.ar/index.php/InFoCyber/index

INFO-CYBER CYBERSECURITY AND DIGITAL FORENSICS JOURNAL

Detección e Identificación de Vehículos Mediante Técnicas de Inteligencia Artificial

Vehicle Detection and Identification Using Artificial Intelligence Techniques

E. Valentín Iglesias

Ministerio Público Fiscal de San Luis (Argentina)

liciglesiasvalentin@gmail.com

https://orcid.org/0009-0001-4508-5859

Gustavo J. Meschino

Universidad FASTA, Mar del Platas (Argentina)

gmeschino@ufasta.edu.ar

https://orcid.org/0000-0003-3835-7745

Artículo de Investigación

Recibido: 06-09-2024. Aceptado: 14-02-2025.
Publicado: 03-03-2026

Universidad FASTA. Facultad de Ingeniería; Mar del Plata, Argentina

Resumen

Este trabajo presenta un modelo para la detección e identificación de vehículos en la vía pública. A este fin, se considera un modelo de inteligencia artificial que utiliza como base el modelo YOLO (You Only Look Once) y el algoritmo DeepSORT (Simple Online and Realtime Tracking). El modelo resultante logró mantener el rastreo de vehículos en diferentes situaciones de oclusiones y condiciones ambientales variables. Una vez detectado e identificado el vehículo, se procuró su rastreo por el modelo de forma eficiente. Como derivado del desarrollo del modelo, se creó un conjunto de datos de vehículos que cotidianamente se encuentran en las calles de la República Argentina, que contiene 1769 imágenes etiquetadas para el sistema YOLO. Este trabajo representa una prueba de concepto para generar nuevos modelos con funcionalidades superiores. Los resultados muestran que la dirección es adecuada.

Palabras clave

detección, identificación, oclusión, rastro, redes neuronales convolucionales

Abstract

This paper presents a model for detecting and identifying vehicles on public roads. To this end, it considers an artificial intelligence model based on the YOLO (You Only Look Once) model and the DeepSORT (Simple Online and Realtime Tracking) algorithm. The resulting model was able to track vehicles in different situations of occlusions and variable environmental conditions. Once the vehicle was detected and identified, the model tracked it efficiently. As a result of the model's development, a dataset of vehicles commonly found on the streets of Argentina was created, containing 1,769 images labeled for the YOLO system. This work represents a proof of concept for generating new models with superior functionalities. The results show that the direction is appropriate.

Keywords

detection, identification, occlusion, tracking, convolutional neural networks

Este artículo surge del trabajo final de la especialización en Informática Forense del primer autor, con el apoyo en calidad de director del Dr. Gustavo Javier Meschino. El trabajo de investigación no tuvo ningún tipo de apoyo financiero. Sin embargo, se contó con el apoyo del Procurador de la Provincia de San Luis Dr. Luis M. Martínez. para la autorización de la utilización videos de cámaras de vigilancia ciudadana con los que el modelo ad hoc fue entrenado y validado. El primer autor de este artículo es el Licenciado Especialista Eduardo Valentín Iglesias. La contribución del primer autor se enfocó en el desarrollo y redacción total del trabajo de investigación y la del segundo autor fue en calidad de tutor, guía y participe.

EDUARDO VALENTÍN IGLESIAS se desempeña en el Ministerio Publico Fiscal de la Provincia de San Luis, en el Departamento de Delitos Complejos.

GUSTAVO JAVIER MESCHINO es el director del Laboratorio de Bioingeniería de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata y el codirector del Grupo de Informática y Salud de la Universidad FASTA, ambas en Mar del Plata.

I. INTRODUCCIÓN

El despliegue de modelos de Inteligencia Artificial en la vida cotidiana ya es una realidad ineludible para cualquier actor humano de la sociedad; se encuentran actualmente en prácticamente todos los aspectos de la vida y uno de los aspectos en que rápidamente se destacaron es en la visión. Estos modelos de aprendizaje automático se crearon específicamente para imágenes y, por extensión, para videos. Tuvieron su auge hace no mucho tiempo y comenzaron con enfoques matemáticos sencillos. Nos referimos a las redes neuronales convolucionales o, por sus siglas, CNN (*Convolutional Neural Networks*).

Las imágenes para estos algoritmos se representan como una pila de *matrices* o, con una denominación más actual, *tensores de orden 3*. En otras palabras, podríamos decir que una matriz es una tabla de dos dimensiones, las filas y las columnas. Mientras que un tensor es una estructura de datos multidimensional, es decir, puede tener más de dos dimensiones, como por ejemplo una pila de matrices.

Una imagen se determina por su alto y su ancho en píxeles y su profundidad (la cantidad de canales de colores que posea). Aplicar una operación convolucional sobre la imagen es realizar operaciones matemáticas simples como la suma o la multiplicación escalar entre el tensor que conforma la imagen y un filtro que tiene pesos. Esto genera una matriz a la cual se puede aplicar una función de activación y luego, eventualmente, una capa de agrupamiento (traducción no convencional de *pooling*[1]). Esta secuencia de operaciones (en orden determinado por pruebas) se denomina la arquitectura de la red y su función es detectar características (*features*) en las imágenes. En las primeras operaciones (capas) se detectan elementos simples como líneas o bordes y, conforme van pasando las capas convolucionales, se van descubriendo características cada vez más abstractas o de mayor nivel.

Detectar, identificar y rastrear fueron conceptos clave en el desarrollo de este trabajo, siendo el modelo YOLO [1] un modelo que, además, lo puede hacer en tiempo real debido a su eficiencia en el cálculo.

Existen avances que, por su relevancia en el contexto de las redes neuronales convolucionales, merecen una mención especial. El más relevante de ellos es el que poseen los sistemas de conducción autónomos de Tesla que, por medio de su supercomputadora DOJO, entrenan modelos en sus inicios de redes neuronales convolucionales de aprendizaje automático. Con el fin de mejorar su sistema de asistencia a la conducción llamado FSD (por *full self-driving*) y deno solo de ser eficaz sino también de actuar en tiempo real, analiza imágenes sin procesar para realizar segmentación semántica, detección de objetos y estimación de profundidad monocular. En estos sistemas, una compilación completa de redes neuronales de *Autopilot* incluye 48 redes que tardan 70 000 horas de GPU en entrenarse, produciendo 1 000 vectores de predicción distintos en cada paso temporal [2]. A pesar de que a la fecha actual dejaron de utilizarlas, estas fueron parte de sus cimientos fundacionales.

A una distancia considerable de la realidad de Tesla y con la modestia obligada a nuestra capacidad computacional y operacional, este trabajo de investigación construyó un modelo de aprendizaje automático *ad hoc*. Esta labor fue realizada mediante el uso del algoritmo YOLO, que permitió la detección de vehículos con fines identificatorios, y del algoritmo Deep SORT [3], cuya finalidad es identificar, rastrear y agregar un parámetro más de asociación profunda al modelo, para obtener información extra de la apariencia. Con el objetivo de mantener el rastreo del vehículo por periodos de oclusiones más largos. Es decir, en temporalidades más amplias y ante la existencia de obstáculos fijos o móviles que se encuentren en la vía pública, que bloqueen total o parcialmente la vista del vehículo rastreado. Para crear este modelo se utilizaron durante el entrenamiento videos de cámaras de videovigilancia ciudadana.

En una primera fase se realizó el procesado de videos para extraer las imágenes necesarias para entrenar el modelo. En una segunda fase se entrenó el modelo, utilizando los modelos preentrenados de YOLO. De esta forma, se obtuvieron las métricas asociadas que permitieron analizar la efectividad que logramos. Finalmente, se le incorporó la lógica del algoritmo Deep SORT con el objetivo de rastrear a los vehículos en periodos de oclusión largos. Así, se pretendió proporcionar un punto de partida para futuras investigaciones.

Como se desprende de este punto, las redes neuronales convolucionales son la herramienta elegida para llevar adelante esta investigación. Este trabajo surgió del visible desarrollo en seguridad pública que las fuerzas de seguridad obtuvieron al utilizar para el análisis de hechos de inseguridad el material generado por cámaras de video. Advertida esta situación por el investigador, se propuso aprovechar el rico y amplio campo de desarrollo que ofrecen. Ya en el año 2022, la provincia de San Luis amplió el desarrollo del sistema de inteligencia de seguridad ciudadana con la incorporación de 870 cámaras de videovigilancia a las que ya mantenía en funcionamiento, instaladas en distintas ciudades de la provincia. Estas cámaras cuentan con diferentes funcionalidades: observación, lectura de chapas patentes, detección de marca, color y tipo de vehículo.

Estos equipos capturan alrededor de 8 millones de lecturas

[1] *Pooling* su función es reducir la dimensionalidad de mapas de características extrayendo valores representativos de regiones locales en la imagen. Disminuyendo la carga computacional.

mensuales, con operarios que monitorean en tiempo real las 24 horas del día, siendo el fin el resguardo de la integridad de las personas que habitan en la provincia y aquellos que circunstancialmente transitan por la misma.

II. REDES CONVOLUCIONALES

Para comprender este tipo de redes, es fundamental y se requiere el concepto de red neuronal artificial. De forma concisa podemos indicar que una red neuronal artificial es un modelo matemático que acepta entradas de datos (vectores numéricos), y que los procesa multiplicándolos con pesos, cuya función es modificar el estado de la neurona. Su objetivo es especializar la actividad de la neurona para la cual se entrena la red, es decir, llevar la red, iterativamente, a obtener un resultado deseado (*target*).

Cada neurona además cuenta con una función que se aplica a la combinación lineal de las entradas y los pesos asociados, determinando un nivel de activación que es un valor numérico (Fig. 1). La red de neuronas interconectadas cuenta con una salida que contiene la predicción.

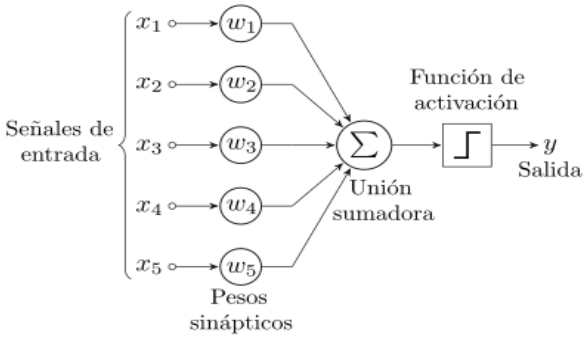

Fig. 1. Perceptrón simple; modelo neuronal básico.

Estos modelos matemáticos se inspiran en la forma en que las neuronas biológicas manipulan la información.

Las neuronas se conectan entre sí. Para poder tratar esto matemáticamente, se ubican en "capas" que se interconectan cada una con la siguiente y con la anterior. Concatenar una capa seguida de otra capa en un número determinado de veces da por resultado una arquitectura denominada Red Neuronal alimentada hacia delante, totalmente conectada (*fully connected feedforward network*).

A las capas intermedias de la red se las denomina capas ocultas. Cuando la cantidad de capas es elevada, surge una primera conceptualización del *Deep Learning*.

Las redes se entrenan para que "aprendan" o para se adapten a un objetivo. Para lograr esto, es necesario utilizar algún mecanismo que permita que, ante un resultado o salida equivocada de la red, se pueda mejorar. Uno de los algoritmos más utilizados se denomina de retropropagación (*backpropagation*). Este algoritmo logra su objetivo mediante una función de pérdida y un optimizador, modificando iterativamente el valor de los pesos de las neuronas. Toma como base la diferencia que se obtuvo entre el valor obtenido por la red en la predicción y el valor esperado por la red. Este resultado se conoce como métrica de error. De esta manera es que las redes se entrenan para aprender a cumplir una tarea mediante una predicción acertada.

En este trabajo se utilizó una red neuronal convolucional, que es especialmente efectiva para tareas que implican visión por computadora, como la clasificación de imágenes, detección de objetos en imágenes, rastreo de objetos en imágenes y segmentación de imágenes, entre otras, y su característica distintiva es la capa convolucional.

En este trabajo de investigación, las imágenes que conforman los videos son a color, ya mencionada la forma en que estas se almacenan y representan. Tanto el alto como el ancho están dados en píxeles. Los canales, en general, son rojo, verde y azul (RGB), por lo cual es necesario considerar esto al aplicar los *filtros convolucionales*, que tienen la misma profundidad.

En estas redes, la cantidad de valores en la capa de entrada está determinada por la intensidad de los píxeles en los diferentes canales. Por ejemplo, una imagen de 128 de alto por 128 de ancho genera una capa de entrada con $128 \times 128 = 16384$ valores, pero como tenemos 3 canales, entonces el total de valores de entrada es de $128 \times 128 \times 3 = 49152$ valores de entrada.

Además, se normalizan los valores que toman las intensidades de los píxeles, que originalmente van de 0 a 255 (8 bits) en valores entre 0 y 1, simplemente dividiéndolos por 255.

En la Fig. 2 se muestra una imagen típica del conjunto de datos que se utilizó en este trabajo.

Fig. 2. Imagen perteneciente al conjunto de datos creado.

A continuación, se explican más profundamente las operaciones involucradas en las redes convolucionales.

A. Capas Convolucionales

Las capas convolucionales son etapas de procesamiento con filtros (también llamados máscaras o *kernels*) presentes en toda el área de la imagen de entrada, con la finalidad de obtener características.

Estos filtros tienen un tamaño establecido por el operador, por ejemplo, de 3×3 o 5×5 píxeles. Se desplazan horizontal y verticalmente por la imagen, efectuando un cálculo similar a la neurona biológica (combinación lineal de las intensidades de la imagen con los pesos del filtro). Este desplazamiento puede efectuarse píxel a píxel o saltear algunos. Este salto se denomina *stride,* el cual, mediante un valor, determina a los filtros cuánto deben moverse horizontal y verticalmente (aunque comúnmente se mantienen ambos desplazamientos). Como se indicó, por cada desplazamiento de los filtros se aplica una suma ponderada entre el filtro y la región de la imagen que abarca el filtro.

Considerar la cantidad de filtros aplicados en cada capa no es fácil de determinar. Si se aplican 10 filtros en una determinada capa para realizar la convolución y extraer características, se tendrá como resultado un tensor de profundidad 10, que será la entrada a la capa siguiente. Los resultados que ofrecen cada uno de estos filtros sobre su aplicación se suman junto a un valor de sesgo o *bias.* Este es un parámetro adicional que tiene la función de agregar un grado de libertad por cada operación de convolución, ya que debe entrenarse junto con los pesos.

Una vez finalizado el paso de los filtros por la imagen, se conforma un nuevo tensor.

Como se mencionó previamente, en la primera capa los filtros obtienen características básicas. Posteriormente, otras convoluciones y otras operaciones determinarán características más abstractas.

B. Función de Activación

Una de las funciones de activación más usadas es la función ReLU (unidad lineal rectificada). El objetivo que tiene esta función es permitir a la red aprender relaciones NO lineales entre las características de la imagen. Matemáticamente, la función tiene la forma:

$$ReLU(x) = max\,(0, x) \qquad (1)$$

Esta función toma la salida de los filtros y devuelve el valor solo si ese valor es positivo; en caso contrario, lo transforma en 0.

Si se da el caso de que un *kernel* no pase el filtro ReLU, este se desactiva y esencialmente no contribuye al resultado final; sin embargo, permite introducir en la red la dinámica de concentrarse en las características más relevantes en la imagen.

Finalmente, ReLU no es la única función utilizada para esta tarea. Existen otras como la función Sigmoide que comprime los valores entre 0 y 1 suavizando las salidas; la función Than que comprime las salidas entre -1 y 1, que también suaviza las salidas y agrega el centro en 0, lo cual puede ser de ayuda en algunas arquitecturas; la función Leaky ReLU similar a la usada, pero permite pendiente de valores negativos; y otras como ELU y Softmax que más adelante veremos.

C. Capas de Agrupamiento o Pooling.

En una red neuronal convolucional, el proceso de convolución aplica filtros a las imágenes para extraer características, creando así nuevas representaciones de estas. Estos filtros generan una gran cantidad de valores (activaciones) en cada ubicación de la imagen, resultando en un número elevado de entradas para la capa siguiente.

Para gestionar esta gran cantidad de datos y reducir la complejidad computacional, se utiliza una técnica llamada *pooling* o agrupamiento. El *pooling* se encarga de reducir el tamaño de las activaciones de la capa de convolución, al mismo tiempo que mantiene las características más relevantes.

El *pooling* tiene dos objetivos principales:

- *Reducir el tamaño del tensor que entrará a la capa siguiente*: al reducir las dimensiones espaciales del mapa de características, se disminuye la carga computacional.
- *Resaltar las características más importantes*: ayuda a preservar las características dominantes, haciendo que la red sea más eficiente y capaz de identificar patrones importantes, independientemente de la variabilidad en el tamaño y la posición de los objetos dentro de las imágenes.

Existen diferentes tipos de capas de agrupamiento, siendo el *max pooling* (agrupamiento por máximo) uno de los más comunes. En el *max pooling*, se toma una ventana de tamaño fijo (por ejemplo, 2×2) y se selecciona el valor máximo de esa ventana. Esto limita la variabilidad y reduce el tamaño del mapa de características, al mismo tiempo que mantiene las activaciones más significativas.

Un ejemplo del proceso de convolución y *pooling*:

- Entrada: imagen color, representada con un tensor de tamaño $128 \times 128 \times 3$ (alto, ancho, cantidad de canales de color).
- Aplicación de filtros: Se aplican 10 filtros de tamaño $5 \times 5 \times 3$ para extraer características. Para que no se pierdan píxeles de borde, pueden rellenarse filas y

columnas con 0, en este caso 2 de ellas.

— Mapa de características: Se obtiene un mapa de características de tamaño $2 \times 2 \times 3$ (alto, ancho, cantidad de filtros).

— Aplicación de *pooling*: Se aplica un agrupamiento con una ventana de tamaño 2×2, que reduce las dimensiones espaciales.

— Salida: El resultado es un mapa de características de tamaño $64 \times 64 \times 10$.

De esta forma, el *pooling* ayuda a hacer que el modelo sea más manejable y eficiente, preservando la información clave mientras se reduce la complejidad computacional (figura 3).

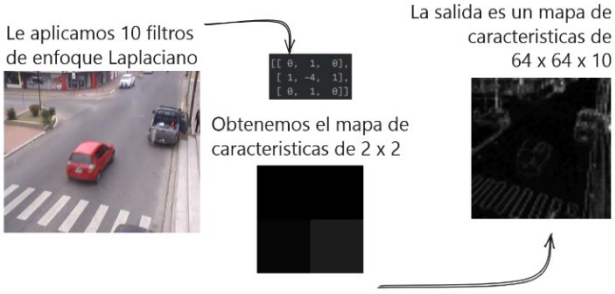

Fig. 3. Representación del ejemplo de convolución y pooling.

A continuación de que se produzca la primera convolución, obtendremos características como bordes, líneas y curvas. A medida que se apliquen más capas de convolución, la red será capaz de reconocer formas más complejas, lo que en conjunto le permitirá a la red aprender y "*ver*".

D. Capas Totalmente Conectadas

Una vez que hemos pasado por varias capas de convolución y *pooling*, el siguiente paso en una red neuronal convolucional es una red *feedforward* de capas totalmente conectadas, que se encuentra al final del modelo.

Las capas totalmente conectadas se utilizan para la clasificación. En estas capas, cada neurona de la capa de salida está conectada a todas las neuronas de la capa anterior. Esto permite que la red realice una combinación lineal de las activaciones recibidas y, generalmente, aplique una función de activación al final.

La principal tarea de las capas totalmente conectadas es transformar el vector de activaciones de la capa anterior en un vector de salida que representa las distintas clases del problema. El número de neuronas en la última capa totalmente conectada corresponde al número de clases en el conjunto de datos. Por ejemplo, si hay 10 clases, la última capa tendrá 10 neuronas.

En resumen, las capas totalmente conectadas toman un vector unidimensional resultante de las capas de convolución y *pooling*, convirtiendo el último tensor en un vector unidimensional, y lo convierten en un vector de salida cuyos valores serán más altos para las clases más probables.

En tareas de clasificación, se utiliza al final la función *softmax* para convertir los valores de salida de la red en un vector de probabilidades, donde cada probabilidad indica la certeza de que la entrada pertenece a una clase específica, y todas las probabilidades suman 1.

La función *softmax* garantiza que las probabilidades estén normalizadas, permitiendo que incluso las clases con menor probabilidad sean consideradas. Esto evita descartar clases basándose solo en valores bajos.

E. Función de Pérdida

La función de pérdida calcula un valor que mide la diferencia entre las predicciones de la red neuronal y las etiquetas reales. Su objetivo es guiar el proceso de entrenamiento ajustando los pesos de la red.

En la función de pérdida se calcula el gradiente, que indica cómo ajustar los pesos de la red. Si el gradiente es cero, como puede suceder con activaciones negativas en funciones ReLU, algunos pesos no se van a actualizar. La modificación de los pesos será tanto mayor cuanto más importante sea el error obtenido

Las probabilidades generadas por *softmax* están, por supuesto, en el rango de 0 a 1 y reflejan la certeza de la red en sus predicciones. La ecuación utilizada es:

$$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \qquad (2)$$

Al final de esta función, la última capa de la red se corresponde con las 119 clases de vehículos etiquetadas en este trabajo, para su detección por marca y modelo, como se describirá más adelante.

Cada valor de salida predicho por la red estará entre 0 y 1, donde el valor más alto indica la clase más probable.

En nuestro modelo, la última capa considera 119 posibles clases correspondientes a marcas y modelos de vehículos.

Las salidas deseadas se representan mediante una técnica denominada *one-hot encoding*. En esta técnica, solo un elemento del vector deseado en la salida estará "caliente", es decir, tomará el valor 1, mientras que todos los demás tomarán el valor 0. La finalidad de esto es permitir que el modelo aprenda a predecir la probabilidad de cada clase.

Como ejemplo, supongamos que la predicción de la red al

pasar por la función *softmax* da como resultado un vector de la forma $[c_1\ c_2\ \cdots\ c_{119}]$ donde $c_{17} = 0.30$, $c_{38} = 0.45$, $c_{70} = 0.25$ y los restantes valores son 0.

Para calcular las probabilidades, usamos la función *softmax*. Primero, calculamos los valores exponenciales para los valores no-cero, $e^{0.30}$, $e^{0.45}$ y $e^{0.25}$.

Las probabilidades para cada clase se obtienen dividiendo los valores exponenciales de cada clase entre la suma total:

Para c_{17}: 0.32 o 32%

Para c_{38}: 0.37 o 37%

Para c_{70}: 0.30 o 30%

Como resultado de esta evaluación de probabilidades, la clase c_{38} es el candidato como respuesta a la detección de la red. Sin embargo, este caso en particular muestra cómo, a pesar de tomar el máximo, las otras dos clases siguientes deberían ser tenidas en cuenta.

III. MODELO YOLO

Se optó por la familia de modelos YOLO [2], disponibles en código abierto. En la actualidad, se encuentra disponible la versión 10 de YOLO. Su autor original, Joseph Redmon, dejó de participar activamente en el desarrollo después de la versión 3. Desde entonces, el modelo ha experimentado numerosas mejoras implementadas por otros desarrolladores y ha evolucionado para satisfacer diversas necesidades de la comunidad, tanto en aplicaciones comerciales como industriales. Tomando como referencia las características que los distintos modelos ofrecen y luego de evaluarlos, se considera la versión 5 como la más adecuada para este trabajo, debido a su estabilidad y prestaciones.

A. You Only Look Once – v5

La versión 5 de YOLO ofrece una amplia variedad de modelos, desde compactos como YOLOv5n (*nano*) hasta más grandes como YOLOv5x (*extra large*), adecuados para imágenes de entrada de 640 píxeles. También están disponibles variantes como YOLOv5n6 y YOLOv5x6, que manejan imágenes de entrada de hasta 1280 píxeles y, en caso de utilizar Test Time Augmentation (TTA), pueden llegar hasta 1536 píxeles. La elección del modelo adecuado no es tan simple como seleccionar el de mejor rendimiento; es fundamental considerar las limitaciones de *hardware*, ya que este factor determina significativamente la viabilidad del modelo elegido.

En este trabajo se optó por utilizar el modelo YOLOv5m (*medium*). Esta elección es el resultado de una evaluación meticulosa de los modelos YOLOv5n y YOLOv5s, que mostraron rendimientos insuficientes para los objetivos requeridos, y del modelo YOLOv5l (*large*), que no pudo ser implementado satisfactoriamente con el *hardware* disponible. Esta elección tuvo implicaciones en la configuración del entrenamiento, ya que se decidió usar 16 lotes (*batches*) con el objetivo de mejorar la generalización mediante actualizaciones más frecuentes, y 100 épocas (*epochs*) para permitir un entrenamiento más prolongado y un mejor ajuste de los pesos del modelo.

B. Funcionamiento de YOLO

La función principal de este modelo es la detección de una sola pasada [4]. Para lograr esto, el modelo utiliza una cuadrícula de dimensiones $s \times s$. Si el centro del objeto de interés se encuentra dentro de alguna de las celdas de la cuadrícula, esa celda es responsable de detectar el objeto. Cada celda en la cuadrícula predice los cuadros delimitadores, conocidos como *bounding boxes*, que incluyen las dimensiones del cuadro y una puntuación de confianza. La ecuación general para realizar esta tarea es:

$$puntuacion\ de\ confianza = p(objeto) * IoU\frac{verdad}{prediccion} \tag{3}$$

Cuanto más alto es el valor resultante, mayor es la confianza del modelo en que el objeto detectado se encuentra dentro del *bounding box*. Sin embargo, al realizar esta tarea, se generan múltiples *bounding boxes*, lo que puede llevar a una sobreabundancia de detecciones en la imagen. Para abordar este problema, YOLO utiliza el concepto de *supresión de no máximo (NMS, por sus siglas en inglés)* [5] . Este proceso emplea un valor de umbral (*threshold*) que, dependiendo de su valor, elimina cuadros delimitadores que no contienen detecciones relevantes. Todo esto se gestiona a través de la función de pérdida.

En la Fig. 3 se observa un esquema del procesamiento, extraído del trabajo original que presenta este algoritmo [1].

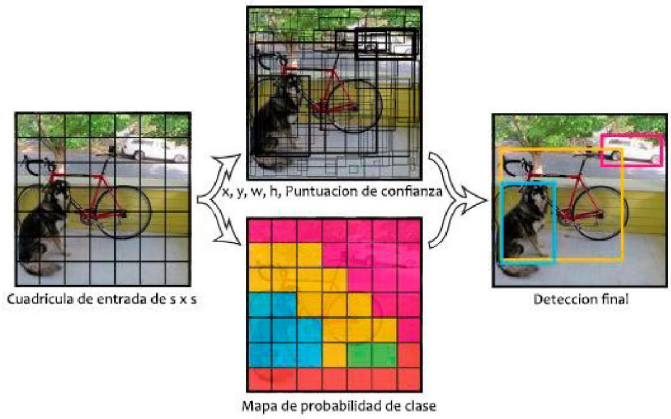

Fig. 3. Proceso de detección de YOLO, extraído de [2].

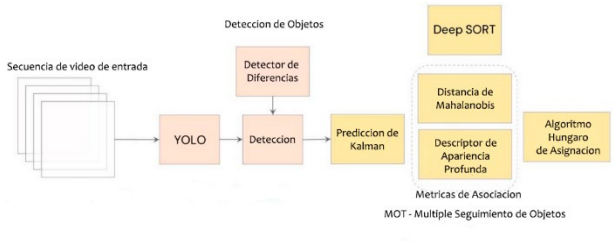

Fig. 4. Representación del algoritmo DeepSORT.

IV. Modelo DeepSORT

Se estudiaron algoritmos de rastreo de objetos, eligiendo DeepSORT por su facilidad de implementación y su capacidad para realizar rastreo de múltiples objetos (MOT). Este algoritmo cumple con el requisito de la investigación para el rastreo de vehículos y facilita la identificación al asignar un identificador único a cada detección.

Históricamente, DeepSORT fue desarrollado por Nicola Wojke, Alex Bewley y Dietrich Paulus en 2017 [3], con el objetivo de mejorar el rastreo de objetos manteniendo el ID (identificador único) durante períodos prolongados, incluso en situaciones de oclusión. Para lograr esto, los autores basaron su trabajo en el algoritmo SORT [6], al cual añadieron una métrica de asociación profunda que combina información de movimiento con información de apariencia. Los autores indicaron que aplicaron una red neuronal convolucional entrenada para distinguir peatones, lo que aumentó la robustez del sistema frente a fallos y oclusiones.

A. Desafío del algoritmo

DeepSORT amplía las funcionalidades del algoritmo SORT, como ya fue explicado con anterioridad, con el agregado de un parámetro más de asociación profunda que permite rastrear objetos que dejen de ser visibles por un período de tiempo. Sin embargo, al igual que su predecesor, enfrenta el desafío principal de lograr la reidentificación correcta cuando, en una detección obtenida, por motivos de obstáculos, oclusiones o cambios en la dirección del objeto, se genera la pérdida momentánea de la identificación lograda. Lo que se busca con DeepSORT es que, a través del parámetro de asociación profunda, el objeto reidentificado y previamente detectado sea el mismo a lo largo de todo el video.

En este trabajo, en particular, la necesidad consiste en asociar un rastreo a un único vehículo a medida que se desplaza por el campo de visión de la cámara. Este movimiento puede ser lineal o no, lo que puede provocar cambios en la forma del vehículo, ya sea debido a la velocidad, cambios en su dirección o la obstrucción causada por la superposición de otros objetos, entre otros factores. Por lo tanto, es crucial prestar atención a las características utilizadas para el rastreo del vehículo.

B. Repositorio Elegido

Después de analizar el repositorio proporcionado por los autores de DeepSORT y comprender que para lograr las capacidades indicadas utilizaron una red neuronal convolucional entrenada *offline* con un conjunto de datos de peatones, se determinó que era necesario seguir un enfoque similar.

Se creó una nueva red de arquitectura específica para vehículos. Sin embargo, debido a restricciones de tiempo y las métricas resultantes, se optó por utilizar un repositorio ya desarrollado para la detección de peatones y vehículos [7], sobre el cual se realizaron algunas modificaciones a este repositorio para cumplir con los requisitos iniciales del trabajo.

V. Modelo *AD HOC*

Se tomaron los videos de cámaras de vigilancia ciudadana distribuidas en la ciudad de San Luis. Cada uno de estos videos cuenta con un promedio de 2 horas de grabación.

La primera tarea consistió en la extracción y procesamiento de las imágenes contenidas en los videos, es decir, en cada uno de sus *frames* que contengan la presencia de vehículos automotores plausibles de ser etiquetados bajo la premisa de "automotor presente". Para ello utilizamos el software gratuito disponible *VLC Media Player* [8] con el cual se extrajeron los *frames* de los videos, para luego almacenarlos en formato *Joint Photographic Experts Group (JPG).*

De esta manera, se obtuvieron 1 483 053 imágenes. Este número es muy elevado para la capacidad humana disponible de procesarlas a todas de forma manual, por lo que se tomó la decisión de realizar recortes en los videos originales. Utilizando la misma herramienta *VLC* se logró extraer porciones de videos

que contenían la premisa impuesta y descartar espacios de tiempo donde no se observan o se consideran innecesarios, atento a las características observadas.

Como resultado, se obtuvieron 24 recortes de video, a los cuales se les extrajeron las imágenes contenidas, para luego analizar cada uno de los *frames*, en busca de aquellos donde se pudieron observar con detalle las características de los vehículos representadas por su marca, modelo y elemento distintivo. Se procuraron distintos ángulos de captura, es decir, obteniendo distintas imágenes del mismo vehículo en posiciones espaciales diferentes sobre una misma cámara y en cámaras distintas, teniendo especial atención a los diferentes escenarios ambientales y circunstancias viales tales como bloqueos parciales o totales que los vehículos puedan sortear al desplazarse por el campo de visión que presenta el lente de la cámara. Toda esta tarea se realizó de forma manual y visual, lo que arrojó como resultado, luego de procesar las imágenes, un conjunto de 1 769 imágenes que contienen las características buscadas en los vehículos que se consideran necesarias para su utilización en el entrenamiento del modelo.

Con las imágenes procesadas se realizó el etiquetado utilizando los datos de las diferentes marcas y modelos que se pueden observar en los vehículos de la vía pública en cualquier provincia argentina.

Para etiquetar se utilizó el programa LabelImg [9], escrito en Python, el cual, mediante una interfaz gráfica creada con la librería PyQt, permite al operador llevar adelante esta tarea, que se puede observar en la Fig. 5.

Fig. 5. Interfaz gráfica de LabelImg.

El resultado de la actividad de etiquetado fue la obtención de un archivo generado por el programa, que contiene 119 etiquetas de vehículos, cuyos nombres fueron dados por el investigador con la configuración de marca y modelo, por, ejemplo: *Peugeot_Partner*.

Para el caso particular de las bicicletas y motos, se optó por generalizar la etiqueta con la salvedad de identificar si el ciclista o motociclista llevaba puesto o no el casco, por ejemplo: *Motociclista_conCasco*, *Ciclista_sinCasco*. Además, se obtuvo cada uno de los archivos de imágenes con su respectivo archivo de texto que contiene las coordenadas etiquetadas dentro de las imágenes donde se identificaron vehículos.

A. Entrenamiento del Modelo

Del total de las imágenes se tomó un grupo conformado por el 75 % del total para el entrenamiento y un segundo grupo conformado por el 25 % para la validación y testeo.

A continuación, se creó el archivo "*dataset.yaml*", mediante el cual se trata y describe la información de entrenamiento y de prueba, estableciendo una estructura mediante rutas, números de clase y nombres de clase. El archivo creado en esta investigación contiene los siguientes datos:

- **train**: dataset/images/train es la ruta a las imágenes de entrenamiento.
- **val**: dataset/images/val es la ruta a las imágenes de validación.
- **nc**: 118 es el número de clases, que representa las 119 etiquetas al iniciar en 0.
- **names**: contiene todas las etiquetas construidas con marcas y modelos de vehículos.

B. Hardware y Software

Para entrenar y validar el modelo se utilizó *hardware* propio. Se utilizó una PC de escritorio con microprocesador Intel i7-4790k, con 8 GB de memoria RAM y una tarjeta gráfica NVIDIA GTX970 de 4 GB para procesamiento.

Como base de trabajo se utilizó el software *Visual Studio Code*, haciendo uso de un entorno virtual y todas las librerías necesarias para hacer de *Pytorch* nuestro marco de trabajo y generación de las operaciones necesarias para lograr obtener el modelo objetivo.

C. Métricas obtenidas

Se analizaron las métricas representativas que ofrecen las curvas F1 de entrenamiento (Fig. 6) y validación (Fig. 7) del modelo, además de la matriz de confusión (Fig. 8), la curva P (Fig. 9), la curva PR y la curva R, las cuales en su conjunto ofrecen una valoración sobre el desempeño del modelo.

Se precisa mostrar los gráficos tanto de la curva F1 del conjunto de entrenamiento como de la curva F1 del conjunto de validación, que en este estudio tuvieron solo una milésima de diferencia en el valor de confianza. Ya que nos permite evaluar el rendimiento, la estabilidad y la capacidad de generalización del modelo, como resultado, el modelo generaliza bien con un rendimiento pobre, pero con una buena

estabilidad.

De la matriz de confusión (Fig. 8) se determinó que las clases "*Toyota_Hilux*", "*Colectivo*", "*Fiat_Uno*", "*Peugeot_207*" y "*Fiat_Qubo*" fueron las clases en las que el modelo realizó mejores predicciones. Se observaron mayores errores de predicción en la mayoría de las clases restantes, siendo las clases más afectadas aquellas en las que se contaba con menor cantidad de imágenes representativas.

De la curva P se logró concluir que para que una predicción sea confiable en las detecciones de verdaderos positivos, la probabilidad tenía que ser de 0.954, lo cual es claramente muy exigente.

La curva PR informó que el modelo es de bajo rendimiento en términos relativos, con solo un 18.6 % de precisión al 0.5 de umbral IoU. Finalmente, la curva R informó que el modelo es capaz de recuperar el 77 % de las detecciones positivas reales con 0.0 de confianza y disminuye significativamente a medida que se requiere más confianza, resultando que a 0.5 de confianza la recuperación de detecciones positivas reales es alrededor del 15 %. Como síntesis de las métricas obtenidas, se puede inferir que el modelo requiere como mínimo la ampliación del *dataset* utilizado, que contenga más vehículos del mismo tipo para que el modelo logre mejorar los parámetros. Por ejemplo, el doble de imágenes etiquetadas que contengan "*Toyota_Hilux*", "*Colectivo*", "*Fiat_Uno*", "*Peugeot_207*", "*Fiat_Qubo*" y el resto de vehículos etiquetados en el *dataset*, teóricamente debería incrementar los valores métricos a una medida razonable para un modelo pequeño de detección.

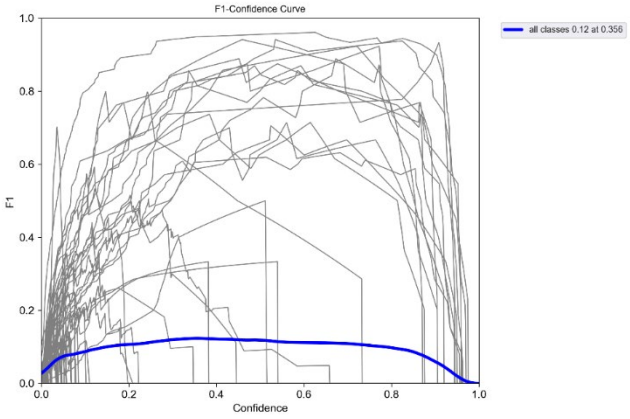

Fig. 6. Curvas F1 de Entrenamiento.

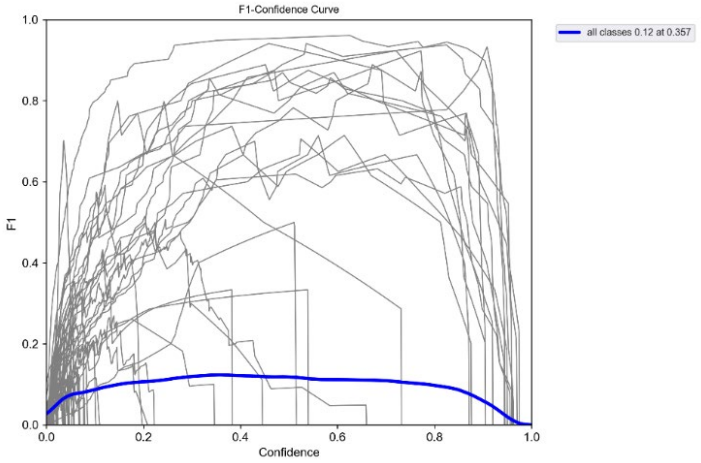

Fig. 7. Curva F1 de Validación.

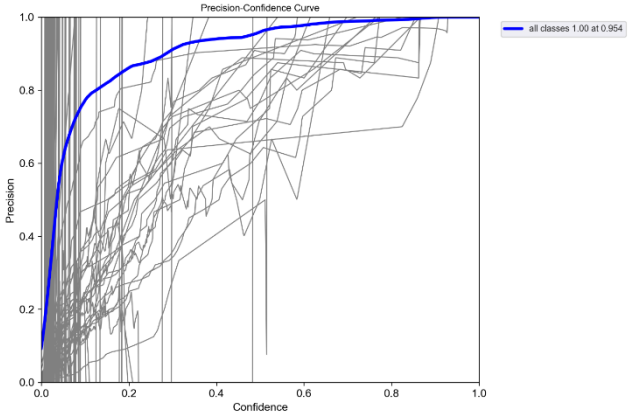

Fig. 8. Matriz de Confusión.

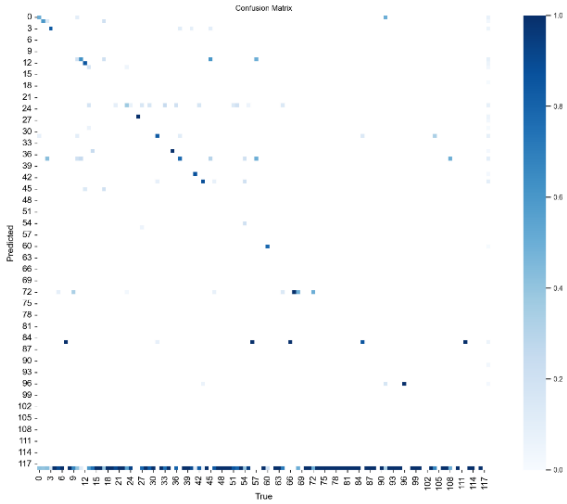

Fig. 9. Curva P.

Una vez que contamos con el modelo de detección, se realizó la reidentificación de vehículos para poder hacer uso del rastreo proporcionado por DeepSORT. Para la reidentificacion fue necesario entrenar un modelo que permita lograr el rastreo de vehículos, utilizando una sucesión mayor de imágenes del mismo vehículo en distintas maniobras y movimientos. Si bien esto no se incluyó en el trabajo final de carrera a partir del cual se inció esta investigación, sí se culminó a posteriori. Para ello, lo que se realizó fue, en primer lugar, tomar videos de las cámaras con las que se contaba y de ellas extraer los *frames* donde se observaba a cada uno de los vehículos etiquetados realizando una maniobra o movimiento, por ejemplo, como se observa en la figura 9.

Una vez que se contó con cada *frame* para cada vehículo, se los etiquetó, obteniendo tanto el ID como las coordenadas de su alto, ancho y centro. Con toda esta información ya clasificada y agrupada en carpetas, se conformó el *dataset* asociado a este modelo; luego se modificó la red neuronal convolucional CNN utilizando un bloque residual y una arquitectura de redbasada en las especificaciones indicadas en el paper oficial de DeepSORT [3] para la reidentificación, como se puede ver en las figuras 10, 11 y 12.

Esta arquitectura permitió crear el modelo entrenado que luego se usó para la reidentificación de vehículos y se concluyó esta etapa.

Fig. 9. La zona de color representa la superposición de frames del mismo vehículo.

```python
# Bloque residual
def residual_block(x, filters, kernel_size=3, stride=1):
    shortcut = x  # Guardamos la entrada original
    x = layers.Conv2D(filters, kernel_size, strides=stride, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Conv2D(filters, kernel_size, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)

    if stride != 1:
        shortcut = layers.Conv2D(filters, (1, 1), strides=stride, padding='same')(shortcut)
    x = layers.add([x, shortcut])  # Suma con el atajo (skip connection)
    return x
```

```python
# Arquitectura CNN basada en DeepSORT
def create_deepsort_cnn(input_shape=(128, 64, 3)):
    inputs = tf.keras.Input(shape=input_shape)

    # Conv 1
    x = layers.Conv2D(32, (3, 3), padding='same', strides=1, activation='relu')(inputs)
    x = layers.BatchNormalization()(x)

    # Conv 2
    x = layers.Conv2D(32, (3, 3), padding='same', strides=1, activation='relu')(x)
    x = layers.BatchNormalization()(x)

    # Max Pool 3
    x = layers.MaxPooling2D((2, 2))(x)  # Reduce a 32 x 64 x 32

    # Residual Bloque 4
    x = residual_block(x, 32)

    # Residual Bloque 5
    x = residual_block(x, 32)

    # Residual Bloque 6 (stride=2, reduce la resolución)
    x = residual_block(x, 64, stride=2)

    # Residual Bloque 7
    x = residual_block(x, 64)

    # Residual Bloque 8 (stride=2, reduce la resolución)
    x = residual_block(x, 128, stride=2)

    # Residual Bloque 9
    x = residual_block(x, 128)
```

```python
# Flatten y capa densa
x = layers.Flatten()(x)
x = layers.Dense(128)(x)  # Proyección a 128 dimensiones

# Proyección en la hiperesfera, especificando la forma de salida
outputs = layers.Lambda(lambda y: tf.math.l2_normalize(y, axis=1), output_shape=(128,))(x)

# Modelo
model = tf.keras.Model(inputs=inputs, outputs=outputs)
return model
```

Fig. 10, 11 y 12. Arquitectura de la red de reidentificacion, basada en el paper de DeepSORT.

Con el fin de disminuir los tiempos de trabajo en el desarrollo de todos los aspectos necesarios para asociar YOLO con DeepSORT, se integró en todos los aspectos tanto el modelo de detección desarrollado como el modelo de reidentificacion, tomando como base un repositorio GitHub, que ya contaba con dicha integración de YOLOv5 y *DeepSORT* [10] A este repositorio se le realizaron modificaciones para que considere el modelo entrenado *ad hoc* y se modificó para que grafique las *bounding boxes* con el nombre de los vehículos por marca, modelo y nivel de confianza de la detección. Se procesaron los videos de prueba, obteniendo como resultado nuevos videos con los vehículos rastreados. En las Fig. 13, Fig. 14 y Fig. 15 se muestran ejemplos de los lotes de entrenamiento y resultados obtenidos.

VI. Discusión y conclusiones

En este trabajo se llevó a cabo el diseño e implementación de un modelo de aprendizaje automático que detecta y rastrea vehículos. Uno de sus objetivos es que se tome como base para el desarrollo de mejoras o para la creación de nuevos modelos de detección y rastreo de vehículos en la vía pública mediante cámaras de seguridad, tanto públicas como privadas; otro objetivo buscado es que se optimice el uso de recursos humanos, automatizando tareas de búsqueda y rastreo de vehículos en imágenes de video. Donde el error humano por exceso de visualización, distracciones, cansancio, etc. produzca errores de identificación y visualización en la búsqueda de vehículos.

Habiendo planteado los objetivos, se logró construir y evaluar un modelo de aprendizaje automático cuyo propósito es detectar y rastrear vehículos que habitualmente transitan por calles y rutas de la República Argentina. Durante el transcurso del proceso de investigación se encontraron diversas dificultades técnicas que se describen a continuación con el fin de que el lector pueda aprovechar el conocimiento adquirido y mejorarlo.

Al momento de seleccionar la herramienta de procesamiento para entrenar el modelo y posteriormente validarlo, se consideró utilizar *Google Colab*, un servicio que puede entrenar el modelo mediante un notebook en la nube. Esta herramienta ofrece más de 50 horas de tiempo de ejecución, 12 GB de memoria RAM y más de 100 GB de espacio en disco de forma gratuita; sin embargo, también advierte que esto "no está garantizado". Esta última afirmación condujo a una prueba en la que se tomó un video obtenido de internet y ajeno a esta investigación para procesarlo y utilizarlo. Al transcurrir el tiempo y ejecutar el proceso, se encontraron fallas en los tiempos de ejecución, disminuciones en la cantidad de memoria y cortes inesperados debido a la conexión de red. Estas circunstancias, unidas al hecho de tener que subir la información a la nube, fueron los elementos decisivos para descartar el uso de esta herramienta y de otras similares como *AWS*, *Azure* o *Kaggle*.

Realizar el entrenamiento mediante una computadora de escritorio permitió mantener el control total del proceso desarrollado. Esta elección se considera correcta especialmente en investigaciones judiciales, ya que no es recomendable subir videos que puedan contener elementos de interés pericial a servicios de terceros en la red.

La implementación del modelo y su evaluación permitieron comprender que, con el hardware disponible y habiendo evaluado todas las métricas, el *hardware* resulta insuficiente para obtener un modelo que cuente, por ejemplo, con un alto valor de equilibrio entre precisión y recuperación de detecciones. Se recomienda, entonces, contar con un *hardware* mínimo de 64 GB de RAM, una GPU RTX 3090, un microprocesador Ryzen 9 o i9 y suficiente espacio en disco, dado que los videos son datos de gran volumen, con la finalidad de obtener resultados que sean extrapolables a la realidad cotidiana con el efecto deseado. Una mayor cantidad de memoria RAM permitiría no solo disminuir los tiempos de procesamiento, sino manejar conjuntos de datos más grandes e incluso mejorar la arquitectura que el paper original del modelo utilizado indica. De forma práctica podríamos pasar de 16 lotes (*batches*), como se utilizó en este trabajo, a muchos más con el fin de optimizar la generalización del modelo; incluso podríamos aumentar las épocas (*epochs*) de las 100 elegidas al doble o más, perfeccionando de esta manera el ajuste de los pesos de la red. Por otro lado, la GPU propuesta cuenta con más de 10000 núcleos CUDA, lo que permite más cálculos en paralelo, acelerando los tiempos de entrenamiento; también cuenta con núcleos de tensores, unidades de procesamiento altamente eficaces en multiplicación de matrices, soporte para CUDA y cuDNN. En resumen, no solo reducirían el tiempo de entrenamiento, sino que también se podrían usar modelos más grandes y complejos, entre otros beneficios. Finalmente, se recomienda el microprocesador Ryzen 9 o i9 de Intel porque tendríamos muchos más núcleos e hilos de procesamiento, obteniendo más cálculos en paralelo y, al tener frecuencias de reloj más altas, ejecutando el código más rápido, entre otros muchos beneficios.

Por otro lado, mediante las evaluaciones sobre los resultados arrojados por la matriz de confusión, se logró establecer que ciertas etiquetas, como *Toyota_Hilux*, Colectivo, *Fiat_Uno*, *Peugeot_207* y *Fiat_Qubo*, presentan más predicciones correctas e incorrectas. Esto se debe a dos componentes fundamentales: el primero es el número de datos de entrenamiento, es decir, la cantidad de vehículos etiquetados de la misma marca y modelo; y el segundo, como sucede en el caso las etiquetas *Colectivo* y *Fiat_Qubo*, ocurre debido a la forma característica (física) de estos vehículos, que les permite diferenciarse del resto de los evaluados por el modelo de forma significativa, conjuntamente con la particularidad de que, al momento del entrenamiento, la cantidad de vehículos de estas características etiquetados era el de más bajo en el conjunto considerado.

Como se observó en las métricas reportadas y en las imágenes de ejemplo de los videos resultantes que fueron plasmadas con anterioridad, el modelo presenta un bajo nivel de detección y rastreo; sin embargo, una vez que el vehículo es detectado y rastreado, el modelo mantiene el seguimiento en diversas condiciones ambientales y con obstrucciones en los lentes de las cámaras a lo largo de los cuadros del video. Esto es un indicador suficiente de que el proyecto es viable si se aumenta el número de datos de entrenamiento, se incrementa

el tamaño de las conexiones entre neuronas del modelo YOLO elegido y se emplea un *hardware* lo suficientemente robusto para entrenarlo y validar sus resultados.

Finalmente, se considera que el conjunto de datos creado para esta investigación es un elemento valioso para la comunidad científica, por lo cual se pretende robustecerlo, ampliando el número de clases y etiquetas, para luego ponerlo a disposición de la comunidad, brindando un elemento útil y de interés para investigadores.

Fig. 13. Lote (batch) de entrenamiento.

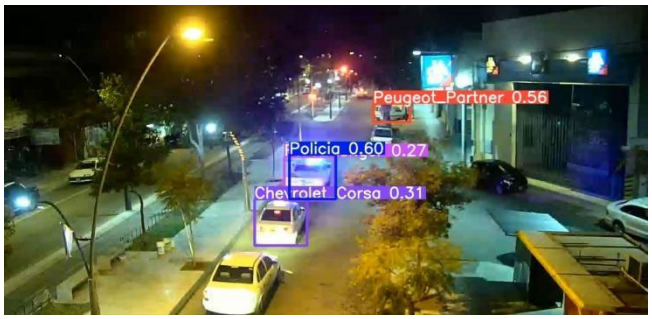

Fig. 14. Detecciones, rastreos y nivel de confianza.

Fig. 15. Detalle de detección, rastreos y nivel de confianza.

REFERENCIAS

[1] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," en *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779–788. [En línea]. Disponible: https://arxiv.org/abs/1506.02640

[2] Tesla, "Inteligencia artificial y Autopilot," 2024. [En línea]. Disponible: https://www.tesla.com/es_cl/AI

[3] N. Wojke, A. Bewley y D. Paulus, "Simple Online and Realtime Tracking with a Deep Association Metric," en *2017 IEEE International Conference on Image Processing (ICIP)*, Beijing, China, 2017, pp. 3645–3649. [En línea]. Disponible: https://arxiv.org/abs/1703.07402

[4] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, real-time object detection (arXiv:1506.02640). arXiv. https://doi.org/10.48550/arXiv.1506.02640

[5] A. Rosebrock, "Non-Maximum Suppression for Object Detection in Python", *pyimagesearch*, 2024. [En línea] Disponible: https://pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/

[6] Ultralytics, "YOLOv5 Documentation," 2024. [En línea]. Disponible: https://docs.ultralytics.com/yolov5/

[7] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958. [En línea]. Disponible: https://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf

[8] VLC Media Player, VideoLAN Organization. [En línea] Disponible: https://www.videolan.org/vlc/index.es.html

[9] LabelImg, GitHub. [En línea] Disponible: https://github.com/HumanSignal/labelImg

[10] H. Ma, "DeepSORT_YOLOv5_Pytorch," Repositorio GitHub. [En línea]. Disponible: https://github.com/HowieMa/DeepSORT_YOLOv5_Pytorch

[11] A. Bewley, Z. Ge, L. Ott, F. Ramos y B. Upcroft, "Simple

online and realtime tracking," en *2016 IEEE International Conference on Image Processing (ICIP)*, Phoenix, AZ, USA, 2016, pp. 3464–3468.

[12] M. Hussain, "YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection," *Machines*, vol. 11, no. 7, p. 677, 2023. [En línea]. Disponible: https://doi.org/10.3390/machines11070677

13] IBM, "¿Qué son las redes neuronales convolucionales?," 2024. [En línea]. Disponible: https://www.ibm.com/es-es/topics/convolutional-neural-networks

[14] X. Zhao, L. Wang, Y. Zhang *et al.*, "A review of convolutional neural networks in computer vision," *Artificial Intelligence Review*, vol. 57, 2024. [En línea]. Disponible: https://doi.org/10.1007/s10462-024-10721-6

[15] A. M. Turing, "Computing Machinery and Intelligence," *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950. [En línea]. Disponible: https://doi.org/10.1093/mind/LIX.236.433

[16] J. McCarthy, M. L. Minsky, N. Rochester y C. E. Shannon, "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence," 1955. [En línea]. Disponible: http://jmc.stanford.edu/articles/dartmouth/dartmouth.pdf

[17] J. Moor, "The Dartmouth College Artificial Intelligence Conference: The Next Fifty Years," *AI Magazine*, vol. 27, no. 4, pp. 87–91, 2006. [En línea]. Disponible: https://www.researchgate.net/publication/220605256_The_Dartmouth_College_Artificial_Intelligence_Conference_The_Next_Fifty_Years

[18] S. Ioffe y C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," en *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, Lille, Francia, 2015, pp. 448–456. [En línea]. Disponible: https://arxiv.org/abs/1502.03167

[19] V. Rastogi, "Max Pooling Layer In Computer Vision," *Medium*, 2023. [En línea]. Disponible: https://medium.com/@vaibhav1403/max-pooling-layer-in-computer-vision-8cccfd91c521

[20] W. S. McCulloch y W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943. [En línea]. Disponible: https://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf