UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Inteligencia computacional para el análisis de datos de tráfico y aprendizaje de estimadores

Juan Serra, Hernán Winter

Programa de Grado en Ingeniería en Computación
Facultad de Ingeniería
Universidad de la República

–

Marzo de 2020

RESUMEN

Este trabajo presenta el diseño e implementación de un sistema de recolección y análisis de datos a partir de grabaciones de cámaras de tráfico. El principal objetivo de este sistema es proveer un medio para la obtención de datos Y para asistir o automatizar tareas visuales de monitorización de situaciones relevantes de tráfico como infracciones, manejo imprudente o embotellamientos. El sistema propuesto se basa en técnicas de inteligencia computacional y visión por computadora, incluyendo redes neuronales convolucionales Y algoritmos de seguimiento de objetos múltiples. Como resultado del análisis del estado del arte realizado, se optó por utilizar la biblioteca Detectron2 para resolver la detección de objetos y la biblioteca Node-Moving-Things-Tracker para el seguimiento de objetos. La arquitectura del sistema propuesto consiste en un pipeline modular de procesamiento de video, donde cada paso es un módulo funcional independiente, configurable y débilmente acoplado a los demás. La validación del sistema fue realizada utilizando grabaciones de tráfico de la ciudad de Montevideo, procesando diferentes escenarios, en diferentes condiciones de iluminación y con distintas calidades de video. Los resultados muestran, por un lado, la efectividad del sistema en los escenarios que se presentan durante el día, mostrando una exactitud media del 70 % en la detección, el 85 % en la evaluación de la clasificación de detecciones y una puntuación media del 85 % en la métrica MOTA para la evaluación del seguimiento. La evaluación de los módulos de análisis de datos mostraron resultados de similar precisión media con respecto a la detección y el seguimiento. Por otro lado, existe la necesidad de continuar el trabajo para aquellos escenarios que presentan condiciones de poca iluminación.

Palabras claves:
inteligencia computacional, redes neuronales, datos de tráfico, ciudades inteligentes.

Lista de figuras

Lista de tablas

Tabla de contenidos

Capítulo 1

Introducción

El crecimiento acelerado de la población urbana y las ciudades ha generado varios problemas asociados al transporte como, por ejemplo, el aumento de distancia entre lugares, el incremento del tráfico vehicular y la poca adecuación de la infraestructura vial al ritmo del tráfico actual. Estos factores generan escaso cumplimiento de las normas de tránsito y la ausencia de mecanismos de control que supervisen de manera efectiva el comportamiento de los usuarios de las vías de tránsito (Cardozo y Rey, 2007). La urbanización también ha traído consigo la incorporación de tecnologías de información y comunicación a la infraestructura de las ciudades. Los sistemas de transporte y tráfico urbano son generalmente abordados bajo el paradigma de las ciudades inteligentes en lo referente a movilidad inteligente (Benevolo *et al.*, 2016). Relacionado con este concepto se encuentran los Sistemas Inteligentes de Transporte (SIT), los cuales hacen uso de la tecnología para desarrollar y mejorar la movilidad en las ciudades de manera que los ciudadanos puedan trasladarse de forma segura reduciendo problemas como accidentes y tiempos de viaje. A su vez, los SIT permiten recolectar grandes volúmenes de datos urbanos (Figueiredo *et al.*, 2001). Extraer conocimiento de estos datos recopilados es fundamental para mejorar los procesos de toma de decisiones en las ciudades, así como también mejorar la calidad y la eficiencia de los servicios públicos de transporte Massobrio y Nesmachnow (2020).

Varios métodos se han utilizado para la recolección de datos de tráfico, incluyendo el uso de diferentes tipos de sensores, video vigilancia, vigilancia aérea (incluyendo vehículos aéreos no tripulados), policía de tránsito, información de viajeros, entre otros. Incluso los datos pueden ser obtenidos indirectamente a

través de sistemas de gestión de emergencias que almacenan datos de accidentes. La obtención y análisis de los datos generados por esta clase de sistemas pueden ser utilizados para reducir los tiempos de respuestas y así disminuir o incluso prevenir los embotellamientos en el tráfico (Dagaeva *et al.*, 2019). Entre todos estos métodos de recopilación de datos de tráfico, el procesamiento de video se ha convertido en uno de los más convenientes y de menor costo, ya que su implementación no requiere del despliegue de sistemas o equipamiento costoso con el único propósito de detectar vehículos. Las ciudades modernas ya cuentan con una gran cantidad de cámaras y agregar nuevas cámaras requiere poca modificación de la infraestructura vial (Makhmutova *et al.*, 2020).

Tareas como el recuento de vehículos y la detección de infracciones son de gran importancia para los SIT (Yang y Pun-Cheng, 2018). Recientemente, los algoritmos de conteo y detección basados en visión por computadora (Yang y Qu, 2018) han demostrado ser más efectivos y superar a los métodos tradicionales de vigilancia del tráfico, como los métodos que utilizan diferentes tipos de sensores (Lou *et al.*, 2019). Por otro lado, los métodos basados en visión por computadora y aprendizaje computacional pueden ser utilizados en el procesamiento de videos para detectar distintas características del tráfico automáticamente. Esto permite reducir el número de funcionarios necesarios para monitorizar las situaciones en el tráfico, permitiendo el uso del personal humano en tareas más importantes como la toma de decisiones.

Sin embargo, todavía existen muchos desafíos y problemas abiertos en los procesos de conteo y detección de vehículos basados en visión por computadora. En primer lugar se encuentra el costo computacional de estos algoritmos, lo cual dificulta la implementación de sistemas embebidos o que puedan realizar procesamiento de videos en tiempo real. También están los problemas causados por condiciones externas que afectan las imágenes capturadas por las cámaras de vigilancia agregando ruido. Este tipo de factores incluye variaciones de iluminación y condiciones climáticas como lluvia o niebla. Adicionalmente, un problema que afecta fuertemente el desempeño de estos algoritmos es la oclusión parcial o total de los objetos. Este suceso es común en el tráfico ya que los vehículos pueden ser ocluidos por otros vehículos o por distintos elementos de la infraestructura vial.

En esta línea de trabajo, esta investigación presenta el diseño e implementación de un sistema que permite mejorar los STI agregando la capacidad realizar recolección de datos de tráfico y automatizar tareas visuales. Se propone uti-

lizar visión por computadora e inteligencia computacional basada en Redes Neuronales Artificiales (ANN) para resolver problemas de análisis de tráfico mediante grabaciones de video proporcionadas por cámaras de vigilancia. Los problemas resueltos incluyen detección de vehículos, conteo, clasificación, seguimiento y detección de diferentes tipos de infracciones de tránsito, como cruce de intersecciones de semáforo en rojo y estacionamiento de vehículos en zonas no permitidas. La validación del sistema propuesto se desarrolla a partir de videos reales de tráfico de la ciudad de Montevideo, Uruguay.

Las principales contribuciones de la investigación reportadas en este trabajo incluyen: i) una metodología para el diseño de sistemas de software para el análisis de tráfico utilizando videos; ii) la implementación de un sistema modular y fácilmente extensible para realizar el análisis y recolección de datos de trafico incluyendo la implementaciones específicas de métodos de detección, conteo, clasificación y seguimiento de vehículos y iii) la validación de los métodos propuestos en distintos escenarios reales de la ciudad de Montevideo.

Parte de la investigación desarrollada en este proyecto fue publicada como artículo científico (Winter *et al.*, 2020) y presentado en el III Congreso Iberoamericano de Ciudades Inteligentes (ICSC Cities). El ICSC Cities busca promover el desarrollo de ciudades inteligentes a través de la publicación y presentación de trabajos en distintas áreas temáticas, como por ejemplo infraestructuras, energía, medio ambiente, movilidad e IoT. El artículo completo es incluido en el Anexo 1.

Este documento se organiza como se describe a continuación. En el capítulo 1 se presentan las principales motivaciones y objetivos del proyecto. A continuación, en el capítulo 2 se exponen los fundamentos teóricos en los que se basa el proyecto y se mencionan investigaciones relacionadas. El capítulo 3 describe los aspectos técnicos de la solución, el diseño y la arquitectura en conjunto con las distintas bibliotecas de software utilizadas. Luego, en el capítulo 4 se presentan los resultados de la experimentación con la arquitectura y los algoritmos implementados. Finalmente, el capítulo 5 se exponen las conclusiones y las consideraciones finales del proyecto así como también las lineas de trabajo a futuro.

Capítulo 2

Fundamentos teóricos

En este capítulo se exponen los conceptos que fueron utilizados y guiaron este estudio, necesarios para la comprensión de la implementación y la mayoría de este trabajo. Luego se realiza un estudio del estado del arte, presentando algunos trabajos relacionados en los últimos años.

2.1. Visión por computadora

Esta sección describe los principales conceptos relacionados con la detección de objetos en imágenes mediante técnicas de visión por computadora.

2.1.1. Procesamiento de imágenes

El principal objetivo de la visión por computadora es reconstruir e interpretar escenas naturales basadas en el contenido de imágenes capturadas por cámaras. El procesamiento de imagen consiste en el estudio de la formación de la imagen digital, su manipulación, la extracción de características, descripción y visualización. Esta información provee una base sólida para el punto focal de la visión por computadora: formas y patrones de objetos en una imagen que pueden ser detectados, analizados y clasificados. De hecho, la visión por computadora es el estudio de estructuras y patrones de imágenes digitales, una capa de abstracción por encima del procesamiento de imagen y la fotónica. La visión por computadora incluye procesamiento de imagen y fotónica en su conjunto de técnicas en su búsqueda de patrones de geometrías y regiones en la imagen (Peters, 2017).

2.1.2. Clasificación de imágenes

Uno de los problemas fundamentales en la visión por computadora es la tarea de asignar a una imagen de entrada una etiqueta de un conjunto fijo de categorías. Esta tarea tiene un gran número de aplicaciones siendo la base de diversos sistemas que buscan automatizar desde análisis de estudios médicos hasta tareas de control de calidad.

Una imagen se representa por una matriz donde el ancho y el largo se corresponden con la cantidad de píxeles de ancho y largo de la imagen. La matriz tiene profundidad tres donde cada nivel corresponde a cada uno de los canales RGB cuyo valor varía entre 0 y 255.

Al momento de clasificar una imagen se encuentran varias dificultades (Stanford, 2019):

- Variación de punto de vista: una misma instancia de un objeto puede encontrarse orientado de varias formas con respecto a la cámara.
- Variación de escala: los objetos presentan variación en su tamaño, no solo a nivel de escala por la cercanía de la cámara sino que también difieren en tamaño en el mundo real.
- Deformación: algunos objetos no son cuerpos rígidos y pueden ser deformados en varias formas.
- Oclusión: muchas veces solo una parte del objeto a clasificar es visible.
- Malas condiciones de iluminación.
- Camuflaje con el entorno: Los objetos pueden confundirse con su entorno volviéndolos difíciles de identificar.
- Variación entre clase: Se puede encontrar diferentes tipos de objetos de una misma clase cada uno con distinta apariencia.

El enfoque utilizado para los algoritmos de clasificación se considera "basado en datos" ya que consiste en tomar como entrada imágenes ya etiquetadas con aquellas clases que se quiere aprender a clasificar, este conjunto de datos se llama conjunto de entrenamiento.

2.1.3. Detección y segmentación

Una de las principales tareas en visión por computadora es la detección de objetos en imagen. Esta tarea se puede subdividir en segmentación, localización y detección. A continuación se describe como estas subtareas pueden ser aproximadas con técnicas de aprendizaje computacional basadas en redes neuronales convolucionales.

2.1.3.1. Segmentación semántica

El objetivo de la segmentación semántica (SS) es obtener una categoría para cada píxel dada una imagen de entrada. Un aspecto interesante de SS es que no diferencia instancias de un mismo objeto. Esto se debe a que se clasifica cada píxel de la imagen de manera independiente. Para llevar a cabo esta tarea, se utiliza una red formada por una gran pila de capas convolucionales (CONV), sin capa totalmente conectada (FC).

2.1.3.2. Clasificación y localización

La tarea de clasificación y localización consiste en etiquetar los distintos objetos presentes en una imagen indicando con un recuadro delimitador la ubicación de los mismos. En esta tarea se espera no más de una clasificación, es decir, una etiqueta que identifica cada objeto y un recuadro que indica dónde el objeto está ubicado. La figura 2.1 muestra un ejemplo de localización.

Figura 2.1: Clasificación y localización: localización del objeto detectado.

Una arquitectura básica de redes convolucionales que resuelve esta tarea recibe una imagen de entrada, la procesa a través de una gran cantidad de capas y finalmente se obtiene un vector que resume el contenido de la imagen,

el cual pasa por una capa FC para producir un conjunto de puntajes que indican la posibilidad de que la imagen contenga un objeto perteneciente a cada clase pre-establecida. Adicionalmente, se tiene otra capa FC que recibe el mismo vector y produce cuatro números que representan el ancho, el largo y las posiciones x e y, esta estructura se denomina recuadro delimitador e indica la posición del objeto dentro de la imagen.

2.1.3.3. Detección de objetos

En esta tarea se tiene un conjunto de categorías de interés y dada una imagen de entrada cada vez que una de estas categorías aparece en la imagen se quiere dibujar un recuadro alrededor y predecir cuál es. Este problema es distinto de clasificación y localización ya que puede haber un número variable de salidas para cada imagen de entrada. La figura 2.2 muestra un ejemplo de esta tarea, donde se detectaron tres objetos (dos clasificados como "DOG" y uno como "CAT").

Figura 2.2: Ejemplo de detección de objetos.

Para la detección se debe determinar todas las regiones donde pueda existir objetos para clasificar. Para ello existe una línea de trabajo llamada propuestas de regiones (RP) que tiene origen en técnicas más tradicionales de visión por computadora. Una red de propuestas de regiones (RPN) utiliza técnicas de procesamiento de señales para crear una lista de propuestas de regiones en las cuales puede haber un objeto dada una imagen de entrada. Esta lista puede contener miles de propuestas. Una RPN busca regiones (*blobs*) en la imagen y retorna un conjunto de propuestas de regiones candidatas potenciales

a contener objetos. En lugar de aplicar la red de clasificación a cada posible lugar y tamaño de recuadro dentro de la imagen, lo que se hace es primero aplicar una RPN para obtener un conjunto de RP donde los objetos pueden estar ubicados, para luego aplicar una CNN con el fin de clasificar cada una de las regiones propuestas.

Este tipo de arquitectura lleva el nombre de R-CNN. Dada una imagen de entrada, se ejecuta una RPN para obtener las propuestas, también llamadas regiones de interés (RoI). Esta aproximación presenta algunos inconvenientes ya que es computacionalmente costosa, en la práctica el entrenamiento es lento y ocupa mucho espacio en disco.

Con el fin de solucionar estos problemas se creó Fast R-CNN (Girshick, 2015), una red que trabaja de manera similar a R-CNN. Se comienza con la imagen de entrada, pero en lugar de procesar cada RoI de forma separada, se ejecuta la imagen completa en capas convolucionales para obtener un vector convolucional de características de alta resolución de la imagen. Luego se utilizan RoI, pero no sobre la imagen, sino que se obtienen desde el vector convolucional. Esto permite la reutilización de cómputo convolucional a través de toda la imagen. Si se tiene capas FC, estas capas esperan una entrada de tamaño fijo y por lo tanto se debe remodelar estas RoI del vector convolucional y esto se hace de una forma diferenciable utilizando una capa RoI de agrupación (RoI Pooling).

En términos de velocidad Fast R-CNN ha demostrado ser diez veces más rápida que CNN en tiempo de entrenamiento gracias a que comparte el computo entre los diferentes vectores de clasificación. Al momento de clasificar una nueva imagen el tiempo de cómputo es determinado por el cálculo de las RoI, lo cual resulta ser el cuello de botella de la arquitectura. Este último problema es resuelto en Faster R-CNN (Ren *et al.*, 2015) cuyo objetivo principal es hacer que la CNN realice las propuestas. Nuevamente, se parte de la imagen de entrada, se ejecuta la imagen completa a través de capas convolucionales para obtener un vector convolucional representando la imagen completa en alta resolución y se tiene una RPN separada que trabaja sobre el vector convolucional prediciendo sus propias RoI dentro de la red. Una vez que se obtienen las propuestas continúa similarmente a Fast R-CNN.

2.1.3.4. Segmentación de instancias

Dada una imagen de entrada la segmentación de instancias busca predecir, por un lado, las ubicaciones y clasificaciones de los objetos en la imagen (similar a detección de objetos), y adicionalmente, en lugar de predecir simplemente una región para cada objeto, se busca predecir una máscara de segmentación para cada uno, determinando de esta forma qué píxeles en la imagen corresponden a cada objeto. La figura 2.3 muestra un ejemplo de la resolución de segmentación de instancias para una imagen en la que se detectaron tres objetos.

Figura 2.3: Segmentación de instancias.

Uno de los métodos más recientes para resolver la segmentación de instancias es el llamado Mask R-CNN (He *et al.*, 2017), el cual es similar a Faster R-CNN y sigue un enfoque de procesamiento multi etapa. Este método recibe la imagen completa, se ejecuta la imagen a través de una red convolucional y una RPN (al igual que Faster R-CNN), una vez aprendidas las RoI se proyectan estas propuestas en el vector convolucional (al igual que Fast y Faster R-CNN). Luego, en lugar realizar la clasificación y la región para regresión para cada una de las RoI, se busca predecir una máscara de segmentación para cada una de las regiones.

2.1.4. Redes Neuronales Convolucionales

En esta sección se describen las redes neuronales convolucionales, las capas que las componen y la función que cumplen. Luego, se exponen arquitecturas de redes neuronales convolucionales de interés para este trabajo.

2.1.4.1. Descripción de las redes neuronales convolucionales

En los últimos años la inteligencia computacional y el aprendizaje profundo han dado lugar a resultados exitosos en una variedad de problemas, incluyendo la clasificación de imágenes. Entre los diferentes tipos de redes neuronales artificiales profundas, las redes neuronales convolucionales (CNN por sus siglas en inglés, Convolutional Neural Networks) se han estudiado ampliamente (Gu *et al.*, 2018). Las CNN asumen que la entrada a clasificar es una imagen, lo cual permite que la red sea más eficiente y diseñar arquitecturas que reduzcan en gran medida el número de parámetros de la red.

A diferencia de las redes neuronales regulares (NN por sus siglas en inglés, Neural Networks), cada capa de las CNN dispone las neuronas en 3 dimensiones: alto, ancho y profundidad. A su vez, lugar de que cada capa se encuentre completamente conectada las neuronas de una capa solamente se conectan con algunas neuronas de la capa anterior.

En las CNN se puede hallar distintos tipos de capas de neuronas, algunas de las más utilizadas son:

- Entrada: esta capa es del tamaño de la imagen que se quiere procesar incluyendo el alto, ancho y profundidad (3 en caso de una imagen RGB, 1 en caso de escala de grises).
- Convolucional: capa que aplicará distintos filtros computando el producto de matriz entre sus pesos y distintas regiones locales de la entrada, cada filtro generará una nueva dimensión de profundidad en la salida, pero se mantendrá el ancho y largo de la entrada.
- Activación: capa que aplica función de activación, el tamaño de la salida es el mismo que el de la entrada.
- Agrupación (pooling en inglés): Capa que realiza una reducción de la entrada con respecto al ancho y la altura.
- Completamente Conectada (FC por sus siglas en inglés, Fully Connected): Capa utilizada en la salida, realiza la clasificación en la distintas clases, por lo tanto tendrá un tamaño de $(1 \times 1 \times N)$ donde N es la cantidad de clases en la que se puede clasificar la imagen.

Mediante estas capas una CNN convierte una imagen de entrada en una salida que indica una puntuación para cada una de las clases a clasificar. Algunas de estas capas cuentan con parámetros y por los tanto con pesos que se

pueden ajustar, como por ejemplo las convolucionales o las FC, mientras que otras capas como las de agrupación o las que aplican función de activación son simplemente una función fija.

Las capas convolucionales (CONV) son el principal componente de las CNN y las que tienen un mayor costo computacional. Los parámetros de una capa CONV consisten en un conjunto de filtros de un tamaño reducido de ancho y largo con respecto a la entrada pero se extiende a lo largo de toda la profundidad. Por ejemplo, un filtro puede ser de tamaño $5 \times 5 \times 3$, 5 píxeles de ancho y largo y 3 para la profundidad que serían los 3 canales RGB.

El tamaño de la salida de una capa CONV está dado por 3 hiper parámetros:

- Profundidad: corresponde a la cantidad de filtros que se utilizan, cada filtro busca aprender alguna característica distinta de la entrada.
- Paso (stride en inglés): indica cuántos píxeles se mueve el filtro a través de la imágen en cada convolución.
- Relleno con ceros (zero-padding en inglés): permite controlar el tamaño espacial de la salida. Normalmente se utiliza para preservar el mismo tamaño de la entrada, ya que al aplicar los distintos filtros los mapas de activación pueden ser de mayor o menor tamaño que la entrada.

2.1.4.2. Arquitecturas de Redes Neuronales Convolucionales

A continuación se presentan algunas arquitecturas exitosas, ganadoras del ILSVRC en los últimos años. Estas arquitecturas son comúnmente utilizadas como redes troncales para resolver una gran variedad de tareas relacionadas con la visión por computadora.

La ganadora del ILSVRC 2014 fue una CNN de google. Su mayor contribución fue el desarrollo del módulo de inicio (inception module en inglés) que redujo drásticamente el número de parámetros en la red (4 millones, comparando con AlexNet de 60 millones). Adicionalmente utiliza agrupación promedio (average pooling en inglés) en lugar de capas FC en las últimas capas de la CNN, eliminando una gran cantidad de parámetros.

El segundo lugar del ILSVRC 2014 fue para la red VGGNet, cuyo mayor aporte consistió en mostrar que la profundidad de la red es un componente crítico para un buen rendimiento. En su forma final presentó una red con 16 capas CONV/FC, y muestra una arquitectura homogénea que realiza convo-

luciones 3x3 y agrupación 2x2 de comienzo a fin. Por otro lado, es una red que utiliza una gran cantidad de memoria y parámetros (140 millones).

ResNet (Residual Network), ganadora del ILSVRC 2015, presenta las denominadas condiciones de salto (skip conditions en inglés) y un alto uso de normalización batch. Además prescinde de la capa FC al final. Esta red supone un despegue en la cantidad de capas utilizadas hasta el momento.

2.2. Seguimiento de objetos

El seguimiento de objetos (tracking en inglés) consiste en el proceso de estimar el estado de un objeto -identificación, posición, configuración- a lo largo del tiempo a partir de observaciones, generando de esta manera la trayectoria del objeto la cual está dada por su posición en cada fotograma del video.

Décadas de investigación han generado grandes avances en el campo de seguimiento de objetos, considerado un problema fundamental en el área de visión por computadora. Sin embargo aún falta mucho para obtener resultados satisfactorios en videos que presentan desafíos como movimientos bruscos de la cámara u objetos que desaparecen y luego reingresan a la escena debido a que son ocluidos. Los sistemas de seguimiento tienen diversas aplicaciones en la actualidad como ser la comprensión semántica del comportamiento en videos o los sistemas de videovigilancia.

Cuando el seguimiento se realiza sobre varios objetos a la vez se denomina Seguimiento de Objetos Múltiples (MOT por sus siglas en inglés, Multiple Object Tracking). En este escenario la dificultad de la tarea aumenta considerablemente debido a la oclusión generada por la interacción de los objetos que a su vez pueden tener apariencias similares. Por otro lado, variantes como la velocidad en que se mueven los objetos, la iluminación o que estos cambien de apariencia dependiendo de la posición, exigen que los sistemas de seguimiento deban ser robustos manteniendo la identificación del objeto en dichas situaciones.

Los métodos de seguimiento son aplicados en diversos escenarios en los cuales la forma en la que se procesa el video difiere, una de las principales distinciones es si el procesamiento se realizara en linea (online) o fuera de linea (offline). Esto da lugar a la necesidad de distintos modelos de seguimiento que deberán realizar el procesamiento contando con distinta cantidad de información. En los modelos online el video es procesado fotograma a fotograma, el

método deberá generar una salida para cada fotograma basándose solamente en el fotograma actual y los fotogramas anteriores, esto permite su aplicación en escenarios que el video debe procesarse en tiempo real como la navegación de vehículos autónomos por ejemplo. Por otro lado los modelos offline permiten que el método tenga acceso a todos los fotogramas del video al momento del procesamiento, por este motivo este tipo de métodos suelen presentar mejores resultados pero al costo de un dominio de aplicación mas restringido.

2.2.1. Métodos de seguimiento

En los métodos clásicos de seguimiento, las características (features) de los objetos se extraen en cada fotograma del video y luego son utilizadas para localizar el mismo objeto en los fotogramas siguientes (Zhang *et al.*, 2014). Esto causa que los errores se acumulen en el proceso y el seguimiento pueda fallar debido a los cambios que presentar las características en ventanas temporales cuando los objetos son ocluidos o se deforman.

En los últimos años los algoritmos de detección han mejorado substancialmente tanto en precisión así como en velocidad de procesamiento debido a la aplicación de redes neuronales convolucionales. Esto ha dado lugar a métodos de seguimiento basados en detección (DBT por sus siglas en inglés, Detection Based Tracking) (L Leal-Taixé, 2014). Esta aproximación consiste en separar el método de seguimiento en dos etapas: detección de objetos y asociación de datos. La etapa de detección de objetos es resuelta aplicando métodos de detección de objetos como por ejemplo redes neuronales especialmente diseñadas para cumplir con dicho propósito (ver sección 2.1.3.3). Por otro lado, el enfoque de la segunda etapa se encuentra en asociar las detecciones de un mismo objeto a través de los distintos fotogramas del video. Para realizar dicha asociación distintas características de las detecciones son utilizadas, a continuación se presentan algunas que a pesar de su simplicidad consiguen lograr buenos resultados.

2.2.1.1. Intersección sobre unión

Este es un método para resolver la asociación de detecciones a partir de la intersección sobre la unión (IOU por sus siglas en inglés, Intersection Over Union) que presentan las detecciones en distintos fotogramas del video. Este método asume dos supuestos principalmente, en primer lugar que el algoritmo

de detección utilizado deberá tener pocas o ninguna falla en las detecciones realizadas por lo cual se espera una detección en cada fotograma por cada objeto que se desee seguir. En segundo lugar se asume que la detección de un mismo objeto en dos fotogramas consecutivos presentan un gran solapamiento de la intersección sobre la unión, lo cual es común cuando nos encontramos con videos que presentan una alta tasa de refresco.

La intersección sobre la unión se define como:

$$IOU(a,b) = \frac{Area(a) \cap Area(b)}{Area(a) \cup Area(b)}$$

Si ambas condiciones se cumplen, el seguimiento se vuelve una tarea sencilla e incluso puede realizarse sin depender de información sobre la imagen del objeto, simplemente a partir de su posicionamiento. De esta forma, dada una detección, si se desea saber a qué objeto corresponde de los que se han seguido hasta el fotograma actual, lo que se hace es ver cuál de las detecciones en los fotogramas anteriores presenta la mayor IOU. La figura 2.4 ilustra el principio de seguimiento de detecciones a partir de la intersección sobre la unión.

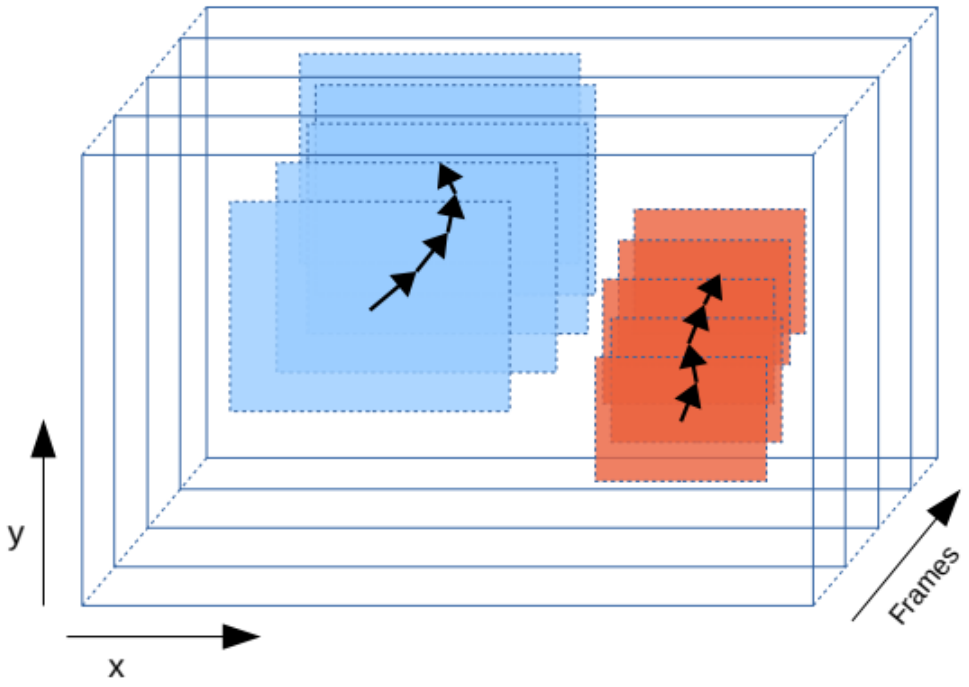


Figura 2.4: Principio básico de IOU.

Luego, el desempeño del método IOU puede mejorarse filtrando aquellas intersecciones que tengan una distancia menor a cierta distancia mínima así como también considerar solamente aquellas detecciones que superen cierto valor de confianza. La ventaja de este método además de su simpleza es que requiere bajo costo computacional en comparación con otros métodos.

2.2.1.2. Seguimiento en tiempo real en línea simple

El seguimiento en tiempo real en línea simple (SORT por sus siglas en inglés, Simple Online Realtime Tracker) fue propuesto por Bewley (Bewley *et al.*, 2016) como un algoritmo que, a pesar de su simpleza, lograba un desempeño comparable con el estado del arte manteniendo una buena velocidad de procesamiento. Es un método en línea basado en detección, lo cual significa que al procesar el video se basa en las detecciones del fotograma actual y las de los fotogramas anteriores.

En el trabajo original para realizar las detecciones los autores se basaron en una red Faster Region CNN la cual fue utilizada con parámetros por defecto y entrenada en el conjunto de datos PASCAL VOC, para luego usarla filtrando solo las detecciones de peatones que superaran una probabilidad del 50 %. Sin embargo por el paradigma de este algoritmo (basado en detección) es posible variar el detector utilizado mientras se mantenga el formato de la detección el cual consiste en recuadros delimitadores (bounding boxes) indicando el centro de estas mediante dos coordenadas y la relación de aspecto dada por el ancho y el largo del recuadro, en conjunto con estos valores el algoritmo espera que para cada detección se indique cual es la probabilidad asociada a la detección.

A partir de los datos provistos por la detección en cada fotograma el algoritmo elabora un modelo de estado interno del objeto, el cual utiliza para propagar la identidad del objeto a través de los fotogramas y así realizar la asociación manteniendo el identificador que le corresponde. Este modelo consiste en los datos del recuadro delimitador en conjunto con la velocidad lineal calculada con respecto al fotograma anterior. Se puede representar dicho modelo como el vector $x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T$, donde u y v representan el centro del recuadro delimitador dada por la cantidad de píxeles horizontal y vertical respectivamente, s y r representan el área y la relación de aspecto del recuadro (la cual es considerada constante) y por último $\dot{u}$, $\dot{v}$ y $\dot{s}$ son los componentes que representan la velocidad lineal, estos se calculan aplicando filtros de

Kalman (Kalman, 1960), estas velocidades se calculan al momento de asociar la detección de un nuevo fotograma al seguimiento que le corresponde a dicho vector. Si no hay una detección que le corresponda el estado simplemente se actualiza realizando la predicción sin la corrección a partir del modelo de velocidad lineal.

Para asociar las detecciones en un nuevo fotograma con los seguimientos que se tienen, se realiza una estimación de cada seguimiento prediciendo la posición del recuadro delimitador en el nuevo fotograma y luego se computa una matriz de costo a partir de la intersección sobre la unión (IOU) de la distancia entre cada detección y cada recuadro delimitador estimada para cada seguimiento. Esta asignación es resuelta de forma óptima aplicando el algoritmo Húngaro. A su vez un parámetro IOU_{min} se utiliza para descartar las asignaciones donde la IOU entre las detecciones en fotogramas consecutivos es muy baja. Mediante este parámetro se puede controlar el comportamiento del algoritmo ante oclusiones momentáneas de los objetos, ya que que en estos casos la IOU entre detecciones de un mismo objeto en fotogramas consecutivos suele ser menor.

Identificadores únicos deben ser creados y destruidos a medida que los objetos entran y salen de la escena. Para crear un nuevo identificador, es decir, un nuevo seguimiento que le corresponde a un nuevo objeto que apareció en el video, se consideran los casos donde la IOU de la detección y todos los seguimientos es menor que IOU_{min}. Por otro se eliminan de la escena aquellos seguimientos que no son detectados nuevamente por una cantidad T_{Lost} de fotogramas.

A pesar de lograr un buen rendimiento en términos de precisión y exactitud de seguimiento, SORT genera un número alto de cambios de identidad. La métrica utilizada para la asociación (IOU) logra mantener los identificadores de los objetos seguidos cuando estos no son ocluidos por varios fotogramas seguidos ya que en dichas situaciones el algoritmo falla al no poner superponer correctamente el objeto en distintos fotogramas entonces el algoritmo pierde el seguimiento del objeto asignándole un nuevo identificador.

2.3. Trabajos relacionados con detección y clasificación de tráfico

Zhou *et al.* (2016) investigaron el problema de detección y clasificación de vehículos basados en imágenes de vistas traseras, capturadas por una cámara estática en una carretera multicarril. En este trabajo se propuso una combinación de enfoques en el uso de redes neuronales profundas (DNN por sus siglas en inglés, Deep Neural Network). Por un lado, se afinaron las capas más altas del modelo de la DNN, y por otro lado, se extrajeron las capas más altas de la arquitectura de la DNN como características de alto nivel. Para la detección de vehículos se utilizó la arquitectura You Only Look Once (YOLO) (argumentando su mayor eficiencia sobre Fast-CNN) y realizando un post procesamiento para eliminar resultados inválidos. Para la clasificación, extracción de características y afinado, se utilizó la arquitectura Alexnet, en la cual removieron dos de las capas FC y ajustaron el tamaño de la tercera. Adicionalmente, se trabajaron en la clasificación de imágenes oscuras aplicando transformaciones sobre estas para que se vieran como imágenes "normales". Finalmente, compararon la red YOLO con el método de modelos de partes deformables (DPM por sus siglas en inglés, Deformable Parts Models) obteniendo una precisión similar. Para la clasificación compararon la red Alexnet-SVM, con otros métodos como análisis de componente principal con diferencia de espacio de vehículo (principal component analysis, difference from vehicle space, PCA+DFVS), obteniendo mejores resultados en un conjunto de datos público y comparando contra métodos como vector de Fisher para su conjunto de datos propio, mejorando también los resultados previos.

Uy *et al.* (2016) analizaron la posibilidad identificar infracciones de tránsito mediante el uso de algoritmos genéticos. A su vez propusieron el reconocimiento de la matrícula de dichos infractores mediante redes neuronales. El algoritmo genético fue utilizado para reconocer vehículos obstruyendo el cruce peatonal. Los autores se basaron en buscar la proporción de píxeles blancos en la región de interés para así distinguir los casos en los cuales había una infracción. A su vez, otro algoritmo del mismo tipo se utiliza para identificar la zona de la imagen donde se encuentra la matrícula. La imagen recortada de la matrícula es sometida a un procesamiento con el objetivo de reducir el ruido y luego se utiliza una red neuronal para reconocer el número. Según los autores, el sistema presentó una precisión de 91,6 % en el reconocimiento de matrículas,

en base a un conjunto de 47 imágenes de prueba. A pesar de esto los autores aclararon que en ciertos casos el algoritmo encargado de identificar la región de la matrícula no podía hacerlo debido a la posición en la cual quedaba el vehículo con respecto a la cámara.

Zhang *et al.* (2017) buscaron resolver el problema de conteo vehicular mediante una red neuronal que combina la arquitectura de FCN (Fully Convolutional Neural Networks) con redes rLSTM (Long Short Term Memory Networks), arquitectura utilizada para incorporar información espacio-temporal al momento de realizar la estimación mediante aprendizaje residual. El modelo fue entrenado y probado en el conjunto de datos anotados WebCamT, un modelo público que presenta videos capturados en ciudades de gran escala, los cuales presentan baja resolución, baja tasa de refresco (1 fotograma/segundo) y alta oclusión. En comparación con el estado del arte sobre dicho conjunto de datos, la arquitectura propuesta redujo el error absoluto medio (EAM) de 2,74 a 1,53. Por otro lado, se entrenó y evaluó el modelo con el conjunto de datos público TRANCOS, el cual presenta una colección de 1244 imágenes de distintos escenarios de tránsito, lo cual da un total de 46796 vehículos anotados. En TRANCOS el modelo propuesto redujo el EAM de 5.31 a 4.21 en comparación con el estado del arte. Como ventaja de la arquitectura propuesta, los autores destacaron que se aceleró el tiempo de entrenamiento de la red en hasta cinco veces. Las evaluaciones realizadas demostraron efectividad y robustez, pero por otro lado una restricción que se presentó es que la red no es capaz de considerar periodos largos de información temporal dado a la cantidad de memoria que esto conlleva.

Dey *et al.* (2018) propusieron una metodología para analizar y categorizar tráfico utilizando CNN en la clase de circuitos integrados System-On-a-Programmable-Chip (SOPC), la cual llamaron Motionless Analysis of Traffic Using Convolutional Neural Networks on SOPC: MAT-CNN-SOPC. En este estudio también se introduce la variable calidad de experiencia (quality of experience) la cual mejoraría el mecanismo de predicción del modelo de CNN elegido. El principal objetivo de esta investigación fue el desarrollar un modelo CNN sencillo de entrenar, que no requiriera muchas imágenes en el conjunto de entrenamiento, y que pudiera ser ejecutado en un SOPC, sin la necesidad de enviar imágenes a un servidor para continuar procesando. Para efectivamente entrenar la CNN para mejorar la precisión de las predicciones se utilizó una combinación de aprendizaje por transferencia con un nuevo mecanismo para

reentrenar modelos CNN, en el cual el modelo es reentrenado con imágenes de un ambiente conocido.

Arinaldi *et al.* (2018) presentaron un sistema de análisis de tráfico basado en técnicas de visión por computadora diseñado para recolectar automáticamente estadísticas como conteo y clasificación de vehículos, estimación de velocidad y uso de carriles. Para este propósito se implementaron dos modelos: Mixture of Gaussian (MoG) + SVM y Faster RCNN. Para el entrenamiento y la validación los autores utilizaron un conjunto de datos propio de videos de carreteras de Indonesia y un conjunto de datos público del MIT. Como resultado, demostraron experimentalmente que Faster RCNN fue el más apto para la detección y clasificación de vehículos en movimiento en una escena dinámica de tráfico. Faster RCNN es superior en la detección, debido a la debilidad de MoG cuando se trata de vehículos que se superponen, que se encuentran muy cerca, o incluso durante la noche. Adicionalmente, se demuestró que Faster RCNN también superó a SVN en la clasificación.

Chauhan *et al.* (2019) estudiaron la posibilidad de utilizar CNN en distintas plataformas embebidas para realizar conteo, clasificación y detección de violaciones de reglas de tránsito en tiempo real sobre calles de la ciudad de Delhi, India. Los autores propusieron el uso de una red YOLO pre-entrenada en el conjunto de datos MS-COCO, que luego fue ajustada con los conjuntos de datos anotados PASCAL VOC y KITTI. Estos conjuntos de datos contenían imágenes relacionadas al tráfico en países desarrollados que luego ajustaron y mejoraron mediante aproximadamente 5500 imágenes recolectadas y anotadas en la zona geográfica donde se propuso el estudio. Por otro lado, se consideraron tres plataformas embebidas de distinto costo y capacidad computacional, para las cuales se comparó el desempeño en cuanto a latencia y consumo energético. Como resultado el mejor modelo entrenado logró un desempeño de 65-75 % EAM dependiendo de la posición de la cámara y la clase de vehículo. Los autores encontraron que uno de los principales factores que afecta al desempeño es la oclusión dada por las características y la cantidad de tráfico en los puntos considerados. En cuanto a las plataformas estudiadas, los autores concluyeron que un aspecto importante es la capacidad de soportar distintas bibliotecas y que la mejor plataforma en relación costo desempeño depende de si se busca desarrollar aplicaciones en tiempo real y si debe depender de una fuente de energía portátil. Este artículo presentó una noción del desempeño esperado de los modelos YOLO luego de ser optimizados mediante varios conjuntos

de datos, incluyendo un conjunto de datos anotados relevado en el contexto que se realizó el estudio. También indicó que actualmente hay plataformas de hardware que permiten ejecutar sistemas embebidos de forma independiente, proporcionando una gran variedad de aplicaciones a costos accesibles.

Zheng *et al.* (2019) propusieron una CNN para la predicción de la gravedad de accidentes de tráfico TASP-CNN (Traffic Accident's Severity Prediction) que, a diferencia de trabajos previos, consideró la relación entre las características del accidente. Los aportes de este trabajo son el algoritmo FM2GI que transforma la matriz de características del accidente de tráfico en una imagen de escalas de gris y la arquitectura TASP-CNN. El método propuesto se adapta exitosamente a la representación de características de gravedad de accidentes de tránsito, tales como la combinación de características y correlaciones más profundas de datos de accidentes. El rendimiento de este método fue evaluado utilizando datos de un período de 8 años, mostrando ser mejor que modelos como el algoritmo K Nearest Neighbors (KNN) y regresión logística.

2.4. Trabajos relacionados con seguimiento de objetos

Bewley *et al.* (2016) buscaron una aproximación pragmática al problema de MOT, desarrollando un método de seguimiento denominado SORT (por sus siglas en inglés, Simple Online Realtime Tracking) que puede ser utilizado en aplicaciones de tiempo real. Se basaron en las detecciones provistas fotograma a fotograma por una red Faster RCNN y aplicaron el algoritmo Húngaro, filtros de Kalman e IOU para realizar el seguimiento de los objetos basándose solamente en su posición y tamaño. El resultado fue un método de seguimiento que a pesar de depender fuertemente del desempeño de la detección, consigue resultados comparables con el estado del arte en términos de velocidad y precisión.

Posteriormente, Bewley *et al.* (2017) extendieron su método de seguimiento basado en detecciones SORT integrando información sobre la apariencia de los objetos a seguir para mejorar el rendimiento. Como resultado obtuvieron un nuevo método, DeepSORT. Para obtener este nuevo método, entrenaron una red CNN para extraer y discriminar características, la cual embebieron a su biblioteca original para determinar las métricas de asociación de cada objeto.

Dichas métricas fueron utilizadas en un algoritmo KNN tradicional para realizar la asociación entre las detecciones en el fotograma actual y los anteriores. Como resultado se obtuvo una biblioteca de seguimiento más robusta, capaz de seguir objetos por periodos más largos de oclusión, reduciendo el número de falsas re-asignaciones de identidad en un 45 % con respecto a su trabajo anterior. Por otro lado, la nueva implementación redujo la velocidad de procesamiento y pasó a depender de las imágenes de las detecciones y no solo de su tamaño y posición. El impacto en el rendimiento hizo que el método sea apenas utilizable para aplicaciones en tiempo real.

Fu *et al.* (2019) extendieron el método SORT agregando una asociación basada en las características de movimiento de los objetos a seguir. Denominaron su trabajo "seguimiento sencillo en línea y en tiempo real con funciones de movimiento" (MF-SORT por sus siglas en inglés, Simple Online and Realtime Tracking with Motion Features). Su argumento para la modificación fue que el computo de características de apariencia era demasiado demandante (como realizaron Bewley *et al.* (2017)). Por otro lado, al centrarse en características del movimiento de los objetos se podría obtener un desempeño similar a una fracción de costo computacional. Para llevar a cabo la extensión, la arquitectura de DeepSORT fue modificada, cambiando el módulo que asocia las correspondencias entre las detecciones del fotograma actual y los seguimientos de los fotogramas anteriores de tal forma que esta se haga a partir de las características de movimiento de los objetos y no de la apariencia. Con este objetivo, los autores utilizaron la distancia de Mahalanobis, ya que es una métrica más rápida de calcular que la que se realiza en DeepSORT. Otro aporte de este trabajo es un nuevo conjunto de datos de pruebas denominado MOT-SOCCER. Los autores consideraban que el conjunto de datos de prueba estándar utilizado para medir este tipo de algoritmos (MOT Challange) presentaba un cuello de botella en el avance de los algoritmos debido a que se enfocaba solamente en vídeos de vigilancia con gran cantidad falsos positivos (falsas detecciones) y falsos negativos (detecciones perdidas). Como resultado obtuvieron que MF-SORT tiene una precisión de seguimiento de múltiples objetos (MOTA por sus siglas en inglés, Multiple Object Tracking Accuracy) comparables con los de DeepSORT en las pruebas realizadas sobre MOT Challange y sobre MOT-SOCCER, pero con un costo computacional hasta 3 veces menor, por lo que hace que MF-SORT tenga un mejor desempeño para aplicaciones en tiempo real.

Capítulo 3

Diseño e implementación del sistema propuesto

El objetivo de este capítulo es presentar los detalles de implementación del sistema propuesto para el análisis de datos de tráfico. Se describen las decisiones tecnológicas en cuanto a librerías de software así como también de entorno de desarrollo y hardware. Luego se detalla la arquitectura, el diseño del sistema propuesto y la implementación de los distintos módulos que lo componen.

3.1. Selección de tecnologías

En esta sección se exponen los principales aspectos del diseño y la implementación del sistema propuesto. En primer lugar, se hace mención de las tecnologías analizadas y seleccionadas para utilizar. Luego, se describe el entorno de desarrollo y ejecución del sistema.

3.1.1. Bibliotecas de Software

Las bibliotecas de software brindan funcionalidades y algoritmos ya implementados disminuyendo así el tiempo requerido para la implementación del sistema. El análisis de trabajos relacionados permitió concluir que existe una gran oferta de tecnologías destinadas al desarrollo de sistemas relacionados con inteligencia artificial y, más concretamente, relacionados con su aplicación para visión por computadora.

En un trabajo de similares características tecnológicas al sistema desarrollado en este proyecto, Chavat y Nesmachnow (2018) se basaron en las siguientes consideraciones al momento de seleccionar las bibliotecas de software:

- Soporte multiplataforma, lo que permite su utilización sin saber, a priori, sobre qué tipo de hardware y sistema operativo será finalmente ejecutado.
- Que el lenguaje y las bibliotecas sean de uso gratuito y preferentemente de código abierto. Esta característica permite aprender de lo que ya está implementado, evaluar su aplicabilidad y eventualmente poder efectuar cambios en los algoritmos de terceros.
- Que el lenguaje y las bibliotecas dispongan de una comunidad amplia y activa para que los problemas encontrados en estas puedan ser rápidamente solucionados. Además, el hecho de tener una comunidad activa es un fuerte indicio de que el lenguaje o biblioteca se encuentra en uso en la actualidad.
- Que el lenguaje presente los mejores niveles de rendimiento posibles en las áreas que aborda el sistema. La elección de la tecnología no debe incidir de forma significativamente negativa en el rendimiento del sistema ya que es un aspecto fundamental al procesar imágenes en tiempo real.

Teniendo en cuenta estas consideraciones se investigaron bibliotecas de software destinadas al desarrollo de aplicaciones basadas en visión por computadora, buscando que las mismas implementen total o parcialmente alguno de los algoritmos presentados o proporcionen un entorno para el desarrollo y la ejecución de esta clase de aplicaciones. A continuación se detallan las opciones mas relevantes.

OpenCV (Howse, 2013) es una biblioteca de código abierto con más de 2500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos de visión por computadora y aprendizaje automático tanto clásicos como de última generación. Estos algoritmos pueden ser utilizados para detectar y reconocer caras, identificar objetos, clasificar acciones humanas en videos, rastrear movimientos de cámara, rastrear objetos en movimiento, etc. OpenCV tiene interfaces en C++, Python, Java y MATLAB, a su vez es compatible con Windows, Linux, Android y Mac OS. OpenCV se inclina principalmente hacia aplicaciones de visión en tiempo real y sus licencias permiten su utilización tanto en el ámbito académico así como también en el comercial.

SimpleCV (SimpleCV, 2020) es una biblioteca de código abierto para crear aplicaciones de visión por computadora. Su objetivo es facilitar la implementación de este tipo de aplicaciones abstrayendo conceptos complejos y exponiendo una API de alto nivel que ofrece funcionalidades como detección y seguimiento de objetos, clasificadores basados en aprendizaje automático y manipulación de imágenes y videos. Con SimpleCV, obtiene acceso a varias bibliotecas de gran potencia como OpenCV además de contar con varios ejemplos de uso.

MATLAB (Matlab, 2020) es una herramienta útil para crear aplicaciones de análisis de datos y procesamiento de imágenes. Tiene un amplio uso en el ámbito académico y de investigación, dado que MATLAB permite la creación rápida de prototipos. Otro aspecto interesante es que el código de MATLAB es conciso, lo que facilita su lectura y depuración. Un inconveniente puede ser que es una herramienta paga. Además, puede ser bastante lento durante el tiempo de ejecución.

Google Cloud (Challita *et al.*, 2018) ofrece un servicio llamado Vision AI el cual permite realizar procesamiento de imágenes y videos al encapsular modelos de aprendizaje profundo pre-entrenados en una API REST, accesible en la nube. Su objetivo es facilitar el acceso al procesamiento de imágenes y videos mediante aprendizaje profundo sin requerir conocimientos previos en el área. Entre las funcionalidades ofrecidas es posible identificar objetos, textos, personas, escenas y actividades, además de proporcionar análisis faciales y reconocimiento facial para el análisis de sentimientos. A su vez se ofrece funcionalidades como etiquetado de datos y posibilidad de entrenar modelos para una tarea especifica. Similares a ésta opción están las ofertas de Amazon Rekognition (Rekognition, 2020) y Microsoft Azure Computer Vision (Del Sole, 2018). Como desventaja de este tipo de servicios es que no son gratuitos y su implementación no es de código abierto.

TensorFlow (Abadi, 2016) es una biblioteca de código abierto mantenida por Google que permite desarrollar y entrenar modelos de aprendizaje automático. Su API de alto nivel facilita la creación de redes de aprendizaje profundo para visión por computadora además de contar con un amplio repositorio de modelos y conjuntos de datos que pueden ser utilizados para dicho fin. Cuanta con soporte oficial para los lenguajes Python, C++ y JavaScript, a su vez es compatible con Ubuntu, Windows, MacOS y Raspbian. Para poder ejecutar Tensorflow es necesario una unidad de procesamiento gráfico (GPU) NVIDIA con arquitectura CUDA mientras que TensorFlow Lite permite la realizar la inferencia en dispositivos móviles.

Detectron2 (Wu *et al.*, 2019) de Facebook AI Research es una biblioteca en Python que haciendo uso de la biblioteca PyTorch, se basa en la primera versión Detectron que implementa redes de tipo Faster R-CNN y Mask R-CNN (Massa y Girshick, 2018). Detectron2 implementa distintos algoritmos de detección del estado del arte, entre los cuales se encuentra DensePose (Güler *et al.*, 2018), redes de pirámides de funciones panorámicas (Kirillov *et al.*, 2019) y numerosas variantes de la familia de modelos Mask R-CNN (He *et al.*, 2017). Su diseño modular es flexible y extensible, facilita el entrenamientos de modelos y permite realizar inferencia con soporte para varios GPUs.

Gilewski expuso una arquitectura modular basada en un pipeline de procesamiento de imágenes (Gilewski, 2019). Este proyecto, implementado en Python, surge como una demostración del uso de Detectron2 y OpenCV. Adicionalmente, este trabajo hace uso de generadores Python para estructurar el pipeline de manera que los módulos tengan un acoplamiento débil entre ellos.

IOU Tracker es una implementación en Python de los algoritmos de seguimiento IOU/V-IOU (Bochinski *et al.*, 2017, 2018). Cuenta con documentación que muestra el uso de este módulo desde la línea de comandos. Tiene la desventaja de que no permite la ejecución en tiempo real, ya que para la invocación requiere la ruta a un archivo con las detecciones de todos los fotogramas.

Move-lab (Move-lab, 2018) expone paso a paso la creación de su algoritmo de seguimiento basado en IOU. Se buscó enriquecer los resultados de detección de una red YOLO agregando y manteniendo un identificador único para cada objeto. Partiendo de un algoritmo simple (Shiffman, 2011), fue mejorado con los aportes de Bochinski (Bochinski *et al.*, 2017) y Bewley (Bewley *et al.*, 2016). El producto final, llamado node-moving-things-tracker, consiste en una implementación de un algoritmo de seguimiento de objetos múltiples basado en detección que permite obtener resultados en tiempo real. Esta biblioteca se encuentra desarrollada en JavaScript y puede ser utilizado desde la línea de comandos o como un módulo auxiliar.

Se decidió utilizar Detectron2 como biblioteca base para el desarrollo ya que cumple con la mayoría de las características tecnológicas deseables: soporte multiplataforma, de uso gratuito (y código abierto), comunidad activa y con rendimiento documentado.

Para el seguimiento de objetos se tomaron criterios similares que para la detección con el objetivo de seleccionar las bibliotecas disponibles. La dificultad en este caso fue que, si bien existen trabajos de investigación referidos al

seguimiento de objetos, la implementación utilizada no se encuentra disponible para el uso público. Adicionalmente, varios de los trabajos encontrados fueron creados en el contexto de alguna competencia específica (por ejemplo MOT-Challenge, 2019), por lo cual su eficiencia no está garantizada en un conjunto de datos distinto al propuesto por la competencia. Otros trabajos relevados no contaban con una documentación que permitiera el entendimiento y el uso de la solución.

La biblioteca seleccionada para el seguimiento fue node-moving-things-tracker (NMTT). En esta decisión se consideró de importancia que la biblioteca permite obtener resultados en línea, es decir, que tras el trabajo de detección en cada fotograma se pueda tener el conjunto de identificadores correspondientes a las nuevas detecciones.

3.1.2. Entorno de desarrollo y ejecución

La utilización de Detectron2 supuso un nuevo requisito a nivel de hardware: una unidad de procesamiento gráfico (GPU) potente. Aunque si bien es posible la ejecución en modo sólo CPU, su rendimiento óptimo, en términos de tiempo de ejecución, se puede alcanzar solamente con el uso de GPU.

Durante el curso de esta investigación no fue posible conseguir localmente el hardware necesario para la ejecución de la aplicación, de modo que no se pudo disponer de un ambiente local. Esto llevó a la configuración de ambientes remotos, que generalmente son virtuales, y se debe solicitar una cuota de recursos.

La primer opción fue utilizar Google Cloud Platform. Este servicio ofrece una interfaz con muchas opciones para configurar uno o más ambientes virtuales en los que se posee permisos de administrador y es posible configurar a gusto el hardware: CPU, RAM, IPs fijas, etc. La desventaja que presenta es que la utilización del servicio tiene costo.

Como segunda opción, se evaluó utilizar la infraestructura del Centro Nacional de Supercomputación (ClusterUY) (Nesmachnow y Iturriaga, 2019). Esta plataforma de computación de alto desempeño posee la capacidad de gestionar en forma coordinada múltiples recursos de cómputo. El Centro cuenta con 1120 núcleos de cómputo CPU de última generación Intel Xeon-Gold 6138 2.00GHz, 3,5 TB de memoria RAM y 100.352 núcleos de cómputo GPU Nvidia Tesla P100 con 12Gb de memoria interconectados por una red de alta velocidad Ethernet de 10 Gbps.

En la plataforma ClusterUY se debe registrar un usuario con el cual se dispondrá de un único ambiente básico. Una vez que se ingresa a este ambiente, es posible solicitar de manera dinámica el acceso a nodos con mayores recursos, que pueden variar en CPU, GPU y RAM. También se dispone de una modalidad de trabajo por lotes en el que se puede agendar la ejecución de una secuencia de comandos bash, y ser notificado en su culminación. Este servicio es gratuito para los estudiantes de la Universidad de la República. Finalmente se seleccionó esta plataforma para el desarrollo y evaluación del sistema dado que contaba con todos los requisitos tanto a nivel de hardware así como también de software de base.

3.2. Arquitectura y diseño

La arquitectura del sistema implementado está basada en el trabajo de Gilewski (Gilewski, 2019), el cual consiste en un pipeline modular para el procesamiento de imágenes. Un pipeline de procesamiento de imágenes es un conjunto de tareas ejecutadas en secuencia con un orden predefinido para transformar la imagen en cierto modo o para extraer alguna característica de interés. Algunas tareas de ejemplo son: transformaciones de imagen (como traslación, rotación, ajuste de tamaño, etc), mejoramiento de la imagen, extracción de regiones de interés (RoI), clasificación de imagen, entre otras. El resultado final puede ser una nueva imagen o solo un archivo de texto con la información sobre la imagen.

A modo de ejemplo, se puede asumir que se tiene un conjunto de imágenes en un directorio local, se quiere detectar objetos en ellas, guardar los resultados individualmente y, adicionalmente, guardar un resumen de los resultados. Un pipeline que llevara a cabo estas tareas se vería como el de la figura 3.1, donde cada recuadro verde representa un módulo de procesamiento.

Si se quisiera mejorar el ejemplo anterior para que procese videos, se debería agregar algunos pasos al pipeline. Primero se debe capturar el video, por lo que la primer tarea será generar una secuencia de imágenes desde un archivo o una fuente de video (fotograma por fotograma). Luego se pasará a la detección de objetos en cada fotograma y su almacenamiento. Opcionalmente, se podría anotar el video con, por ejemplo, un recuadro alrededor de los objetos detectados, mostrar el video anotado y almacenarlo. Finalmente, se almacena la información y se guarda en un archivo con información sobre los objetos

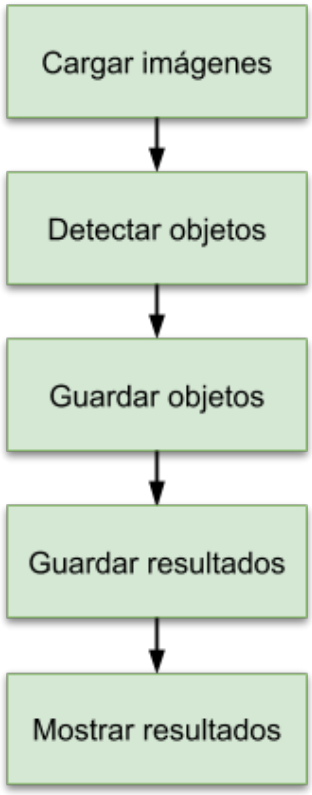


Figura 3.1: Pipeline para la detección de objetos en imágenes.

detectados (coordenadas de los recuadros, porcentaje de confianza, etc). La figura 3.2 muestra el pipeline modificado para el procesamiento de video (se representan en amarillo los pasos opcionales).

Un aspecto a destacar es que los pasos Detectar objetos, Guardar objetos y Mostrar resultados en las figuras 3.1 y 3.2 representan el mismo módulo. Es posible reutilizar y reordenar los módulos mientras se mantengan los parámetros de entrada y de salida. Este aspecto de la arquitectura permite la reutilización de código y facilita la depuración del código.

El pasaje de información entre módulos es un aspecto clave en esta arquitectura. En ese aspecto Gilewski (2019) propuso el uso de generadores de Python. Según la documentación, los generadores se describen como "funciones que permiten declarar funciones que se comportan como un iterador, que pueden ser utilizadas en un bucle for" (Python, 2020). En otras palabras, un generador es una función que retorna un objeto (iterador) sobre el cual se puede iterar (un valor a la vez).

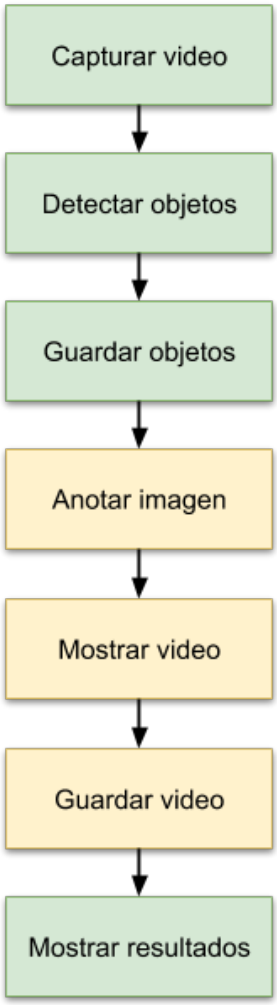


Figura 3.2: Pipeline para la detección de objetos en videos.

El código fuente de la clase principal Pipeline se puede ver en el listado 3.1. Pipeline es una clase abstracta que contiene una función generadora (generator) que por defecto pasa los datos (del generador anterior) a través de la función filtro (filter) y la función transformadora (map). La función filtro permite filtrar los datos que pasan por el pipeline. La función transformadora permite manipular los datos en el pipeline o cambiar el estado del paso. Al sobrescribir el operador lógico "o" (__or__), es posible invocar el pipeline de una forma similar a como se hace en los sistemas Unix. El listado 3.2 muestra un ejemplo de invocación de un pipeline, donde *cargar_imagenes*, *detectar_objetos*, *guardar_objetos*, *guardar_resultados*, *mostrar_resultados* son instancias de subclases de *Pipeline*.

Listado 3.1: Código fuente de la clase Pipeline

```
class Pipeline(object):
    def __init__(self):
        self.source = None

    def __iter__(self):
        return self.generator()

    def generator(self):
        while self.has_next():
            data = next(self.source) if self.source else {}
            if self.filter(data):
                yield self.map(data)

    def __or__(self, other):
        other.source = self.generator()
        return other

    def filter(self, data):
        return True

    def map(self, data):
        return data

    def has_next(self):
        return True
```

Listado 3.2: Ejemplo de uso del pipeline

```
pipeline = cargar_imagenes
    | detectar_objetos
    | guardar_objetos
    | guardar_resultados
    | mostrar_resultados

for i in pipeline:
    pass
```

3.3. Implementación

En esta sección se exponen los detalles de implementación de cada módulo, la función que cumplen y los parámetros de entrada y salida.

3.3.1. Descripción general del pipeline

La arquitectura modular permitió realizar un proceso de desarrollo progresivo, partiendo de algunos módulos generales en la propuesta de Gilewski (2019), a la incorporación de módulos específicos para la recopilación de los datos útiles para esta investigación.

La figura 3.3 muestra la arquitectura final del pipeline destinado al análisis de videos de tráfico que consta de doce módulos. Los módulos pueden ser agrupados en tres categorías según el tipo de función que realizan. En primer lugar, están los módulos de recolección de datos, en color verde en la figura. Luego, en color amarillo, están los módulos dedicados al análisis de los datos recolectados. Finalmente, los módulos en color rojo tienen el objetivo de producir los datos de salida del pipeline.

El pipeline propuesto cuenta con más de 40 parámetros que permiten controlar diferentes aspectos de procesamiento en cada módulo, donde algunos son requeridos y de no especificarse el módulo no se activa. Entre los parámetros más relevantes del pipeline se encuentran la ruta la video de entrada (–input), el archivo de configuración de la línea base de parámetros de Detectron2 (–d2-config-file), las clases a detectar (–include-classes) y el algoritmo de seguimiento (–tracker). La lista completa de parámetros se encuentra en el Apéndice 1.

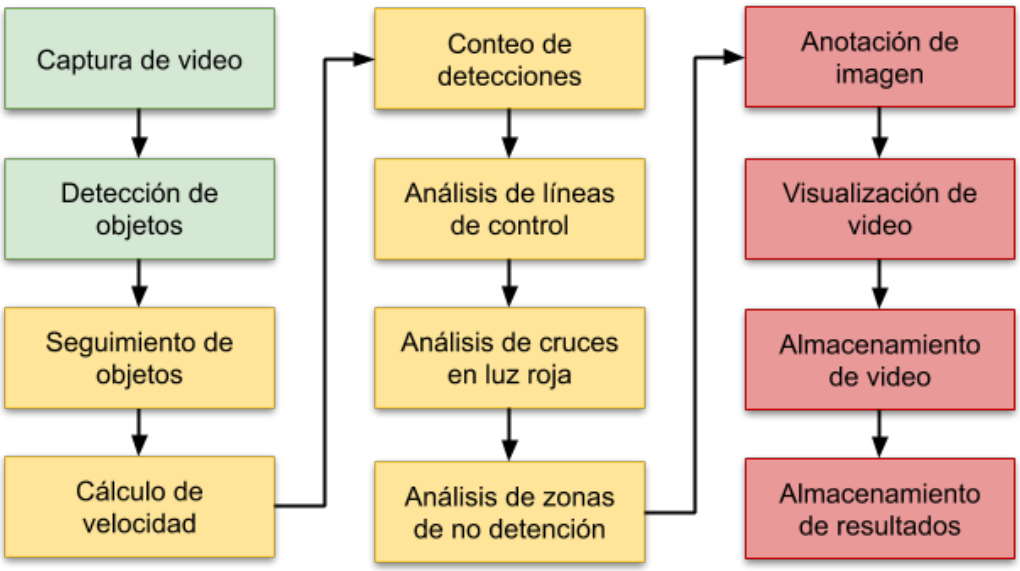


Figura 3.3: Pipeline para el análisis de videos de tráfico.

3.3.2. Captura de video

El módulo de captura de video tiene el objetivo de producir los fotogramas que serán utilizados por el resto del pipeline. Concretamente, se encarga de capturar un flujo de video (archivo local o de alguna fuente directa), utilizando procesamiento multi-hilado para la lectura de los fotogramas del video.

Durante su inicialización este módulo utiliza varias facilidades de OpenCV para capturar el video desde el archivo o cámara web, obtener el número de fotogramas por segundo (FPS) y el tamaño del fotograma. Luego, su método generador produce uno a uno el contenido de cada fotograma junto con algunos metadatos como el número de fotograma y un identificador de imagen.

Por ser este el primer módulo del pipeline, tiene la responsabilidad de inicializar el objeto de transferencia de información acumulativa (OTI). Este objeto será utilizado por todos los módulos para leer información expuesta allí por módulos anteriores en el pipeline y grabar información propia.

El módulo de captura de video guarda los siguientes campos en el OTI:

- número de fotograma: en la secuencia de fotogramas leídos de la fuente.
- imagen: objeto creado por la funcionalidad de lectura de fotogramas del módulo de captura de video de OpenCV.
- anotaciones: objeto utilizado para producir anotaciones a nivel general en el fotograma. Las anotaciones son descritas en detalle en la sección 3.3.10.

Adicionalmente, es posible indicar al módulo si anotar el video de salida con el número de fotograma mediante un parámetro que, si se especifica, agrega la información necesaria en la sección de anotaciones del OTI.

3.3.3. Detección de objetos

El módulo de detección de objetos es el encargado de realizar las detecciones de objetos fotograma a fotograma. Se dispone de dos implementaciones, ambas basadas en Detectron2, por lo que los parámetros configurables son los parámetros para Detectron2: archivo de configuración de línea base de parámetros, archivo de pesos, y umbral de confianza. Las implementaciones se diferencian en que una de ellas realiza detección síncrona y la otra asíncrona. A su vez, la detección asíncrona permite el procesamiento secuencial y paralelo de fotogramas.

La entrada del módulo para la predicción de detecciones es el contenido de un fotograma (y opcionalmente sus metadatos) y en su función transformadora realiza las predicciones. La salida consiste en un listado de propiedades pertenecientes a los objetos detectados en el fotograma:

- recuadro: coordenadas del recuadro de la detección en el fotograma. Es una lista con dos duplas que indican los puntos superior izquierdo e inferior derecho de la detección. La figura 3.4 muestra el espacio de coordenadas utilizado por Detectron2 cuyo origen se encuentra en la esquina superior izquierda de la imagen.
- máscara: máscara de segmentación de la detección. Se trata de una grilla de valores booleanos que indica en qué píxeles se encuentra la detección.
- clase: índice de la clase en el listado de nombres de clases que depende la línea base de parámetros de Detectron2 utilizada.
- nombre de clase: nombre de la clase del objeto.
- puntaje: indica el grado de confianza en la clase del objeto asignada. Es un valor porcentual de cuán seguro está el algoritmo en que la clase asignada es la correcta.
- etiquetas: lista de anotaciones que tendrá el objeto detectado. En el caso del módulo de detección de objetos agrega la clase y el puntaje en valor porcentual.

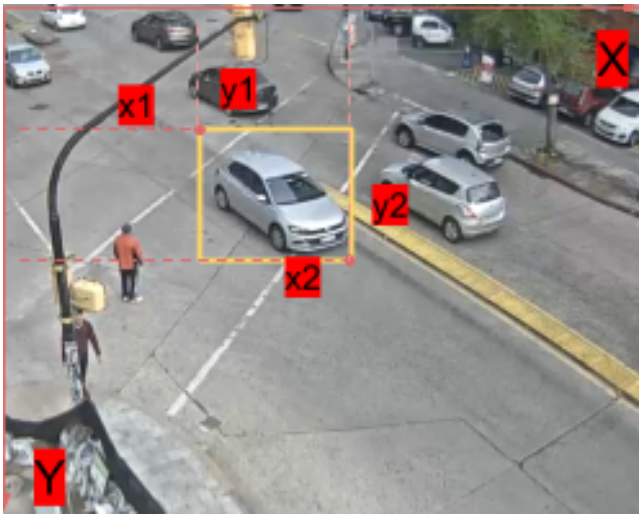


Figura 3.4: Espacio de coordenadas de detección Detectron2.

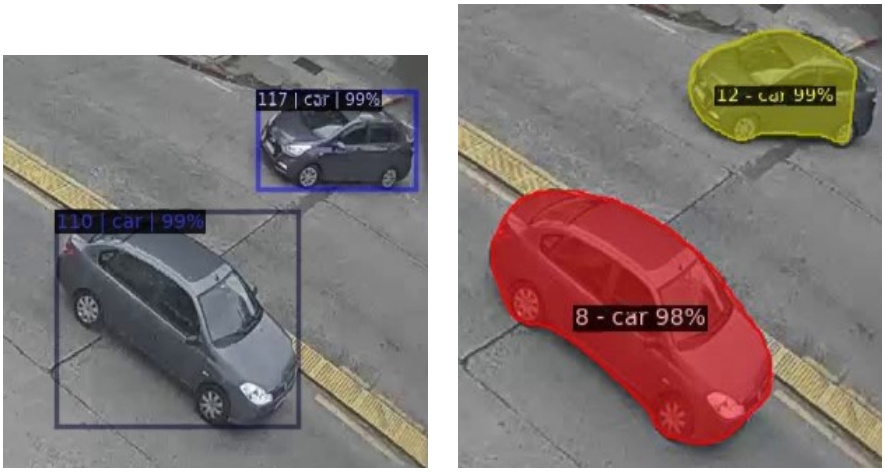
Se realiza un procesamiento luego de la predicción para transformar los resultados obtenidos del procesamiento de Detectron2 a diccionarios, para no acoplar el funcionamiento del resto del pipeline a esta clase objetos dependientes de la biblioteca. En investigaciones futuras podría reemplazarse Detectron2 por otra biblioteca de detección sin afectar el resto de los módulos, siempre y cuando se mantenga el mismo formato de salida.

Durante el procesamiento posterior a la predicción también se filtran las clases de interés. Los nombres de clase de los objetos con los que interesa trabajar se pasan por parámetro al módulo de detección de objetos. Este parámetro es opcional y, si no se utiliza, no se realiza el filtro y se continúa trabajando con todos los objetos detectados.

Adicionalmente, se puede configurar el módulo de detección de objetos para generar una máscara del fotograma que indique la región de interés. Este comportamiento se indica mediante un parámetro que consiste en un conjunto de puntos que delimitan una región dentro del fotograma. Cuando este parámetro se encuentra presente en el módulo de detección de objetos la detección se realiza solamente dentro de la región, en lugar de hacerlo en todo el fotograma. La máscara de región de interés permite ahorrar tiempo de cálculo de Detectron2 en el análisis de regiones que pueden no ser de interés.

La disponibilidad del objeto de máscara de cada detección depende de la configuración de línea base de Detectron2 utilizado. La detección con máscara de segmentación requiere un archivo de pesos con segmentación de instancias que es utilizado para, además de calcular el recuadro, calcular la máscara de segmentación de la detección.

La figura 3.5 muestra un fragmento de fotograma anotado con recuadros y con máscaras. Se observa que la detección con máscara delimita más precisamente los objetos detectados.



(a) Detección con recuadros (b) Detección con máscara

Figura 3.5: Comparación entre recuadros y máscaras de segmentación.

3.3.4. Seguimiento de objetos

El módulo de seguimiento de objetos tiene la tarea de asignar un identificador a cada objeto detectado mientras se mantenga en el fotograma. Su objetivo consiste en, por un lado, asignar identificadores a los nuevos objetos que parecen en el video y, por otro lado, hacer que el identificador se mantenga asignado al mismo objeto durante toda su permanencia a través de los distintos fotogramas.

La implementación de este módulo está basada en el trabajo de Move-lab (2018) llamado node-moving-things-tracker (NMTT) y, como se mencionó en la sección 3.1.1, este algoritmo está implementado en un ambiente Node, por lo cual es necesario establecer una interfaz para poder integrarlo con el módulo. Para lograr este objetivo se desarrolló un servidor HTTP llamado tracking-server, implementado en Node, con dos servicios, uno de configuración y otro que accede a la biblioteca NMTT para invocar las funciones de seguimiento.

La figura 3.6 muestra un diagrama de los componentes del módulo de seguimiento de objetos. Desde el módulo de seguimiento se invoca al servicio Node que recibe como parámetros de entrada las detecciones que se obtuvieron en un fotograma y un número de fotograma. Como respuesta, el servicio genera un identificador para cada una de las detecciones enviadas.

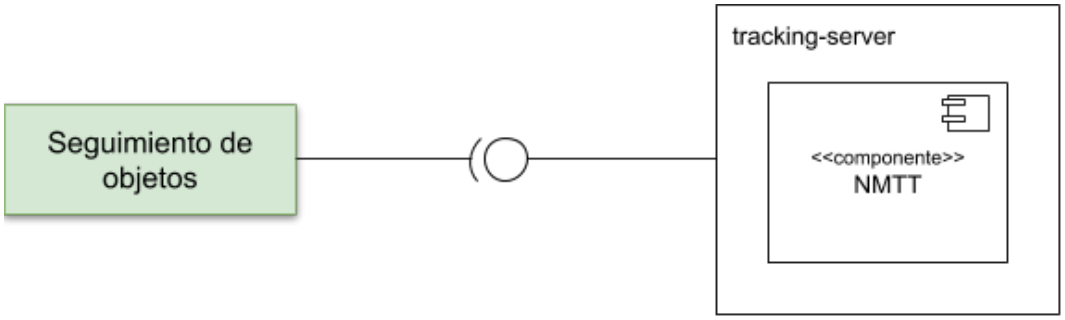


Figura 3.6: Interacción entre el módulo de seguimiento y el servidor de seguimiento.

A nivel de configuración se tienen dos parámetros que permiten controlar la coincidencia de un objeto detectado en un fotograma reciente y en los anteriores. Por un lado se tiene el número de fotogramas de tolerancia sin coincidencia, que es la cantidad de fotogramas que se espera mientras un objeto no es detectado antes de considerar que ya no se encuentra en la imagen. Un valor pequeño de este parámetro puede llevar a perder el seguimiento de algunos objetos debido a una oclusión temporal, mientras que un valor muy grande puede llevar a detectar más falsos positivos. Por otro lado, el límite IOU define el umbral de coincidencia de áreas. El valor máximo 1 espera un solapamiento total mientras que 0 significa que puede no haber solapamiento.

NMTT espera las coordenadas de las detecciones en el formato YOLO: valores del punto central (x, y), ancho w y largo del recuadro h, tal como se presenta en la figura 3.7. Por esta razón se realiza una transformación de las coordenadas de la salida de Detectron2 a este nuevo formato. Las fórmulas utilizadas para la transformación se presentan en la ecuación 3.1.

$$\begin{aligned} w &= x_2 - x_1 \\ h &= y_2 - y_1 \\ x &= w/2 + x_1 \\ y &= h/2 + y_1 \end{aligned} \tag{3.1}$$

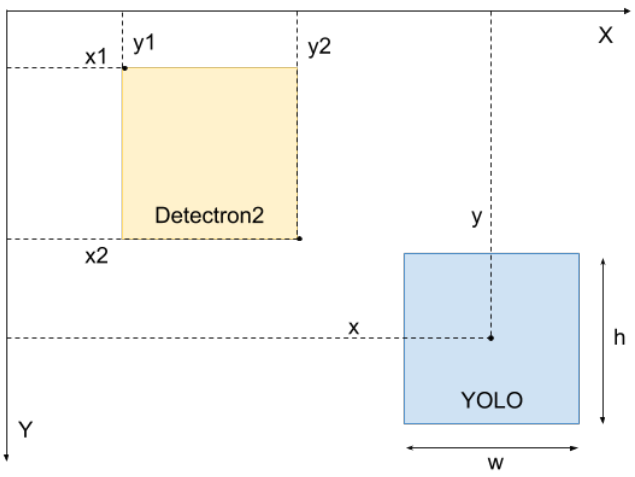


Figura 3.7: Coordenadas de recuadros Detectron2 - YOLO

La salida de del módulo de seguimiento de objetos consiste en la modificación de la lista de detecciones en el OTI para asignar el identificador calculado y una etiqueta para mostrar luego. Un valor nulo es asignado en los casos en el que el algoritmo no es capaz de asignar un identificador.

La figura 3.8 muestra un ejemplo de seguimiento de vehículo. En este caso la detección de interés informada por el módulo de detección fue asignada al identificador 26 en el fotograma 104 del video de ejemplo. Unos fotogramas más adelante, en el fotograma 137, se ve el mismo objeto real también con el identificador 26 aún cuando la detección fue ocluida por un semáforo en los fotogramas intermedios. Se observa que aunque el algoritmo de asignación de color a las instancias de detección aplica un método más simplificado de seguimiento para la selección del color, en este caso no fue capaz de seguir al vehículo cuando pasó por debajo del semáforo, y por ese motivo en el fotograma 104 el vehículo tiene color púrpura y en el 137 tiene color verde.

(a) Detección con identificador 26 en el fotograma 104

(b) Detección con identificador 26, en el fotograma 137

Figura 3.8: Seguimiento de detecciones a través de fotogramas con oclusión.

3.3.5. Cálculo de la velocidad de las detecciones

El cálculo de velocidad de las detecciones muestra una estimación de la velocidad media en los fotogramas recientes, dada en píxeles por fotograma (PPF) o en píxeles por segundo (PPS).

El algoritmo de cálculo de velocidad almacena la posición del centro de cada detección y un número parametrizable de fotogramas que indica la periodicidad del cálculo. Entonces, la velocidad está dada por la norma euclídea de la primer y la última posición almacenada. En caso de que el resultado se solicite en PPF basta con esta norma y en el caso de que el resultado se solicite en PPS, se multiplica por el valor de FPS del video: $PPS = PPF \times FPS$.

3.3.6. Conteo de detecciones

El conteo de objetos detectados requiere la definición de una o más líneas sobre la imagen del vídeo. Estas líneas están dadas por dos puntos y una etiqueta que la identifica. El módulo de conteo utiliza estas líneas para contabilizar las detecciones que las crucen.

El módulo de conteo depende de las detecciones y su seguimiento. Una vez que las detecciones son procesadas por el módulo de seguimiento, estas se agregan a una estructura que mantiene cada vehículo detectado como un objeto con varias propiedades que lo identifican, como por ejemplo su identificador, posición, clasificación, entre otras. A partir de esta estructura el módulo de conteo analiza los vehículos que se superponen con las líneas de conteo. Este análisis considera la intersección del polígono de la máscara o el recuadro con respecto a las líneas definidas. Si se encuentra una superposición del vehículo con la línea, se actualiza la información del vehículo con las líneas que cruzó y el número de fotograma en el cual lo hizo.

En la figura 3.9 se puede leer en el borde izquierdo con texto blanco, la cantidad de vehículos que cruzaron por cada línea, agrupándolos según la clase. Por otro lado, también se puede leer en el texto rojo, el conteo de vehículos detectados en el fotograma actual.

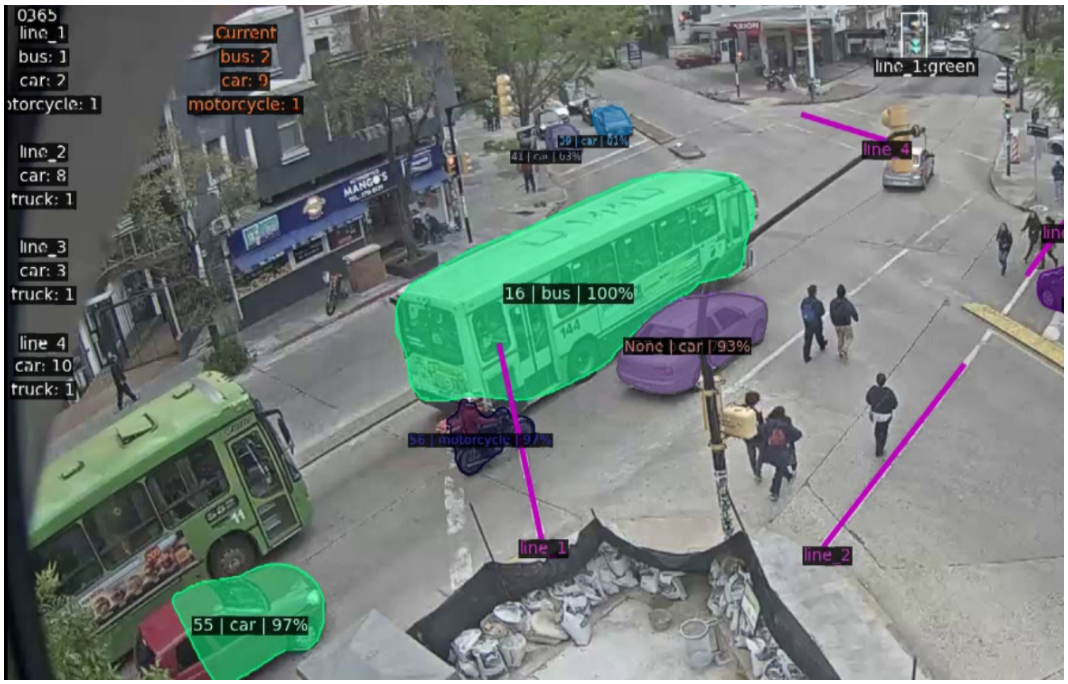


Figura 3.9: Conteo de vehículos detectados.

3.3.7. Análisis de líneas de control

El módulo de análisis de líneas de control permite realizar un análisis del flujo de tránsito mediante la definición de dos líneas de conteo y la relación entre éstas. Una vez definido el par de líneas, se debe indicar una relación entre ellas con un orden de precedencia. Así, el módulo contabiliza cada vehículo que cruza ambas líneas en el orden establecido.

El análisis de líneas de control permite la detección de infracciones que impliquen la circulación de un vehículo a contramano, giros a la izquierda o cambios de carril cerca de una esquina. Este control puede también ser usado para casos positivos, por ejemplo para contabilizar simplemente giros permitidos en cierta dirección.

Para analizar si un vehículo cruzó por un par definido de líneas se analiza la estructura donde se mantienen los vehículos identificados. Una vez procesados por el módulo de conteo, los objetos de esta estructura contienen las líneas que cruzaron y el número del primer fotograma en el que lo hicieron. A partir de esta información y de los pares de líneas definidos el algoritmo identifica cuales cruzaron ambas líneas en cada par.

3.3.8. Análisis de cruces con luz roja

El objetivo del módulo de análisis de cruces con luz roja es detectar los vehículos que cometen infracción al atravesar un cruce cuando un semáforo no lo permite. Para esto es posible definir la región donde se encuentra cada semáforo como parámetro de entrada. A su vez cada semáforo se asocia a una línea, de esta forma el sistema considera una infracción si un vehículo cruza la línea cuando el semáforo se encuentra en rojo.

Para lograr el objetivo, se analizan los semáforos definidos para determinar el color de la luz fotograma a fotograma, aplicando un algoritmo que transforma el fotograma recortado del semáforo a un modelo de color HSV (Hue, Saturation, Value por sus siglas en inglés). Luego se determina el color a partir de la proporción de píxeles rojos en comparación con la cantidad de píxeles verdes. Para que el algoritmo sea mas preciso y robusto al momento de evaluar el color de la luz se toma en cuenta la clasificación mas común en una cantidad configurable de fotogramas anteriores.

Luego de evaluar el color de todos los semáforos definidos para el fotograma, el algoritmo analiza las líneas asociadas a los semáforos y determina si estas

se encuentran habilitadas o no. Una línea habilitada es aquella cuyo semáforo asociado se encuentra en color verde. Si el semáforo asociado a una línea se encuentra de color rojo se analizan las detecciones a fin encontrar si alguna detección cruzó por la línea inhabilitada en el fotograma actual, detectando de esta forma una infracción. Una vez detectada una infracción se le agrega la etiqueta "RLR" (Red Light Run) al vehículo correspondiente.

En la figura 3.10 se puede observar un automóvil y un ómnibus que cruzaron por la línea denominada "line_1" estando el semáforo correspondiente en luz roja. El módulo agrega a los vehículos la etiqueta "RLR" indicando cual línea fue la que cruzó en infracción.

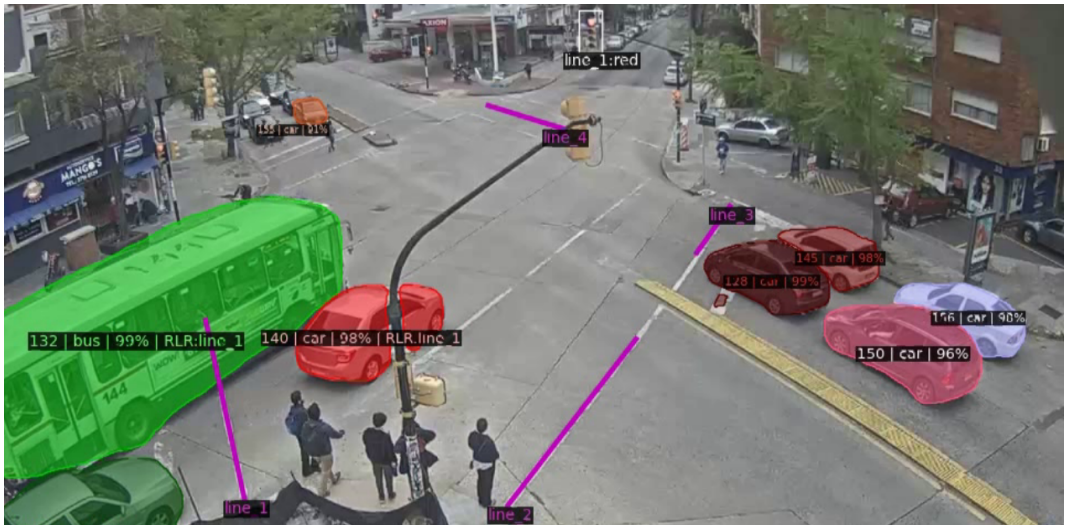


Figura 3.10: Detección de cruce con luz roja.

3.3.9. Análisis de zonas de no detención

El objetivo del módulo de análisis de zonas de no detención es detectar vehículos que se detienen en zonas no permitidas. Una zona de no detención se define como una región del fotograma, representada mediante un polígono, en la que los vehículos no deberían detenerse. Un vehículo puede considerarse detenido en un intervalo de tiempo si su velocidad media es igual a cero en ese intervalo, pero en el contexto de las detecciones es difícil obtener información que permita concluir que su velocidad es exactamente cero. Esto se debe a que al considerar un mismo vehículo real detenido, su recuadro puede variar en posición y tamaño por algunos píxeles. Por esa razón, en este caso un vehículo se considera detenido si su velocidad en PPF es menor a 1.

Para flexibilizar el estado de detención, el algoritmo considera la velocidad media en los últimos n fotogramas (valor parametrizable). Así, el módulo registra la posición de cada vehículo fotograma a fotograma en cada zona de no detención definida y cuando su velocidad media alcanza un valor menor a 1 es etiquetado como detenido.

En la figura 3.11 se puede observar un ómnibus y un automóvil con una fracción ubicada dentro de la zona marcada como "NSR1". Dado que el módulo solo considera que el vehículo está en infracción si la fracción ubicada dentro de la zona es mayor a un valor determinado parametrizable, solamente el automóvil es considerado en infracción. El módulo agrega una etiqueta a la detección indicando las zonas en las cuales se detuvo.

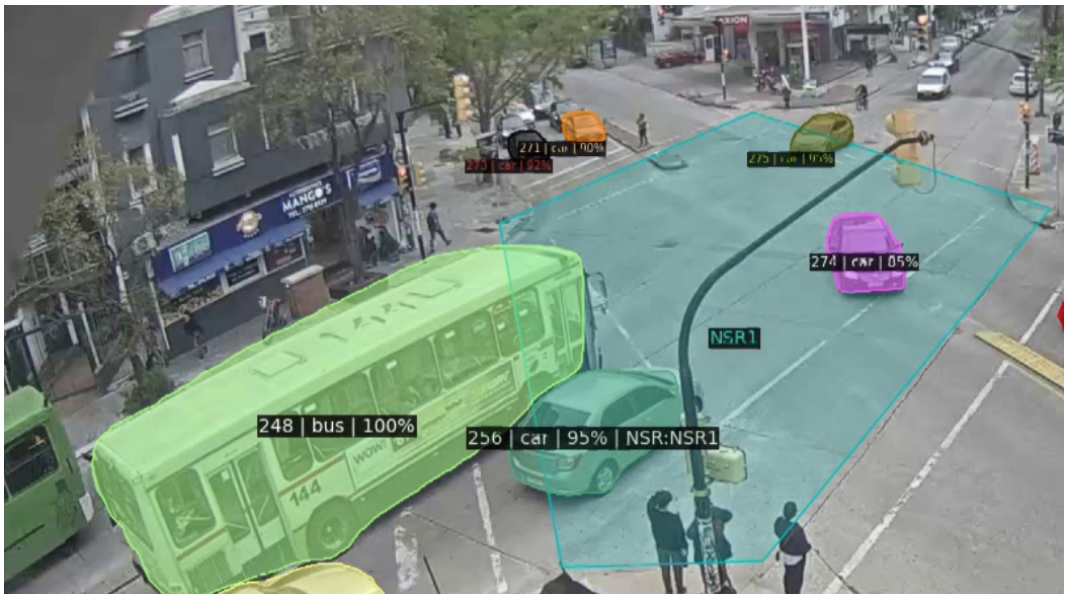


Figura 3.11: Detección de infracción en zona de no detención.

3.3.10. Anotación de imagen

La anotación consiste en la modificación de un fotograma para que muestre cierta información. El módulo de anotación de imagen se encarga de realizar las modificaciones necesarias a cada fotograma, los cuales formarán parte del video de salida. Adicionalmente, este módulo se encarga de dibujar las detecciones, recuadros o máscaras según corresponda.

La figura 3.12 muestra un ejemplo de fotograma anotado. La información anotada en el fotograma fue introducida en el OTI por los módulos del pipeline que ejecutaron previamente y pueden ser de cuatro tipos: etiquetas en las detecciones, texto de información, líneas o polígonos.

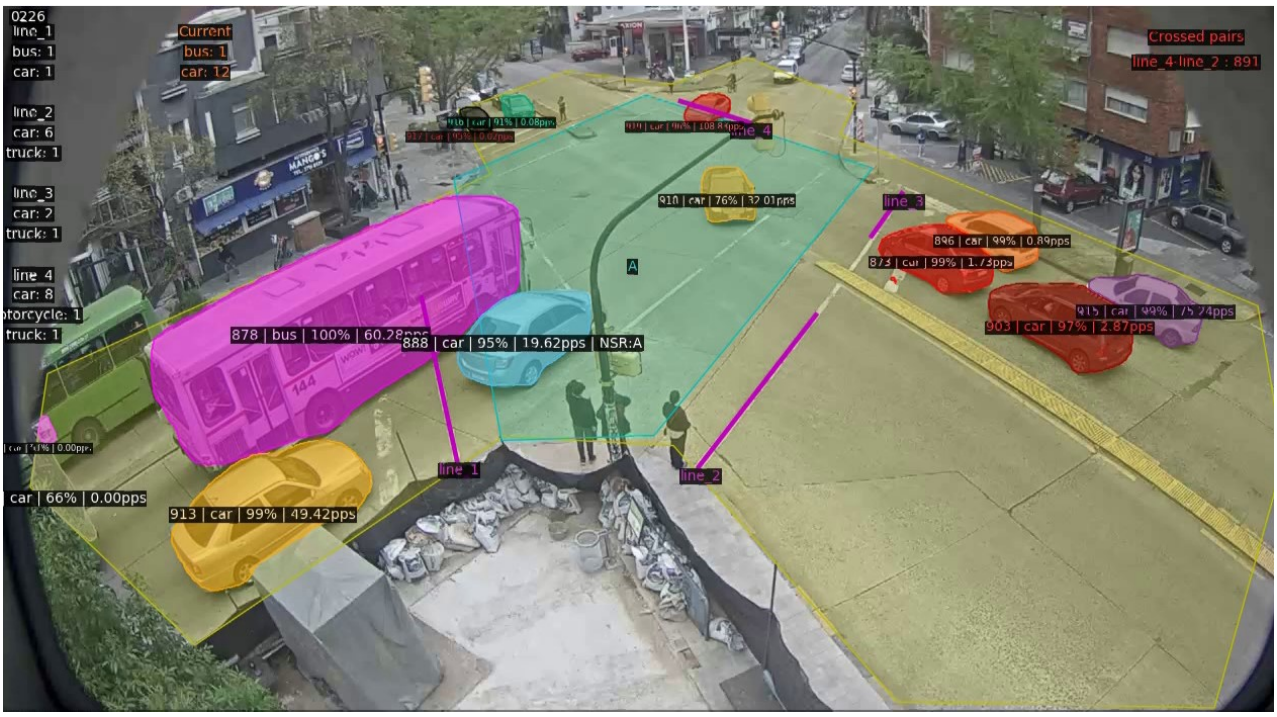


Figura 3.12: Fotograma anotado con etiquetas, líneas, polígonos y detecciones.

El proceso de anotación de imagen consiste en primero realizar una copia del fotograma. Luego, se hace uso de un módulo utilitario de Detectron2 para realizar las modificaciones. Finalmente, se utiliza OpenCV para convertir el formato RGB con el que fue leída la imagen a BGR y se almacena en el OTI.

3.3.11. Visualización de video

El módulo de visualización de video se encarga crear una ventana gráfica y mostrar el video a medida que los fotogramas de salida son producidos. Utiliza OpenCV para crear la ventana y mostrar los fotogramas.

3.3.12. Almacenamiento de video

El almacenamiento de video consiste en guardar los fotogramas en un archivo en una ruta especificada en parámetros. Para esto utiliza OpenCV y es posible especificar la cantidad de FPS y el formato de salida.

3.3.13. Almacenamiento de resultados

El objetivo del módulo de almacenamiento de resultados este módulo es mostrar la información generada en cada fotograma y la acumulativa. Utiliza un sistema de registro basado en JSON.

3.3.14. Interfaz para la selección de puntos

Durante el desarrollo del sistema fue necesario contar con un medio para la selección de los puntos de líneas y polígonos que servirían luego como parámetros de los distintos módulos. Por esa razón se desarrolló una aplicación que permite cargar un video y seleccionar líneas, rectángulos y polígonos en general y que tras la selección de la figura muestra el listado de puntos en la consola.

Capítulo 4

Evaluación de la solución

En este capítulo se presentan las validaciones del sistema realizadas. Detallando el ambiente de ejecución, el conjunto de datos de pruebas utilizado y la configuración de cada prueba. Finalmente, se muestran los resultados obtenidos en términos de tiempos y precisión.

4.1. Ambiente de ejecución

El conjunto de pruebas fue ejecutado completamente en un ambiente virtual alojado Centro Nacional de Supercomputación (ClusterUY) (Nesmachnow y Iturriaga, 2019). En esta plataforma fue posible reservar dinámicamente los recursos necesarios para la ejecución de trabajos. La tabla 4.1 muestra las principales características del ambiente en el cual fueron ejecutadas las pruebas.

Tabla 4.1: Características del ambiente de ejecución en ClusterUY.

Característica	*Valor*
Procesador	Xeon Gold 6138
# núcleos	40
# hilos de núcleo	80
Memoria	8 GB
GPU	NVIDIA P100
Disco	300 GB SSD

4.2. Descripción de los datos de prueba

Para la ejecución de las pruebas se utilizaron cuatro videos de dos ubicaciones diferentes, en dos momentos del día distintos. Las grabaciones corresponden a lugares de la ciudad de Montevideo y fueron obtenidos mediante la Unidad de Acceso a la Información de la Intendencia de Montevideo.

Una de las ubicaciones corresponde a la intersección de la avenida 8 de Octubre con la avenida Garibaldi y corresponden a las grabaciones del día 30 de julio de 2020 a las 8:30 (G8) y a las 19:30 (G19). La segunda ubicación es en la Rambla Presidente Wilson en la intersección con la avenida Sarmiento y las grabaciones fueron tomadas el día 30 de julio de 2020 a las 8:30 (R8) y a las 19:30 (R19).

La figura 4.1 muestra los escenarios de los videos utilizados y la tabla 4.2 resume sus principales características. Las grabaciones fueron realizadas con una cámara modelo AXIS P1365Mk II que captura hasta 60 fotogramas por segundo.

Tabla 4.2: Características de los videos de prueba.

Ref.	*Formato*	*Resolución*	*# fotogramas*	*Tasa*	*Duración*
G8	MP4	1280×720 píxeles	2393	8 FPS	5 minutos
G19	MP4	1280×720 píxeles	2398	8 FPS	5 minutos
R8	MP4	1280×720 píxeles	5998	20 FPS	5 minutos
R19	MP4	1280×720 píxeles	5997	20 FPS	5 minutos

Las grabaciones seleccionadas dan muestra de un flujo de tráfico representativo de la ciudad en horas pico tanto en la mañana así como también en la noche. En los videos, el ángulo de la cámara es tal que el flujo de vehículos de norte a sur ocasionalmente ocluye los vehículos que se desplazan de oeste a este. Además de esto, las grabaciones también muestran cruces con múltiples semáforos que hace que los vehículos se detengan y se acumulen produciendo también situaciones de oclusión entre vehículos. Este tipo de situaciones es un factor relevante a considerar ya que presentan una mayor dificultad en detectar y realizar el seguimiento de los vehículos ocluidos.

(a) G8

(b) G19

(c) R8

(d) R19

Figura 4.1: Escenarios de los videos utilizados para las pruebas

4.3. Métricas de evaluación

Para la evaluación del rendimiento del sistema se aplicaron medidas estadísticas. Para las pruebas que involucraron tareas de clasificación se consideraron las métricas propuestas por Sokolova y Lapalme (2009). En base a este trabajo, se utilizaron las siguientes métricas: verdadero positivo (vp), indica la cantidad de ocurrencias que el modelo predice correctamente la clase; verdadero negativo (vp), indica la cantidad de ocurrencias correctamente reconocidas que no pertenecen a la clase; falso positivo (fp), la cantidad de ocurrencias que el modelo predice incorrectamente la clase; falso negativo (fn), ocurrencias que no fueron reconocidas.

Con los valores de las métricas se obtiene un conjunto medidas de interés para la evaluación de algoritmos de detección. La exactitud media (ecuación 4.1) indica el porcentaje total de elementos clasificados correctamente, esta métrica es un indicador de la efectividad general del algoritmo. La tasa de error (ecuación 4.2) indica el porcentaje de elementos clasificados incorrectamente. La precisión (ecuación 4.3) indican el porcentaje de resultados relevantes. Por último, la exhaustividad (ecuación 4.4) indica un porcentaje del total de resultados relevantes correctamente clasificados.

$$em = \frac{vp + vn}{vp + vn + fp + fn} \tag{4.1}$$

$$te = \frac{fp + fn}{vp + vn + fp + fn} \tag{4.2}$$

$$p = \frac{vp}{vp + fp} \tag{4.3}$$

$$r = \frac{vp}{vp + fn} \tag{4.4}$$

En el caso de los algoritmos de clasificación se obtienen medidas análogas como una generalización de las anteriores al considerar l clases. Las generalizaciones de la exactitud media, la tasa de error, la precisión y la exhaustividad se muestran en las ecuaciones 4.5 a 4.8.

$$EM = \frac{\sum_{i=1}^{l} \frac{vp_i + vn_i}{vp_i + vn_i + fp_i + fn_i}}{l} \tag{4.5}$$

$$TE = \frac{\sum_{i=1}^{l} \frac{fp_i + fn_i}{vp_i + vn_i + fp_i + fn_i}}{l} \tag{4.6}$$

$$P = \sum_{i=1}^{l} \frac{vp_i}{\sum_{i=1}^{l} vp_i + fp_i} \tag{4.7}$$

$$R = \sum_{i=1}^{l} \frac{vp_i}{\sum_{i=1}^{l} vp_i + fn_i} \tag{4.8}$$

En la evaluación de los algoritmos de seguimiento se utilizaron las métricas propuestas por Leal-Taixé *et al.* (2015) quienes definieron métricas en conjunto con un juego datos públicos para probar algoritmos de seguimiento. Algunas de las métricas estándar definidas previamente tienen una connotación diferente en este caso: falsos positivos (fp) indica el número de ocurrencias en las cuales se realiza una detección pero no hay un objeto en dicha posición; falsos negativos (fn) indica el número de ocurrencias en las cuales un objeto existente no se detecta.

Adicionalmente, se deben considerar dos métricas que son aplicables solo en el contexto de seguimiento de objetos. Por un lado, el cambio cambio de identidad (IdSw por sus siglas en inglés, Identity Switch) que indica el número de ocurrencias en las cuales a un objeto ya visto se le asigna un nuevo identificador. Por otro lado, la exactitud del seguimiento de objetos múltiple (MOTA por sus siglas en inglés, Multiple Object Tracking Accuracy), que es un indicador del rendimiento del algoritmo de seguimiento que involucra las tres fuentes de error detalladas anteriormente. Esta métrica es definida en la ecuación 4.9, donde t representa el número de cuadro, fn_t el número de falsos negativos, fp_t el número de falsos positivos, id_s el número de cambios de identidad y g_t el número de objetos presentes en el cuadro t.

$$MOTA = 1 - \frac{\sum_t (fn_t + fp_t + id_s)}{\sum_t g_t} \tag{4.9}$$

4.4. Evaluación del sistema

En esta sección se describe el proceso de evaluación realizado, el enfoque tomado, las condiciones de cada validación y los resultados obtenidos.

4.4.1. Enfoque de la evaluación

La validación realizada siguió un enfoque modular que permitió obtener resultados de la eficacia de cada módulo de forma individual, aunque también se tuvo en cuenta que el rendimiento de los primeros pasos del pipeline condicionan la eficacia de los que se ejecutan luego. Por ejemplo, la mayoría de los módulos de análisis de datos dependen del módulo de detección de objetos y de los identificadores asignados por el módulo de seguimiento de objetos. Por esta razón, en la evaluación de cada módulo se estableció dos conjuntos de parámetros: uno fijo que se repite en los diferentes análisis y uno dinámico.

En cada caso de validación se especifica la lista de parámetros con sus valores. Como el sistema cuenta con más 40 parámetros, solo se mencionan los relevantes de cada caso y de no mencionarse considera su valor por defecto. La lista de parámetros completa con sus valores por defecto se encuentra en el apéndice 1.

4.4.2. Detección de objetos

La validación del módulo de detección de objetos consistió en tomar medidas de la eficacia y el tiempo de ejecución. Se consideraron parámetros relacionados con la detección como la línea base de configuración de Detectron2 y el umbral de confianza de la detección.

La línea base de parámetros determina un conjunto de parámetros entre los cuales se encuentran el tipo de red troncal a ejecutar y el modelo de pesos. Estas líneas base forman parte del Model Zoo de Detectron2 (Wu *et al.*, 2019), una colección de redes entrenadas con esta biblioteca y con rendimiento documentado. De este conjunto de modelos se seleccionaron los que presentaron mejores resultados de precisión media para la detección con recuadros y con máscaras.

La tabla 4.3 presenta las principales características de las líneas base de configuración utilizadas. Las redes troncales seleccionadas son ResNet-101+FPN (R101-FPN) que es una Faster R-CNN y ResNeXt-101+FPN (X101-FPN) que es una Mask R-CNN. El modelo de pesos R101 es una adaptación del modelo ResNet-101 original (He *et al.*, 2016) y X-101-32x8d es un modelo ResNeXt-101-32x8d entrenado con Caffe2 (Jia *et al.*, 2014). El umbral de confianza de la detección permite descartar aquellos objetos detectados en los que el valor de su función de puntaje es menor a este umbral.

Tabla 4.3: Características de las líneas base de configuración para Detectron2 utilizadas para la evaluación experimental de la detección de objetos.

	R101-box	*X101-box*	*R101-mask*	*X101-mask*
Red troncal	R101-FPN	X101-FPN	R101-FPN	X101-FPN
Modelo de pesos	R101	X-101-32x8d	R101	X-101-32x8d
Utiliza máscaras	no	no	sí	sí

La tabla 4.4 muestra las configuraciones ejecutadas y las combinaciones de parámetros utilizadas en cada caso.

Tabla 4.4: Configuraciones de ejecución para evaluación de detección.

Configuración	*Línea base*	*Umbral*
R101-box-03	R101-FPN Faster R-CNN	0,3
R101-box-05	R101-FPN Faster R-CNN	0,5
R101-box-07	R101-FPN Faster R-CNN	0,7
X101-box-03	X101-FPN Faster R-CNN	0,3
X101-box-05	X101-FPN Faster R-CNN	0,5
X101-box-07	X101-FPN Faster R-CNN	0,7
R101-mask-03	R101-FPN Mask R-CNN	0,3
R101-mask-05	R101-FPN Mask R-CNN	0,5
R101-mask-07	R101-FPN Mask R-CNN	0,7
R101-mask-03	X101-FPN Mask R-CNN	0,3
R101-mask-05	X101-FPN Mask R-CNN	0,5
R101-mask-07	X101-FPN Mask R-CNN	0,7

Por otro lado, se evaluó también el tiempo de ejecución del pipeline con las configuraciones establecidas para el video G8. Para las configuraciones R101-box-05 y R101-box-05 se realizaron configuraciones adicionales para evaluar el rendimiento del modo asíncrono de detección.

Detectron2 permite establecer en su modo asíncrono el grado de paralelismo mediante un parámetro que indica el tamaño de una cola, la cual determina cuántos fotogramas se evalúan en paralelo. Además es posible indicar cuantos recursos de GPU están disponibles para la ejecución. En la evauación de eficiencia se evaluaron las combinaciones con tamaño de cola 3, 6 y 9, con una y dos GPUs.

A continuación se muestran los resultados obtenidos para las métricas de detección y clasificación expuestas en la sección 4.4.2 para la evaluación del módulo de detección de objetos.

Las tablas 4.5, 4.6, 4.7 y 4.8 muestran los resultados de evaluación de detección de objetos para los videos G8, G19, R8 y R19, respectivamente. Se tomaron mediciones de tres fotogramas esparcidos en el tiempo. Los resultados de detección de objetos muestran una exactitud media del 70 % para los casos que transcurren durante el día y del 45 % para los casos en la noche. Con respecto al modelo utilizado, el X101-mask obtuvo un rendimiento superior a los demás tomando como referencia nuevamente la exactitud media: 47 % (R101-box), 45 % (X101-box), 66 % (R101-mask) y 70 % (X101-mask).

Tabla 4.5: Métricas de detección obtenidas con las distintas configuraciones en el video G8.

Configuración	EM	TE	P	R
R101-box-03	0,75	0,25	0,91	0,81
R101-box-05	0,52	0,48	1,00	0,52
R101-box-07	0,32	0,68	0,95	0,32
X101-box-03	0,76	0,24	0,90	0,82
X101-box-05	0,51	0,49	1,00	0,51
X101-box-07	0,24	0,76	1,00	0,24
R101-mask-03	0,78	0,22	0,91	0,85
R101-mask-05	0,75	0,25	0,97	0,77
R101-mask-07	0,63	0,37	0,97	0,64
X101-mask-03	0,77	0,23	0,89	0,85
X101-mask-05	0,67	0,33	0,96	0,68
X101-mask-07	0,63	0,37	1,00	0,63

Tabla 4.6: Resultados de detección obtenidas con las distintas configuraciones en el video G19.

Configuración	EM	TE	P	R
R101-box-03	0,59	0,41	0,79	0,70
R101-box-05	0,40	0,60	0,90	0,42
R101-box-07	0,23	0,77	1,00	0,23
X101-box-03	0,45	0,55	0,71	0,56
X101-box-05	0,42	0,58	0,90	0,44
X101-box-07	0,23	0,77	1,00	0,23
R101-mask-03	0,58	0,42	0,70	0,77
R101-mask-05	0,55	0,45	0,70	0,72
R101-mask-07	0,53	0,47	1,00	0,53
X101-mask-03	0,63	0,37	0,79	0,77
X101-mask-05	0,60	0,40	0,93	0,63
X101-mask-07	0,55	0,45	0,96	0,56

Las tablas 4.9, 4.10, 4.11 y 4.12 muestran las métricas de evaluación de clasificación de objetos para los videos G8, G19, R8 y R19 respectivamente. También se tomaron mediciones de tres fotogramas esparcidos en el tiempo. En este caso, los resultados indican que el sistema propuesto obtuvo una exactitud media general del 85 % e indican que las configuraciones de máscara obtuvieron mejores resultados que las configuraciones que utilizan cajas delimitadoras en la mayoría de los casos (10 de 12 casos). No se detectaron diferencias de

Tabla 4.7: Resultados de detección obtenidas con las distintas configuraciones en el video R8.

Configuración	EM	TE	P	R
R101-box-03	0,77	0,23	0,85	0,88
R101-box-05	0,56	0,44	0,94	0,58
R101-box-07	0,41	0,59	0,92	0,42
X101-box-03	0,82	0,18	0,92	0,88
X101-box-05	0,52	0,48	0,93	0,54
X101-box-07	0,41	0,59	0,92	0,42
R101-mask-03	0,96	0,04	1,00	0,96
R101-mask-05	0,96	0,04	1,00	0,96
R101-mask-07	0,92	0,08	1,00	0,92
X101-mask-03	1,00	0,00	1,00	1,00
X101-mask-05	1,00	0,00	1,00	1,00
X101-mask-07	0,96	0,04	1,00	0,96

Tabla 4.8: Resultados de detección obtenidas con las distintas configuraciones en el video R19.

Configuración	EM	TE	P	R
R101-box-03	0,40	0,60	0,91	0,42
R101-box-05	0,46	0,54	1,00	0,46
R101-box-07	0,29	0,71	1,00	0,29
X101-box-03	0,36	0,64	0,90	0,38
X101-box-05	0,38	0,63	1,00	0,38
X101-box-07	0,29	0,71	1,00	0,29
R101-mask-03	0,43	0,57	0,75	0,50
R101-mask-05	0,44	0,56	0,92	0,46
R101-mask-07	0,36	0,64	0,90	0,38
X101-mask-03	0,52	0,48	0,75	0,63
X101-mask-05	0,63	0,38	1,00	0,63
X101-mask-07	0,50	0,50	1,00	0,50

rendimiento significativas entre los modelos R101 y X101 en la predicción de recuadros ni máscara de segmentación, pero X101 generó resultados ligeramente mejores en los casos nocturnos para la segmentación.

Tabla 4.9: Resultados de clasificación obtenidas con las distintas configuraciones en el video G8.

Configuración	*EM*	*TE*	*P*	*R*
R101-box-03	0,93	0,07	0,91	0,73
R101-box-05	0,87	0,13	1,00	0,48
R101-box-07	0,78	0,22	0,93	0,23
X101-box-03	0,93	0,07	0,89	0,75
X101-box-05	0,88	0,12	1,00	0,49
X101-box-07	0,79	0,21	1,00	0,24
R101-mask-03	0,94	0,06	0,90	0,80
R101-mask-05	0,93	0,07	0,97	0,72
R101-mask-07	0,90	0,10	0,96	0,58
X101-mask-03	0,93	0,07	0,89	0,79
X101-mask-05	0,91	0,09	0,96	0,66
X101-mask-07	0,91	0,09	1,00	0,61

Tabla 4.10: Resultados de clasificación obtenidas con las distintas configuraciones en el video G19.

Configuración	*EM*	*TE*	*P*	*R*
R101-box-03	0,89	0,11	0,78	0,67
R101-box-05	0,82	0,18	0,89	0,40
R101-box-07	0,75	0,25	1,00	0,23
X101-box-03	0,84	0,16	0,70	0,53
X101-box-05	0,83	0,17	0,90	0,42
X101-box-07	0,73	0,27	1,00	0,21
R101-mask-03	0,87	0,13	0,67	0,67
R101-mask-05	0,85	0,15	0,67	0,60
R101-mask-07	0,86	0,14	1,00	0,49
X101-mask-03	0,89	0,11	0,77	0,70
X101-mask-05	0,87	0,13	0,92	0,56
X101-mask-07	0,87	0,13	0,96	0,53

La figuras 4.2 y 4.3 muestran el comparativo de tiempos de ejecución con las distintas configuraciones de tamaño de cola y cantidad de recursos de GPU para el video G8. En las figuras, “Simple” indica ejecuciones sin paralelismo y “Paralelo i” indica ejecuciones con un tamaño de cola i. Fueron realizadas tres ejecuciones con cada configuración.

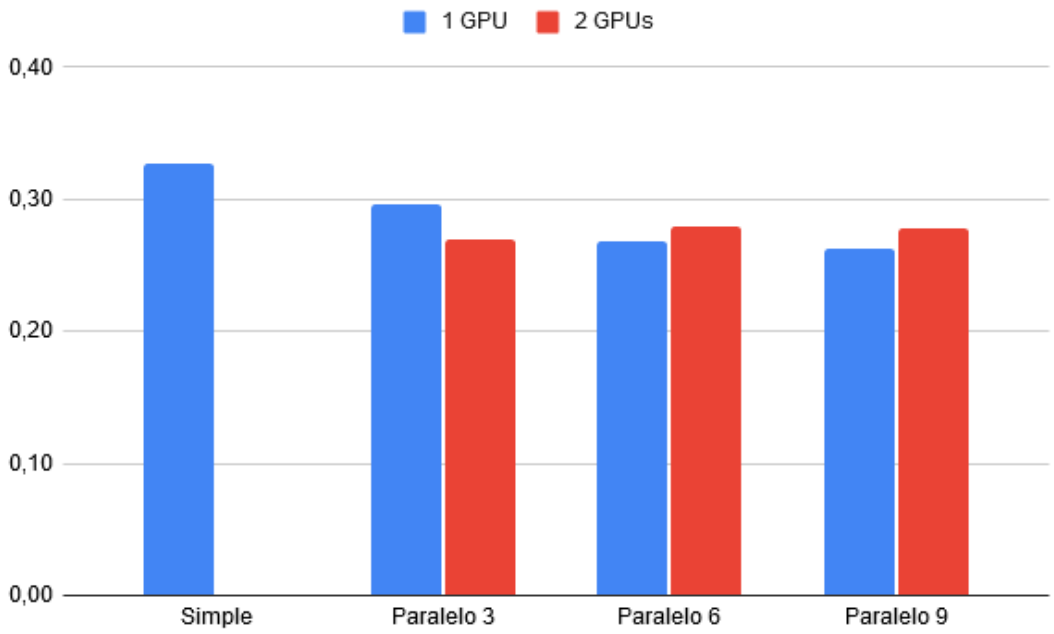
Tabla 4.11: Resultados de clasificación obtenidas con las distintas configuraciones en el video R8.

Configuración	*EM*	*TE*	*P*	*R*
R101-box-03	0,95	0,05	0,85	0,88
R101-box-05	0,90	0,10	0,94	0,58
R101-box-07	0,87	0,13	0,92	0,42
X101-box-03	0,94	0,06	0,91	0,81
X101-box-05	0,89	0,11	0,93	0,54
X101-box-07	0,73	0,27	1,00	0,21
R101-mask-03	0,99	0,01	1,00	0,96
R101-mask-05	0,98	0,02	1,00	0,92
R101-mask-07	0,98	0,02	1,00	0,88
X101-mask-03	0,98	0,02	1,00	0,92
X101-mask-05	0,98	0,02	1,00	0,92
X101-mask-07	0,98	0,02	1,00	0,88

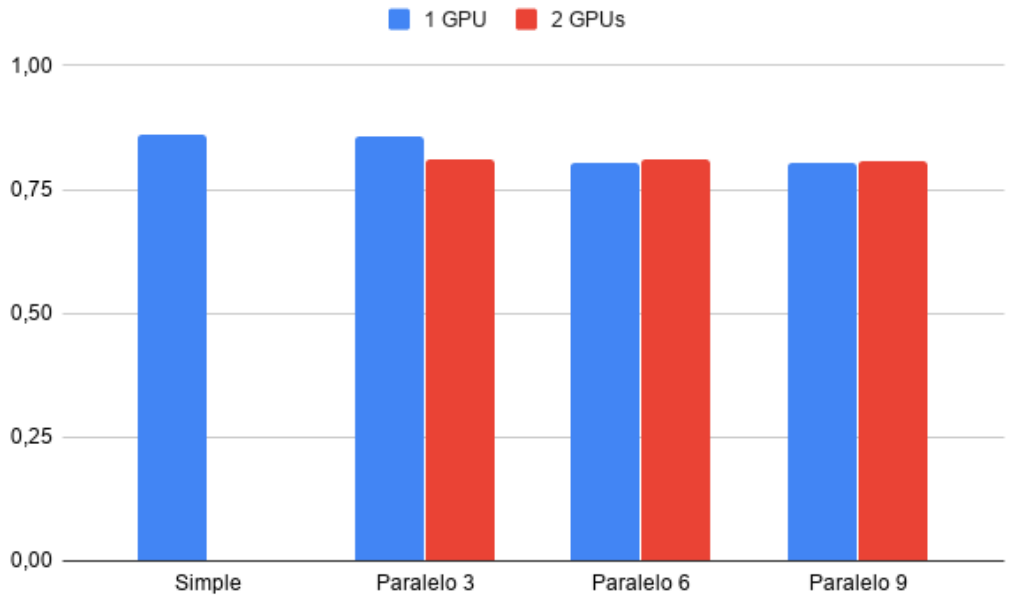
Tabla 4.12: Resultados de clasificación obtenidas con las distintas configuraciones en el video R19.

Configuración	*EM*	*TE*	*P*	*R*
R101-box-03	0,88	0,12	0,91	0,42
R101-box-05	0,89	0,11	1,00	0,46
R101-box-07	0,86	0,14	1,00	0,29
X101-box-03	0,87	0,13	0,90	0,38
X101-box-05	0,88	0,13	1,00	0,38
X101-box-07	0,86	0,14	1,00	0,29
R101-mask-03	0,89	0,11	0,75	0,50
R101-mask-05	0,89	0,11	0,92	0,46
R101-mask-07	0,87	0,13	0,90	0,38
X101-mask-03	0,89	0,11	0,75	0,63
X101-mask-05	0,93	0,08	1,00	0,63
X101-mask-07	0,90	0,10	1,00	0,50

El análisis del tiempo de procesamiento muestra que aumentar el tamaño de la cola a 6 y 9 produce una reducción leve en el tiempo global del procesamiento por fotograma con respecto a un tamaño de cola 3 y el modo sin paralelismo, con uno y dos recursos de GPU. Además, el uso de un segundo recurso de GPU no produjo mejoras en los tiempos en todos los casos. Esto puede deberse a que el tamaño de cola no es lo suficientemente grande como para utilizar correctamente el poder de procesamiento de las dos GPU. Por otro lado, al



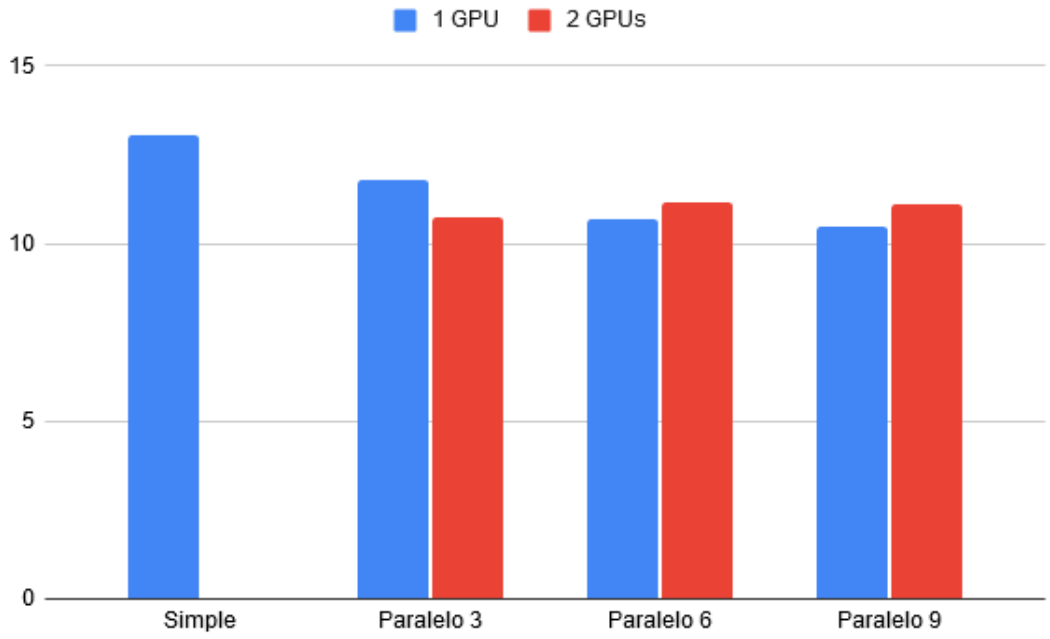
(a) Detección con recuadros



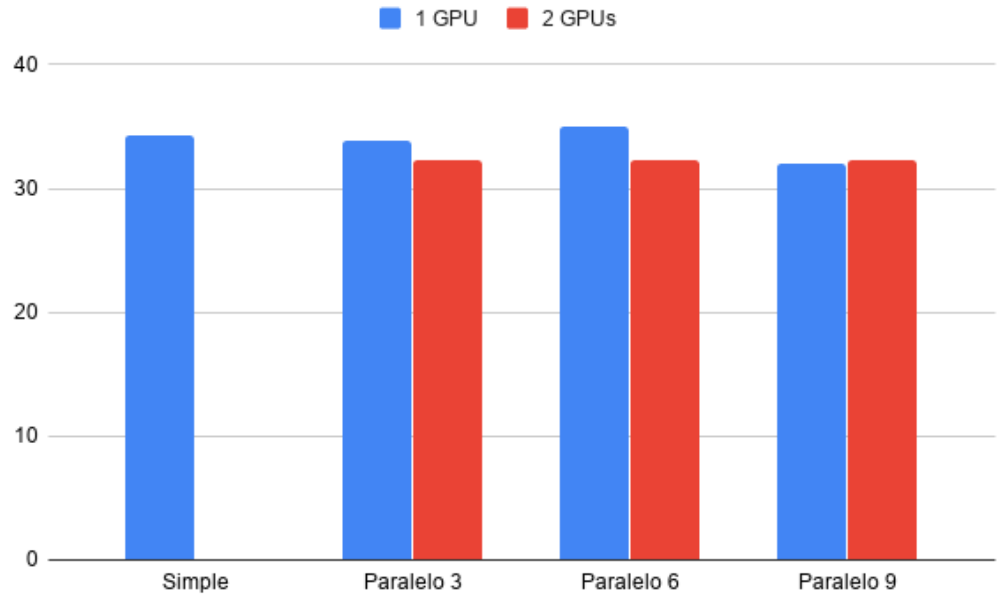
(b) Detección con máscaras

Figura 4.2: Tiempo medio de procesamiento por fotograma en segundos.

comparar los tiempos producidos por los modelos con cuadros y los modelos de segmentación, se puede ver que los modelos de segmentación insumen, en promedio, tres veces más que los de cajas delimitadoras.



(a) Detección con recuadros



(b) Detección con máscaras

Figura 4.3: Tiempo total de ejecución en minutos.

4.4.3. Seguimiento de objetos

La validación del seguimiento de objetos midió la capacidad del módulo para asignar y mantener los identificadores de los objetos. El principal parámetro

considerado fue el número fotogramas de tolerancia antes de que el algoritmo concluya que un objeto seguido ya no se encuentra en la secuencia.

Los valores de tolerancia considerados dependieron de la tasa de fotogramas del video evaluado. Se consideraron valores correspondientes a medio, uno y dos segundos ya que representan ventanas de tiempo en las que la probabilidad de un cambio de identidad entre las detecciones es baja. Esto es, 4, 8 y 16 fotogramas de tolerancia para los videos G8 y G19, y 10, 20 y 40 fotogramas de tolerancia para R8 y R19 la tabla 4.13 muestra un resumen de estas configuraciones.

Tabla 4.13: Configuraciones de ejecución para evaluación del seguimiento de objetos.

Referencia	*Video*	*Tolerancia*
G8-t04	G8	4
G8-t08	G8	8
G8-t16	G8	16
G19-t04	G19	4
G19-t08	G19	8
G19-t16	G19	16
R8-t04	R8	4
R8-t08	R8	8
R8-t16	R8	16
R19-t04	R19	4
R19-t08	R19	8
R19-t16	R19	16

El rendimiento de este módulo depende directamente de los objetos detectados por el módulo de detección. Por este motivo, se definió la configuración de detección X101-mask-05 como base para todas las evaluaciones de seguimiento ya que mostró buenos resultados en la validación de detección.

La tabla 4.14 presenta los resultados obtenidos. Las puntuaciones MOTA muestran un promedio de 85 % para los casos de luz diurna y 30 % para los casos de noche. Los resultados indican que el módulo funciona significativamente mejor en escenarios diurnos. En 3 de cada 4 casos las configuraciones con un segundo de tolerancia (8 fotogramas en G8 y G19, y 20 fotogramas en R8 y R19) tuvieron resultados ligeramente mejores con respecto a las configuraciones de medio y dos segundos. Por otro lado, el MOTA se vio afectado principalmente por los falsos negativos. En la evaluación de seguimiento, esto ocurre cuando

un vehículo existente no se detecta en un número determinado de fotogramas, por lo que no se realiza un seguimiento. En base a esta observación, mejorar el módulo de detección en malas condiciones de iluminación también mejoraría el rendimiento del seguimiento.

Tabla 4.14: Resultados obtenidos de la validación del seguimiento de objetos.

Configuración	*MOTA*
G8-t04	0,81
G8-t08	0,83
G8-t16	0,82
G19-t04	0,47
G19-t08	0,49
G19-t16	0,46
R8-t04	0,86
R8-t08	0,87
R8-t16	0,89
R19-t04	0,10
R19-t08	0,14
R19-t16	0,13

4.4.4. Conteo de detecciones

La evaluación del módulo de conteo de detecciones consistió en comparar la cantidad real de vehículos que cruzan una línea de conteo en comparación con el conteo de vehículos resultante de la ejecución del pipeline.

Durante la evaluación se utilizaron segmentos de 30 segundos de video extraídos de los cuatro vídeos del conjunto de datos de validación original. Para cada vídeo se consideraron cuatro segmentos, teniendo así un conjunto de datos total de 16 segmentos de 2 ubicaciones distintas de los cuales 8 transcurren durante el día y 8 durante la noche.

Para la ejecución de la validación se estableció la configuración X101-mask-05 para la detección de objetos. Para el modulo de seguimiento objetos se estableció una tolerancia de pérdida de 8 fotogramas para los videos G8 y G19 y 20 fotogramas para los videos R8 y R19 ya que estos fueron los que presentaron mejores resultados en la validación del seguimiento de objetos.

La tabla 4.15 presenta los resultados de la validación realizadas en el módulo de conteo. Se obtuvo una exactitud media del 89 % para los casos de luz

diurna y del 36 % para los casos de noche. Los resultados indican que el módulo de conteo funciona mejor en buenas condiciones de iluminación. En malas condiciones de iluminación el rendimiento es deficiente debido a la gran cantidad de falsos positivos. Este comportamiento se puede deber a que el umbral de confianza establecido para la detección es bajo para estas condiciones de iluminación, lo cual causa que el módulo de detección genere una mayor cantidad de detecciones de vehículos que no existen.

Tabla 4.15: Resultados de la evaluación del conteo de detecciones.

Segmento	EM	TE	P	R
G8	0,92	0,08	0,92	1,00
G19	0,52	0,48	0,62	0,76
R8	0,87	0,13	0,91	0,95
R19	0,19	0,81	0,27	0,33

El rendimiento de este módulo se puede mejorar principalmente mediante el uso de un modelo de clasificación más preciso en malas condiciones de iluminación. Para esto, es necesario entrenar el modelo con ejemplos anotados que contemplen grabaciones de tránsito en la noche. Por otro lado, también es necesario ajustar el umbral de confianza, aumentando su valor a medida que disminuye el nivel de luminosidad con el fin de generar la menor cantidad de falsos positivos posibles.

4.4.5. Análisis de líneas de control

Para la evaluación del módulo de análisis de líneas de control se consideraron los videos G8 y G19 y un conjunto de líneas ubicadas de forma tal que permitieron contabilizar el tráfico proveniente desde el norte hacia el este, sur y oeste. La figura 4.4 muestra las líneas y las direcciones de tráfico que se espera contabilizar.

Se consideró la configuración X101-mask-05 para el módulo de detección. Para el módulo de seguimiento se tomó una tolerancia de pérdida de 8 fotogramas para los videos G8 y G19 y 20 fotogramas para los videos R8 y R19.

La tabla 4.16 muestra los resultados de la validación realizada en el módulo de análisis de líneas de control. El escenario diurno obtuvo mejores resultados que el nocturno. Este resultado se corresponde con el obtenido para el módulo de conteo, del cual este módulo es dependiente. Algunos falsos negativos se

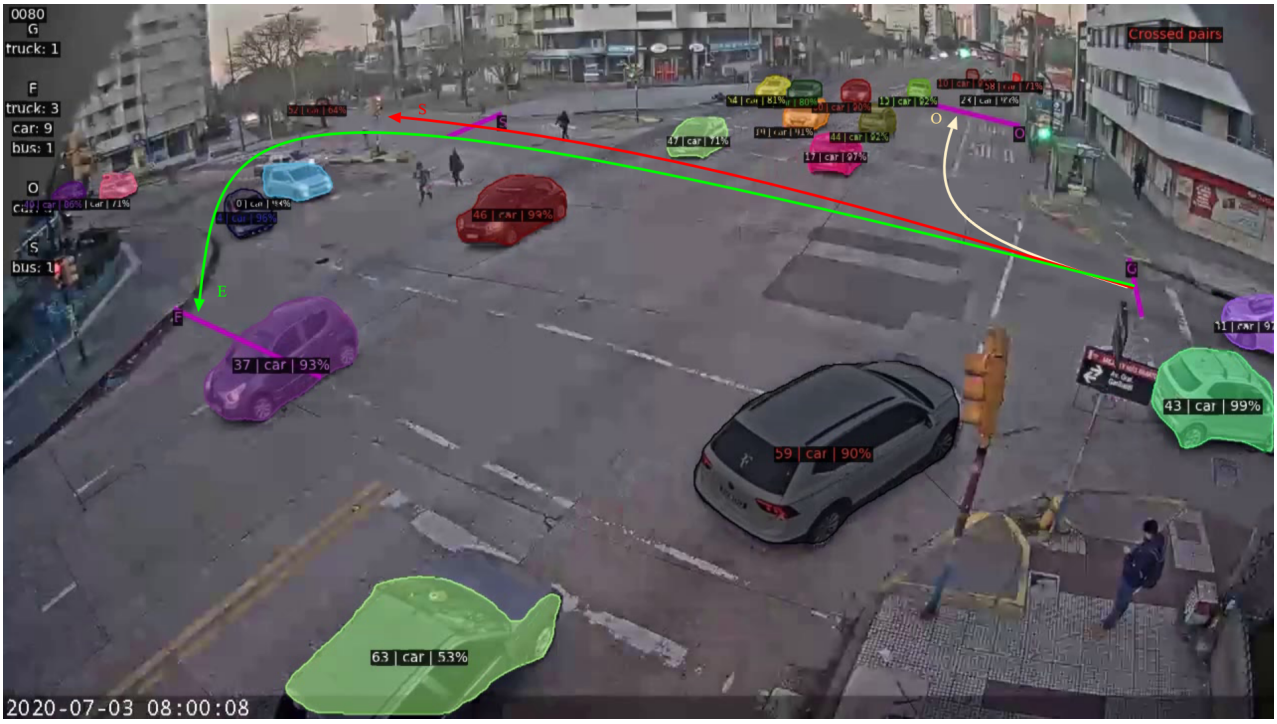


Figura 4.4: Lineas de control establecidas para la validación del módulo de análisis de líneas de control

deben a la oclusión de los vehículos al cruzar una de las líneas. Esto provoca que no sean correctamente contabilizados y el par de líneas no pueda ser validado. En el video de salida se observa que la incorrecta delimitación de una detección de vehículo puede provocar que el área detectada se superponga con una línea de control, generando así casos de conteo incorrecto, los cuales son contabilizados como falsos positivos.

Tabla 4.16: Resultado de la validación del análisis de líneas de control.

Video	*EM*	*TE*	*P*	*R*
G8	0,88	0,01	0,99	0,89
G19	0,63	0,00	1,00	0,63

4.4.6. Análisis de cruces con luz roja

La evaluación del módulo de detección de cruces con luz roja consistió en contabilizar las infracciones observadas en las grabaciones para luego compararlas con los resultados generados a partir de procesar dichas grabaciones

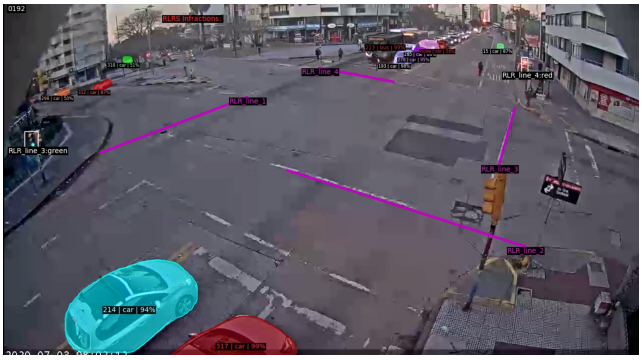
utilizando el sistema. En la validación de este módulo se consideró el conjunto de datos consistente en los vídeos G8, G19, R8 y R19 donde en total se observaron y etiquetaron 10 segmentos de vídeos de aproximadamente 40 segundos conteniendo cada uno una infracción de cruce con luz roja, de estas 10 infracciones, 5 ocurren en condiciones de buena luminosidad y 5 en condiciones de baja luz.

Para la ejecución de la validación se establecieron las luces de semáforos y las lineas de control asociada a cada luz, como muestra la figura 4.5. Se consideró la configuración X101-mask-05 para el módulo de detección de objetos. Para el módulo de seguimiento de objetos se tomó una tolerancia de pérdida de 8 fotogramas para los videos G8 y G19 y 20 fotogramas para los videos R8 y R19.

Los resultados de la validación del análisis de cruce con luz roja, expuestos en la tabla 4.17, indican que de los 10 casos observados se detectaron correctamente 8. A su vez se generaron un total de 39 falsos positivos, casos en los cuales se detectó una infracción de forma incorrecta. El análisis de los resultados indica que estos casos se dan principalmente en los videos con poca luz y suceden principalmente en situaciones en las que el semáforo asociado se encuentra en luz amarilla y es mal clasificada como roja, generando así que los vehículos que crucen en ese instante sean etiquetados como en infracción de forma incorrecta.

Tabla 4.17: Resultados de la validación del análisis de cruce con luz roja.

Video	*Observados*	*VP*	*FP*	*FN*
G8	4	3	15	1
G19	4	3	17	1
R8	1	1	6	0
R19	1	1	1	0



(a) G8

(b) G19

(c) R8

(d) R19

Figura 4.5: Lineas de control y luces asociadas definidas para la validación del análisis de cruce con luz roja.

4.4.7. Análisis de zonas de no detención

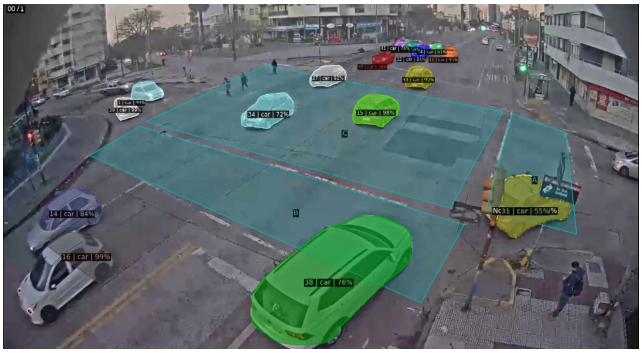
Para la evaluación del módulo de análisis de zonas de no detención se tomaron las versiones de 30 minutos de los videos G8, G19, R8 y R19. El objetivo de la validación fue evaluar la efectividad del módulo en detectar los casos en que un vehículo se detiene en una zona establecida como de no detención. Para cada escenario se definió un conjunto de zonas para contabilizar estos casos. La figura 4.6 muestra las zonas definidas.

El tiempo límite establecido para la contabilización es de 5 segundos. Dada la baja población de casos en los videos de validación este valor permite considerar más casos. Se consideró la configuración X101-mask-05 para el módulo de detección y para el módulo de seguimiento se tomó una tolerancia de pérdida de 8 fotogramas para los videos G8 y G19 y 20 fotogramas para los videos R8 y R19.

La tabla 4.18 presenta el resultado de la validación realizada en el módulo de análisis de zonas de no detención. Se observa que se detectaron correctamente dos de los tres casos existentes. Sin embargo surgieron un total de 11 falsos positivos, donde la mayoría de los casos se dieron por cambios de identidad durante el seguimiento de las detecciones. Los cambios de identidad observados se dieron mayoritariamente entre vehículos que circulaban por la misma calle en direcciones opuestas, aumentando así el tiempo en que el identificador permanecía dentro de la zona de no detención y provocando que el algoritmo considere que el vehículo excedió el tiempo limite de detención.

Tabla 4.18: Resultado de la validación del análisis de zonas de no detención.

Video	*Observados*	*VP*	*FP*	*FN*
G8	0	0	6	0
G19	2	1	4	1
R8	1	1	1	0
R19	0	0	0	0



(a) G8

(b) G19

(c) R8

(d) R19

Figura 4.6: Zonas de contabilización para la validación del análisis de zonas de no detención.

4.4.8. Análisis del brillo y el contraste

Los resultados de detección y seguimiento se vieron afectados en gran medida por las condiciones de luminosidad, ocasionando que la precisión del sistema disminuyera considerablemente al procesar grabaciones nocturnas. Para tener una mejor noción sobre el comportamiento del sistema en estas condiciones, se realizó una evaluación en la cual se analizaron distintos valores de brillo y contraste con el fin de simular distintos niveles de luminosidad.

Para la evaluación de brillo y contraste se consideró un segmento del video R8 al cual se le aplicó un procesamiento para disminuir el brillo desde su valor original hasta llegar a -100 % en intervalos de 10 %. Una vez llegado al menor nivel de brillo posible se realizaron variaciones sobre el nivel de contraste de forma análoga, empezando en 50 % hasta llegar a 100 % en incrementos de 10 %. En la figura 4.7 se puede observar capturas de los videos resultantes luego de variar los parámetros de brillo y contraste del video original.

Considerando los videos resultantes de la variación de brillo y contraste se efectuaron evaluaciones de detección y seguimiento con las configuraciones para las cuales se obtuvo los mejores resultados. Se utilizó la configuración X101-mask-05 para la detección y una tolerancia de pérdida de 8 fotogramas para el seguimiento.

Los resultados de este análisis se presentan en la tabla 4.19. El sistema mantuvo una exactitud media para la detección y MOTA de seguimiento por encima del 50 % hasta el caso de brillo -90 % y contraste 50 %, luego de esto el rendimiento disminuyó significativamente. Esta evaluación presenta un indicador del nivel de rendimiento del sistema esperado en función de las condiciones de luminosidad.

(a) Original (b) Brillo -70 %, Contraste 50 %

(c) Brillo -80 %, Contraste 50 % (d) Brillo -90 %, Contraste 50 %

(e) Brillo -100 %, Contraste 70 % (f) Brillo -100 %, Contraste 100 %

Figura 4.7: Casos de prueba resultantes luego de aplicar variaciones de brillo y contraste

Tabla 4.19: Resultado evaluación de brillo y contraste.

Brillo (%)	*Contraste (%)*	*EM*	*MOTA*
-50	50	0.86	0.83
-60	50	0.85	0.80
-70	50	0.82	0.80
-80	50	0.72	0.74
-90	50	0.57	0.50
-100	50	0.31	0.24
-100	60	0.28	0.18
-100	70	0.26	0.15
-100	80	0.24	0.16
-100	90	0.21	0.12
-100	100	0.20	0.10

Capítulo 5

Conclusiones y trabajo futuro

Este trabajo presentó el diseño e implementación de un sistema para el análisis de datos de tráfico mediante técnicas de inteligencia computacional.

El sistema propuesto se desarrolló siguiendo una arquitectura de tipo pipeline flexible construida sobre la moderna biblioteca de detección de objetos Detectron2. El diseño propuesto proporciona un procesamiento de fotogramas eficiente, mediante el uso de módulos funcionales configurables e independientes, acoplados ligeramente entre ellos. Esta característica permite incluir nuevos métodos, modificar los existentes y evaluar diferentes alternativas y configuraciones.

Los problemas de detección y seguimiento de objetos se resuelven utilizando las bibliotecas Detectron2 y Node Moving Things Tracker, respectivamente. La información generada por estos módulos a partir de los videos de tráfico permitió implementar un conjunto de módulos para la recolección y análisis de datos de tráfico.

La validación del sistema propuesto se realizó mediante videos reales de dos escenarios que incluyen importantes calles de Montevideo, Uruguay, bajo diferentes condiciones (luz diurna, nocturna y diferentes calidades de video). Estas grabaciones fueron tomadas en horas pico y muestran un interesante flujo de vehículos.

Los resultados demuestran la efectividad del sistema en escenarios con condiciones adecuadas de iluminación. En estas condiciones, la evaluación de la detección de objetos mostró una exactitud media global del 70 %, la evaluación de la clasificación de objetos resultó en un 85 % para la misma métrica y en la evaluación de seguimiento la puntuación media de MOTA fue del 85 %. En

el caso del la detección y el seguimiento, presentaron un mejor rendimiento con los modelos de máscara. Por otro, lado en las grabaciones que presentaban condiciones de poca iluminación los resultados disminuyeron considerablemente hasta 30 % tanto para la clasificación así como para el seguimiento.

Los módulos de análisis de datos obtuvieron resultados en la misma línea que el seguimiento, algo esperable dada la dependencia directa que existe entre ellos. El módulo de conteo obtuvo una exactitud media del 89 % en las grabaciones durante el día y del 36 % para las grabaciones de noche. En el caso del análisis de líneas de control se obtuvo 88 % y 63 % para el día y la noche respectivamente. El análisis de los cruces con luz roja se detectaron correctamente 8 de 10 casos observados, aunque se obtuvieron 39 falsos positivos. Finalmente, la evaluación del módulo de análisis de zonas de no detención mostró que se pudo detectar correctamente dos de los tres casos observados pero obteniendo a su vez 11 falsos positivos.

Una de las principales contribuciones de este proyecto de grado es un sistema flexible con la posibilidad de adaptarse a las necesidades de otros sistemas de transporte existentes, aportando capacidades de recolección de datos y la posibilidad de automatizar o asistir en tareas de monitorización del tráfico. Por otro lado, este trabajo muestra las dificultades que se presentan a la hora de implementar este tipo de sistemas y realizar validaciones sobre escenarios reales.

Las principales líneas de trabajo futuro están relacionadas con la mejora del módulo de detección de objetos entrenando los modelos con conjuntos de datos que incluyan imágenes de trafico que presenten malas condiciones de iluminación o malas condiciones meteorológicas. Esto permitirá obtener mejoras en la precisión del módulo de seguimiento, lo cual derivará en mejores datos de entrada para los módulos de análisis.

Adicionalmente, se puede experimentar con otras bibliotecas, tanto de detección como de seguimiento. El uso de otro tipo de redes como YOLO o de otras APIs como Tensorflow puede llevar a otros resultados en la precisión y en los tiempos de ejecución de la detección. Por otro lado, el uso de una biblioteca de seguimiento implementada en Python podría ayudar a reducir los tiempos de ejecución, ya que se evitaría el uso de HTTP y se realizaría dentro del mismo proceso.

Otra línea de trabajo interesante está relacionada con el desarrollo de métodos de detección de patrones más sofisticados para capturar eventos relevantes como cambios bruscos de carril o incluso accidentes de tráfico.

Referencias bibliográficas

Abadi, M. (2016). Tensorflow: Learning functions at scale. *SIGPLAN Not.*, 51(9):1.

Arinaldi, A., Pradana, J., & Gurusinga, A. (2018). Detection and classification of vehicles for traffic video analytics. *Procedia Computer Science*, 144:259–268.

Benevolo, C., Dameri, R., & D'Auria, B. (2016). Smart mobility in smart city. En Torre, T., Braccini, A., & Spinelli, R., editores, *Empowering Organizations*, pp. 13–28, Cham. Springer International Publishing.

Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. *IEEE International Conference on Image Processing*, p. 3464.

Bewley, A., Wojke, N., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. En *International Conference on Image Processing, ICIP*, pp. 3645–3649, University of Koblenz-Landau.

Bochinski, E., Eiselein, V., & Sikora, T. (2017). High-speed tracking-by-detection without using image information. En *14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6. IEEE.

Bochinski, E., Senst, T., & Sikora, T. (2018). Extending IOU based multi-object tracking by visual information. En *IEEE International Conference on Advanced Video and Signals-based Surveillance*, pp. 441–446.

Cardozo, O. & Rey, C. (2007). La vulnerabilidad en la movilidad urbana: aportes teóricos y metodológicos. volumen Aportes conceptuales y empíricos de la vulnerabilidad global, pp. 399–425.

Challita, S., Zalila, F., Gourdin, C., & Merle, P. (2018). A precise model for Google Cloud Platform. En *6th IEEE International Conference on Cloud Engineering*, pp. 177–183.

Chauhan, M., Singh, A., Khemka, M., Prateek, A., & Sen, R. (2019). Embedded cnn based vehicle classification and counting in non-laned road traffic. En *Tenth International Conference on Information and Communication Technologies and Development*, pp. 1–11.

Chavat, J. P. & Nesmachnow, S. (2018). Computational intelligence for detecting pedestrian movement patterns. pp. 148–163.

Dagaeva, M., Garaeva, A., Anikin, I., Makhmutova, A., & Minnikhanov, R. (2019). Big spatio-temporal data mining for emergency management information systems. *IET Intelligent Transport System*, 13(11):1649–1657.

Del Sole, A. (2018). Introducing microsoft cognitive services. En *Microsoft Computer Vision APIs Distilled*. Apress.

Dey, S., Kalliatakis, G., Saha, S., Singh, A., Ehsan, S., & McDonald, K. (2018). MAT-CNN-SOPC: Motionless analysis of traffic using convolutional neural networks on system-on-a-programmable-chip. En *2018 NASA/ESA Conference on Adaptive Hardware and Systems*, pp. 291–298.

Figueiredo, L., Jesus, I., Machado, J., Ferreira, J. R., & Martins de Carvalho, J. L. (2001). Towards the development of intelligent transportation systems. En *IEEE Intelligent Transportation Systems.*, pp. 1206–1211.

Fu, H., Wu, L., Jian, M., Yang, Y., & Wang, X. (2019). Mf-sort: Simple online and realtime tracking with motion features. En *International Conference on Image and Graphics*, pp. 157–168. Springer.

Gilewski, J. (2019). detectron2-pipeline: Modular image processing pipeline using OpenCV and Python generators powered by Detectron2. https://github.com/jagin/detectron2-pipeline. [Online; último acceso 2020-03-15].

Girshick, R. (2015). Fast R-CNN. En *IEEE International Conference on Computer Vision*, pp. 1440–1448.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377.

Güler, R., Neverova, N., & Kokkinos, I. (2018). Densepose: Dense human pose estimation in the wild. En *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7297–7306.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. En *IEEE International Conference on Computer Vision*, pp. 2961–2969.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. En *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Howse, J. (2013). *OpenCV computer vision with python*. Packt Publishing Ltd.

ILSVRC (2017). IMAGENET large scale visual recognition challenge. http://image-net.org/challenges/LSVRC/. [Online; último acceso 2020-02-29].

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., & Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. En *22nd ACM International Conference on Multimedia*, pp. 675–678.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, pp. 35–45.

Kirillov, A., Girshick, R., He, K., & Dollár, P. (2019). Panoptic feature pyramid networks. En *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6399–6408.

L Leal-Taixé, M Fenzi, A. K. B. R. S. S. (2014). Learning an image-based motion context for multiple people tracking. *IEEE Conference on Computer Vision and Pattern Recognition*.

Leal-Taixé, L., Milan, A., Reid, I., Roth, S., & Schindler, K. (2015). MOTChallenge 2015: Towards a benchmark for multi-target tracking. *arXiv:1504.01942 [cs]*.

Lou, L., Zhang, J., Xiong, Y., & Jin, Y. (2019). A novel vehicle detection method based on the fusion of radio received signal strength and geomagnetism. *Sensors*, 19(1):58.

Makhmutova, A., Anikin, I., & Dagaeva, M. (2020). Object tracking method for videomonitoring in intelligent transport systems. En *International Russian Automation Conference*, pp. 535–540.

Massa, F. & Girshick, R. (2018). maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. https://github.com/facebookresearch/maskrcnn-benchmark. [Online; último acceso 2020-03-11].

Massobrio, R. & Nesmachnow, S. (2020). Urban mobility data analysis for public transportation systems: A case study in montevideo, uruguay. *Applied Sciences*, 10:5400.

Matlab (2020). Matlab. https://in.mathworks.com/products/matlab.html. [Online; último acceso 2020-03-07].

MOTChallenge (2019). MOT challenge: Multiple object tracking benchmark. https://motchallenge.net/. [Online; último acceso 2020-03-11].

Move-lab (2018). Tracking things in object dectection videos. https://www.move-lab.com/blog/tracking-things-in-object-detection-videos. [Online; último acceso 2020-03-15].

Nesmachnow, S. & Iturriaga, S. (2019). Cluster-UY: Collaborative Scientific High Performance Computing in Uruguay. En *Communications in Computer and Information Science*, pp. 188–202. Springer International Publishing.

Peters, J. F. (2017). *Foundations of Computer Vision: computational geometry, visual image structures and object shape detection*, volumen 124. Springer.

Python (2020). Python generators. https://wiki.python.org/moin/Generators. [Online; último acceso 2020-03-16].

Rekognition (2020). Amazon rekognition. https://aws.amazon.com/rekognition/. "[Online; último acceso 2020-03-07]".

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. En Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., & Garnett, R., editores, *Advances in Neural Information Processing Systems 28*, pp. 91–99. Curran Associates, Inc.

Shiffman, D. (2011). OpenCV matching faces over time. https://shiffman.net/general/2011/04/26/opencv-matching-faces-over-time/. [Online; último acceso 2020-03-15].

SimpleCV (2020). SimpleCV: Computer vision platform using python. http://simplecv.org/. [Online; último acceso 2020-03-07].

Sokolova, M. & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437.

Stanford (2019). Cs231n convolutional neural networks for visual recognition. http://cs231n.github.io/classification/. [Online; último acceso 2020-02-29].

Uy, A., Quiros, A., Bedruz, R., Abad, A., Bandala, A., Sybingco, E., & E.Dadios (2016). Automated traffic violation apprehension system using genetic algorithm and artificial neural network. En *IEEE Region 10 Conference*, pp. 2094–2099.

Winter, H., Serra, J., & Nesmachnow, S. (2020). Computational intelligence for analysis of traffic data. En Nesmachnow, S. & Callejo, L. H., editores, *Smart Cities*, volumen 1359 de *Communications in Computer and Information Sciences*. Springer.

Wu, Y., Kirillov, A., Massa, F., Lo, W., & Girshick, R. (2019). Detectron2. https://github.com/facebookresearch/detectron2. "[Online; último acceso 2020-03-07]".

Yang, H. & Qu, S. (2018). Real-time vehicle detection and counting in complex traffic scenes using background subtraction model with low-rank decomposition. *IET Intelligent Transport Systems*, 12:75–85.

Yang, Z. & Pun-Cheng, L. (2018). Vehicle detection in intelligent transportation systems and its applications under varying environments: A review. *Image and Vision Computing*, 69:143 – 154.

Zhang, K., Zhang, L., & Yang, M.-H. (2014). Fast compressive tracking. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 36(10):2002–2015.

Zhang, S., Wu, G., Costeira, J., & Moura, J. (2017). FCN-rLSTM: Deep spatio-temporal neural networks for vehicle counting in city cameras. En *IEEE International Conference on Computer Vision*, pp. 3667–3676.

Zheng, M., Li, T., Zhu, R., Chen, J., Ma, Z., Tang, M., Cui, Z., & Wang, Z. (2019). Traffic accident's severity prediction: A deep-learning approach-based cnn network. *IEEE Access*, 7:39897–39910.

Zhou, Y., Nejati, H., Do, T., Cheung, N., & Cheah, L. (2016). Image-based vehicle analysis using deep neural network: A systematic study. En *IEEE International Conference on Digital Signal Processing*, pp. 276–280.

APÉNDICES

Apéndice 1

Parámetros del sistema

La tabla 1.1 muestra la salida de la ejecución del comando de ayuda del pipeline que imprime las opciones existentes.

Tabla 1.1: Parámetros del pipeline para el análisis de videos de tráfico.

Atajo	*Parámetro*	*Descripción*
-h	–help	show this help message and exit
-cf	–config-file	argument list file with the same accepted format as in command line; file values are overriden by command line values
-i	–input	path to input video file, image frames directory or camera identifier (default: camera 0)
-o	–output	path to output directory
-ov	–out-video	output video file name
	–fps	overwrite fps for output video or if it is unknown for image frames directory
-ic	–include-classes	list of label classes to annotate; format: ['Car', 'Bus', 'Motorcycle']

-p	–progress	display progress
-d	–display	display video
-sc	–show-count	whether or not annotate the video with the count results
-t	–tracker	tracking algorithm. No tracking if empty. Values: nmtt
-nmttu	–nmtt-url	server URL for the NMTT
-nmttuft	–nmtt-unmatched-frames-tolerance	number of frames we wait when an object isn't matched before considering it gone. Default: 5
-nmttil	–nmtt-iou-limit	exclude things from beeing matched if their IOU is lower than this, 1 means total overlap whereas 0 means no overlap. Default: 0.05
-cl	–counting-lines	counting lines; format: ['label':'A', 'line': [(667, 713), (888, 713)], 'light': [(360, 119), (398,154)]]
-clum	–cl-use-masks	Use the segmentation mask of the detections for line cross calculation
-il	–invalid-lines	invalid lines; format: [('A', 'B'), ('A', 'C')]
-slc	–show-lines-count	whether or not annotate the video with the count results of each counting line
-droi	–detection-roi	the region of interest where detection will be performed; format: [(750, 405), (1094, 398), (1569, 1028), (501, 1028)]
-sdroi	–show-droi	whether or not annotate the video with the selected DROI

-nsr	–no-stop-regions	regions where vehicles are not supposed to stop; format: ['label': 'A', 'region': [(636, 89), (446, 172), (493, 430), (644, 425), (860,161)]]
-nsrf	–nsr-frames	number of frames elapsed to count a vehicle as stopped in a NSR; default: 10
-nsrp	–nsr-percentage	intersection percentage of the vehicle to count as inside the NSR; default: 10
-nsrum	–nsr-use-masks	use the segmentation mask of the detections for region intersection
-snsr	–show-nsr	whether or not annotate the video with the selected NSRs
	–d2-config-file	path to config file (default: configs/COCO- Instance-Segmentation/mask_rcnn_-R_50_FPN_3x.yaml)
	–d2-config-opts	modify model config options using the command-line
	–d2-weights-file	path to model weights file
	–d2-confidence-threshold	minimum score for instance predictions to be shown (default: 0.5)
	–gpus	number of GPUs (default: 1)
	–cpus	number of CPUs (default: 0)
	–queue-size	queue size per process (default: 3)
	–single-process	force the pipeline to run in a single process
	–log-type	values: console | file

	–log-level	values: NOSET | DEBUG | INFO | WARNING | ERROR | CRITICAL. NOSET logs all messages
	–log-files-directory	directory path where the log file will be stored
	–log-main-file-name	name of the main log file
-su	–speed-units	Calculates and show the detection speed in pixels per frame or pixels per second; values: ppf | pps
-suf	–speed-update-frequency	Frequency of speed update (default: 10)
-lcqs	–light-color-queue-size	Queue size for color decision in red light runs analysis (default: 9)

ANEXOS

Anexo 1

Artículo presentado en ICSC 2020

Este anexo presenta el artículo científico publicado y presentado en el III Congreso Iberoamericano de Ciudades Inteligentes (Winter *et al.*, 2020).

Computational intelligence for analysis of traffic data

Hernán Winter[1][0000−0002−4075−3694], Juan Serra[1][0000−0002−4286−9316], Sergio Nesmachnow[1][0000−0002−8146−4012], Andrei Tchernykh[23][0000−0001−5029−5212], and Vladimir Shepelev[3]

[1] Universidad de la República, Montevideo, Uruguay
{hernan.winter,juan.serra,sergion}@fing.edu.uy
[2] CICESE, México
chernykh@cicese.mx
[3] South Ural State University
shepelevvd@susu.ru

Abstract. This article presents a system developed for the collection and analysis of traffic data obtained from traffic camera videos using computational intelligence. The proposed system is developed using the modern object detection library Detectron2. A pipeline-type architecture is used for frame processing, where each step is an independent, configurable functional module, loosely coupled to the others. The validation of the proposed system is performed on real scenarios in Montevideo, Uruguay, under different conditions (daylight, nightlight, and different video qualities). Results demonstrate the effectiveness of the system in the considered scenarios.

Keywords: computational intelligence; neural networks; traffic data; smart cities

1 Introduction

The growth of cities and traffic density have led to an increased demand for surveillance systems capable of automating traffic monitoring and analysis. The main goal of these automatic systems is to aid or even remove the human labor for vision based tasks that can be performed by a computer, providing regulators and authorities the ability to respond quickly to diverse traffic issues and situations.

Tasks such as vehicle counting and infraction detection are of great importance for Intelligent Transportation Systems [23]. Recently, computer vision based detection and counting algorithms [22] have shown to be more effective and outperform traditional traffic surveillance methods, such as methods using different kinds of sensors [12]. However, there are still many challenges and open issues in computer vision based vehicle detection and counting processes, caused by illumination variation, shadows, occlusion, and other phenomena.

In this line of work, this article presents a system applying computational intelligence (based on Artificial Neural Networks, ANN) to solve traffic analysis

problems using video recordings provided by surveillance cameras. The problems solved include vehicle detection, counting, classification, tracking, and detection of different types of traffic offenses, such as red light intersection crossing and parking vehicles in not allowed zones. The validation of the proposed system is developed using real traffic videos from the city of Montevideo, Uruguay.

The main contributions of the reserch reported in this article include: i) a methodology for the design of traffic analysis software systems using videos; ii) specific implementations of vehicle detection, counting, classification and tracking methods, and iii) the validation of the proposed methods on real scenarios.

The article is structured as follows. Section 2 presents a background on computational intelligence for image analysis. A review of the main related work is presented in Section 3. The technical aspects of the solution, including the proposed architecture, modules, and supporting libraries are described in Section 4. The experimental validation is reported and results are discussed in Section 5. Finally, Section 6 presents the conclusions and the main lines of future work.

2 Computational intelligence for image analysis

This section describes the main concepts about the analysis of traffic data using computational intelligence.

2.1 Detection

One of the fundamental problems in computer vision is the task of assigning a label from a fixed set of categories to an input image. This task is known as image classification and is divided into tree subtasks: segmentation, location, and detection.

The goal of semantic segmentation is to obtain a category for each pixel given an input image. It does not differentiate instances of the same object because each pixel in the image is classified independently. The classification and location task consists of classifying an image with a label that describes an object and drawing the box within the image around the object. The output in this task are a label that identifies an object and a box that indicates where that object is located. Object detection takes as input a set of categories of interest and an image. The goal of this task is to draw a box around each one of these categories, each time they appear in the image, and also predict the category. This problem is different from classification and localization since there can be a variable number of outputs for each input image.

Another task to consider is instance segmentation. Given an input image, this task seeks to predict the locations and identities of the objects in that image. Additionally, instead of simply predicting a region for each of those objects, this task seeks to predict a segmentation mask for each of those objects and to predict which pixels in the image correspond to each object instance.

In the last few years, computational intelligence and deep learning have led to successful results on a variety of problems, including image classification.

Among different types of deep ANNs, Convolutional neural networks (CNN) have been extensively studied [8]. CNNs assume that the input to be classified is an image. This assumption allows the network to be more efficient and to design architectures that greatly reduce the number of network parameters.

Solving the object detection problem involves determining all the regions where objects to be classified can be located. Given an input image, a Region Proposal Network (RPN) uses signal processing techniques to create a list of proposed regions in which an object can exist. This architecture class is named R-CNN. Given an input image, an RPN is executed to obtain the proposals, also called Regions of Interest (RoI). The main drawback of this approach is its very high computational demands. In practice, the network training is slow and needs significant memory. Fast R-CNN was proposed to mitigate these problems, working in a similar way to R-CNN. In terms of speed, Fast R-CNN has proven to be nine times faster than CNN in training time [7]. However, the computation time is dominated by the calculation of the RoI, which turns out to be a bottleneck. This last problem is solved in Faster R-CNN [17].

Finally, one of the most recent methods to solve the instance segmentation task is the Mask R-CNN architecture. Similarly to Faster R-CNN, this method follows a multi-stage processing approach. It receives the complete image, which is executed through a convolutional network and a learned RPN. Once the RoIs are learned, they are projected onto the convolutional vector. Then, instead of simply performing the classification and the regression of the regions of each RoI, the method additionally predicts a segmentation mask for each region, solving a semantic segmentation problem within each of the regions proposed by the RPN. The RoI is finally wrapped to the proper shape.

2.2 Tracking

Object tracking consists in the process of accurately estimating the state of an object -position,identity,configuration- over time from observations [14], thus generating a trajectory given by the position of the object in each frame. When several objects are located at the same time, the problem is called Multiple Object Tracking. In this scenario, the difficulty of the task increases considerably due to the occlusion generated by the interaction of the objects, which in turn may have similar appearances. On the other hand, conditions such as the speed at which the objects move, the lighting or that these change their appearance depending on the position, require that the tracking system must be robust, maintaining the object identifier in such situations.

In classical tracking methods, object features are extracted in each frame and used to search for the same object in subsequent frames [25]. This causes errors to accumulate in the process and if occlusion or frame skipping occurs, tracking fails because of the rapid change of appearance features in local windows. Thus, modern tracking methods apply two steps: object detection and data association. First, objects are detected in each frame of the sequence and then, detected objects are matched across frames. This paradigm is called tracking by detection [10] and it relies on the performance of the detection algorithm. Detected

objects are matched across frames using different approaches, including optical flow with mean shift of color signature, Earth mover's distance to compare color distributions, fragment-based features, and computational intelligence.

Another simple but effective method based on the tracking by detection approach is Intersection Over Union (IOU) [2]. This method requires a detection algorithm with a high rate of true positive results, as a detection is expected in each frame for each object to be tracked. It is also assumed that the detection of the same object in two consecutive frames present a great overlap of the intersection over the union (defined in Eq. 1), which is common for videos that present a high refresh rate.

$$IOU(a,b) = \frac{Area(a) \cap Area(b)}{Area(a) \cup Area(b)} \tag{1}$$

The advantage of the IoU method, in addition to its simplicity, is that it has lower computational cost than other methods. IoU can be integrated on other methods to achieve a more robust and accurate monitoring

3 Related work

Several articles have proposed automated systems for the analysis of traffic data applying image processing and computational intelligence techniques. The most related to the research reported in this article are reviewed next.

Zhou et al. [28] studied the vehicle detection and classification problem applying deep neural networks. The You Only Look Once (YOLO) architecture was used for vehicles detection and post-processing was performed to eliminate invalid results. The Alexnet architecture was applied for classification, feature extraction, and fine-tuning. The YOLO network obtained similar precision than a Deformable Parts Model, while the Alexnet network using Support Vector Machines (SVM) outperformed other methods such as Principal Component Analysis and Absolute Difference in a public dataset.

Uy et al. [20] studied methods for identifying traffic offenses using genetic algorithms (GA) and the recognition of offenders through ANN. GA were applied to detect vehicles obstructing pedestrian crossings and to identify the location of license plates in images, while a ANN is used to recognize the license plate number. The license plates recognition accuracy was high (91.6% on 47 test images), but some license plates were not properly located due to the vehicle position respect to the camera. Zhang et al. [26] applied a Fully CNN with Long Short Term Memory for the the vehicles counting problem. Compared to the state of the art, the proposed ANN architecture reduced the mean absolute error (MAE) from 2.74 to 1.53 on the WebCamT annotated dataset and from 5.31 to 4.21 on the TRANCOS dataset. In addition, the training time was accelerated by up to 5 times. However, the proposed ANN was not capable of handling long periods of information due to the large amount of memory required.

Dey et al. [5] applied CNN in a System-On-a-Programmable-Chip to analyze and categorize traffic, including the quality-of-experience variable to improve

predictions. A combination of transfer learning with re-training CNN models, allowed improving the prediction accuracy. Arinaldi et al. [1] applied computer vision techniques to automatically collect traffic statistics using Mixture of Gaussian (MoG) and Faster Recurrent CNN. Training and validation were developed on Indonesian road videos and a public dataset from MIT. Faster Recurrent CNN was best suited for detecting and classifying moving vehicles in a dynamic traffic scene, since MoG was weak for separating overlapping vehicles.

Chauhan et al. [3] studied CNN or real-time traffic analysis on Delhi, India. A YOLO network was used, pre-trained on the MS-COCO dataset and fitted with annotated datasets. The best trained model achieved a performance of 65–75% mean average precision, depending on the camera position and the vehicle class. This article provides the expected performance of YOLO models optimized using annotated data. The recent article by Zheng et al. [27] proposed TASP-CNN for predictinig the severity of traffic accidents, considering relationships between accident features. The proposed method was successfully adapted to the representation of traffic accident severity features and deeper correlations of accident data. The performance of TASP-CNN was better than previous models when evaluated using data from an eight years period.

Our research group has developed research on detection on pedestrian movement patterns applying computational intelligence [4]. A flexible system was developed to process multiple image and video sources in real time applying a pipes and filters architecture to address different subproblems. The proposed system has two main stages: extracting relevant features of the input images, by applying image processing and object tracking, and patterns detection. The experimental analysis of the system was performed over more than 1450 problem instances, using PETS09-S2L1 videos and the results were compared with part of the MOTChallenge benchmark results. Results indicate that the proposed system is competitive, yet simpler, than other similar software methods.

4 The proposed system for traffic data analysis

This section presents the implemented system for traffic data analysis, describing the function of each module and the input and output parameters.

4.1 Overall description

The proposed approach is based on a modular architecture that implements an image processing pipeline [6]. The pipeline executes a set of tasks over input images (e.g., translation/rotation, resizing, etc.) to extract useful features. The modular architecture allowed for a progressive development process, starting from a few general modules and incorporating specific modules for relevant data.

Fig. 1 shows the final architecture of the pipeline for traffic video analysis, consisting of twelve modules. The pipeline has 40 parameters that allow controlling different aspects of the processing in each module.

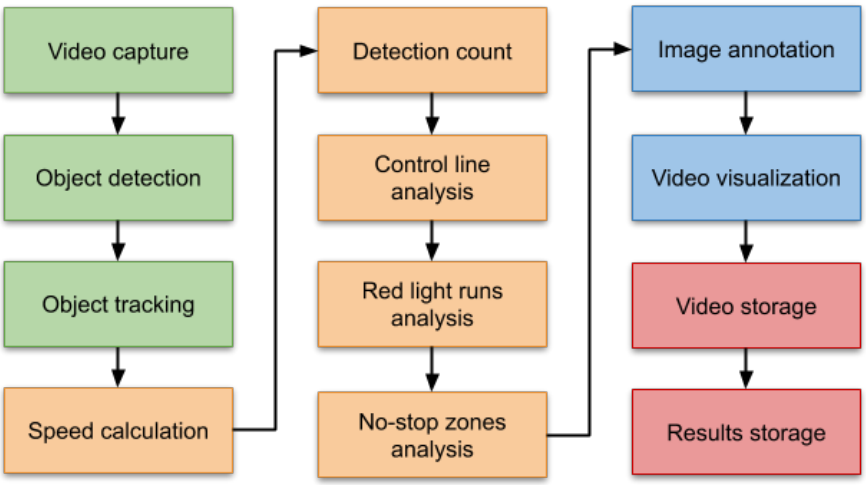


Fig. 1: The proposed pipeline for traffic video análisis

Four main stages are identified: i) video capture, object detection and tracking (in green in Fig. 1), detection data analysis (in orange), results visualization (in blue), results storage (in red). They are described in the following subsections.

4.2 Video capture and object detection

The goal of video capture is producing the frames used in the pipeline. A video stream (e.g., a local file or a webcam) is captured by a fast method using multithreading parallel computing to read the video frames, using OpenCV utilities. Video capture initialize the cumulative data transfer object (DTO), used by all modules to read and write data, and stores several fields in the DTO, including *frame number*, *image* object, and *annotations*.

Then, the object detection process each frame to produce a bounding box, mask, score and class of the detected objects. This module is based on Detectron2 framework by Facebook [21], whose modular design allows using different state-of-the-art detection algorithms, such as Faster R-CNN, Mask R-CNN, or RetinaNet. The implemented module uses both synchronous and asynchronous detection, and adds different functionalities on top of the detection framework such as the possibility of defining regions of interest for the detection or filtering the classes of the detected objects. The output of the detection module is converted to the standard format used by the rest of the pipeline, so it might be replaced by other detection module without affecting the other modules in the pipeline. The output can contain bounding boxes or instance segmentation. The rest of the pipeline is compatible with both type of outputs and can take advantage of instance segmentation when available to compute more precise results when analyzing patterns.

4.3 Detection data analysis

Detection analysis includes five modules to extract information from data generated by previous modules in the pipeline.

The *speed calculation* module computes an estimation of the average speed of each detected object in recent frames, in pixels per frame (PPF) or pixels per second (PPS). The system stores the position of the center of each detected object in the last n frames (n is a parameter) and so the speed is given by the Euclidean norm of the first and last stored positions.

Detection count requires defining one or more counting lines on the video image. Based on a structure that keeps each detected vehicle as an object with several identifying properties, the counting module analyzes the vehicles that overlap with the counting lines. This analysis considers the intersection of the polygon of the mask or box with respect to the defined lines as an input parameter. If an overlapping is found, the vehicle information is updated with the lines it overlapped and the frame number in which it did so.

With *control lines analysis* it is possible to define a relationship between two lines, meaning that a vehicle should not cross both as doing so would be consider an infraction. This allows detecting different types of infractions that involves a vehicle circulating in a no-driving zone, e.g., a wrong turn or lane change near a corner. This module uses detected bounding boxes or masks to recognize if a vehicle overlaps with both lines in the relationship through the video.

The *red light runs analysis* module detect driving offenses of failing to comply with red light signal. The region where each semaphore is located is defined as an input parameter and each traffic light is associated with a line. The defined traffic lights are analyzed to determine their color frame by frame, applying an algorithm that transforms the cropped frame of the traffic light to Hue, Saturation, Value (HSV) color model.

The *no-stop zones analysis* considers zones, represented by polygons, in which vehicles should not stop (or park). The module analyzes the bounding box or mask of those vehicles that intersects with the defined non-stop zone. If an intersection greater than a certain value is detected, then the average speed of the vehicle in the last n frames (n is a parameter) is considered. When the average speed reaches a value lower than one, the vehicle is considered in infraction and labeled as stopped.

4.4 Annotation and results visualization

The *annotation* module is responsible of modifying frames to show information generated by the previous modules. The modified frames will be part of the output video. Additionally, this module is in charge of drawing the detected objects, boxes, or masks as appropriate. Fig. 2 presents an example of an annotated frame in one of the scenarios studied in this article.

The *video visualization* module is in charge of displaying the output frames as they are produced, using OpenCV to create a window and display the frames.

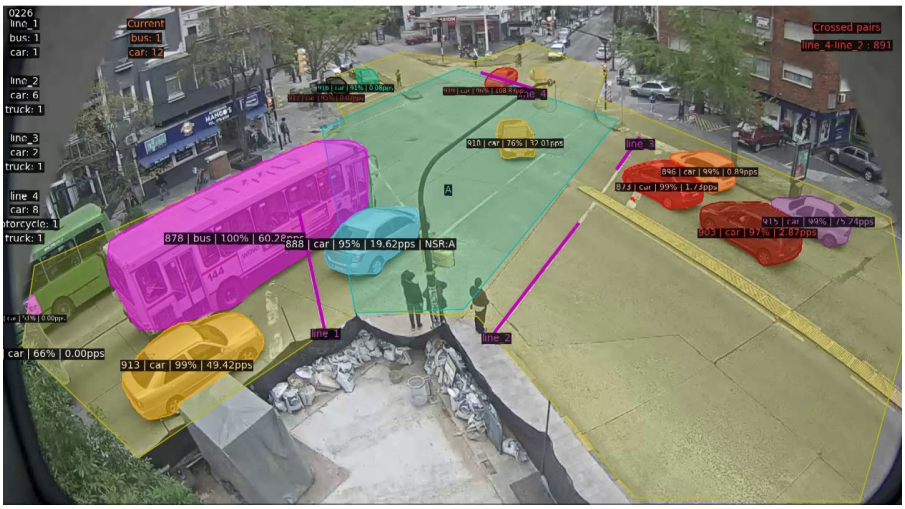


Fig. 2: Frame annotated with object masks, labels, text, lines, and polygons

4.5 Storage

The *video storage* module saves the frames in a file in a given path, considering the output format and FPS rate specified an parameters. Finally, the *results storage* module shows the information generated in each frame and the cumulative one, using a JSON-based logging system.

5 Validation experiments

This section reports the validation experiments of the proposed system.

5.1 Case studies in Montevideo, Uruguay

The experimental evaluation considered two case studies in Montevideo, Uruguay. The first case study correspond to the intersection of 8 de Octubre and Garibaldi avenues, representing a classic intersection between two avenues in Montevideo. The second case study correspond to the intersection between Rambla Wilson and Sarmiento Avenue. This case is relevant because it involves the avenue considered as the main traffic lane during rush hour.

The test dataset used consists of four videos taken by video surveillance cameras from the two studied locations. The cameras model is AXIS P1365 Mk II and record up to 60 frames per second. Recordings were taken at two different times of the day, in the morning (8:00 AM) and in the evening (7:00 PM) to analyze the efficacy of the proposed system under different lighting conditions. Fig. 3 presents sample images of the considered scenarios. In turn, Table 1 summarizes the main properties of the analyzed videos.

(a) G8 (b) G19

(c) R8 (d) R19

Fig. 3: Testing video scenarios

Table 1: Properties of the test videos

reference	*format*	*resolution*	*# frames*	*rate*	*duration*
G8	MP4	1280×720 pixels	2393	8 FPS	5 minutes
G19	MP4	1280×720 pixels	2398	8 FPS	5 minutes
R8	MP4	1280×720 pixels	5998	20 FPS	5 minutes
R19	MP4	1280×720 pixels	5997	20 FPS	5 minutes

5.2 Development and execution platform

The proposed system was developed using Python and Anaconda for project environment management allowing to install and maintain the required libraries easily. For the implementation, training, and execution of the presented ANN models, the Detectron2 framework [21], based on pytorch, was used. The tracking service was provided by a Node.js server [19] running an implementation of the Node Moving Things Tracker [15] library. OpenCV [13] was used for image and video manipulation and processing.

The experimental evaluation was performed on a virtual environment defined on a high-end server with Xeon Gold 6138 processors (40 cores and 80 threads per core), 8 GB RAM, a NVIDIA P100 GPU and a 300 GB SSD, from National Supercomputing Center (ClusterUY) [16]. Using this high performance computing platform, it was possible to dynamically reserve the resources needed for the batch jobs for the system execution and validation.

5.3 Metrics for evaluation

Statistical measures were considered for the evaluation of the developed system. To account for the performance of stages that involve a binary classification, the standard metrics were applied: True Positive (TP), which indicates the number of occurrences where the model correctly predicts the positive class; True Negative (TN) is the number of occurrences where the model correctly predicts the negative class; False Positive (FP) the number of occurrences where the model incorrectly predicts the positive class; and False Negative (FN) indicates the number of occurrences where the model incorrectly predicts the negative class.

Metrics proposed by Sokolova and Lapalme [18] were applied to evaluate the performance of detection and classification algorithms, including:

- *Avarage Accuracy*: indicates the overall effectiveness of a classifier (Eq. 2).
- *Error Rate*: indicates the average per-class classification error (Eq. 3).
- *Precision*: reflects the percentage of the results which are relevant (Eq. 4).
- *Recall*: refers to the percentage of total relevant results correctly classified (Eq. 5).

$$\frac{\frac{TP+TN}{TP+TN+FP+FN}}{n} \quad (2) \qquad \frac{FP+FN}{TP+TN+FP+FN} \quad (3)$$

$$\frac{TP}{TP+FP} \quad (4) \qquad \frac{TP}{TP+FN} \quad (5)$$

Metrics proposed by MOTChallenge [11] were used to evaluate the tracking method. Special metrics are required for evaluating multiple object tracking:

- Identity Switches ($IDSW$): indicates the number of occurrences where an already identified object is assigned a new identificator.
- Multiple Object Tracking Accuracy ($MOTA$): is a global performance indicator of the tracker combining three sources of error. (Eq. 6, where t represents the frame and G_t the number of objects in frame t).

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + ID_{Sw})}{\sum_t G_t} \quad (6)$$

5.4 Results: object detection

For the evaluation of the object detection module, the configuration baseline of Detectron2 and the detection confidence threshold were taken into consideration.

The Detectron2 configuration baseline is a set of parameters which determines the type of ANN to be executed and the weight model. These baselines are part of the Model Zoo in Detectron2 [21]. The main properties of the selected baselines are presented in Table 2.

Table 2: Details of the configuration baselines of Detectron2 used in the experimental evaluation of object detection

	R101-box	*X101-box*	*R101-mask*	*X101-mask*
Backbone	R101-FPN	X101-FPN	R101-FPN	X101-FPN
Weights model	R101	X-101-32x8d	R101	X-101-32x8d
Using masks	no	no	yes	yes

The selected backbones used are ResNet-101+FPN (R101-FPN) which is a Faster R-CNN and ResNeXt-101+FPN (X101-FPN) which is a Mask R-CNN. The weight model R101 is an adaptation of the original ResNet-101 model [9] and X-101-32x8d is a ResNeXt-101-32x8d model trained with Caffe2 [24]. In turn, the detection confidence threshold allows discarding detected objects which classification score value is lower than a given value.

Tables 3 and 4 reports the results of the considered detection metrics for the G8 and G19 videos, respectively. These videos were selected as they account for a representative traffic flow of the city in rush hours in the morning (G8) and in the night (G19). In these videos the camera angle is such that the North to South flow of vehicles occasionally occludes the vehicles in the West to East flow. Additionally, this scenario has a crossing with multiple traffic lights which makes the vehicles stop and accumulate producing interesting detection situations.

Table 3: Classification metrics obtained with different settings in video G8

configuration	*average accuracy*	*error rate*	*precision*	*recall*
R101-box-t03	0.93	0.07	0.91	0.73
R101-box-t05	0.87	0.13	1.00	0.48
R101-box-t07	0.78	0.22	0.93	0.23
X101-box-t03	0.93	0.07	0.89	0.75
X101-box-t05	0.88	0.12	1.00	0.49
X101-box-t07	0.79	0.21	1.00	0.24
R101-mask-t03	0.94	0.06	0.90	0.80
R101-mask-t05	0.93	0.07	0.97	0.72
R101-mask-t07	0.90	0.10	0.96	0.58
X101-mask-t03	0.93	0.07	0.89	0.79
X101-mask-t05	0.91	0.09	0.96	0.66
X101-mask-t07	0.91	0.09	1.00	0.61

Results in Tables 3 and 4 indicate that the proposed system had an overall average accuracy of 85% and indicate that the mask configurations computed better results than the box configurations in most cases (10 out of 12 instances). No significant performance differences between R101 and X101 models on box nor segmentation mask prediction were detected, but X101 had slightly better results in the night cases for segmentation.

Table 4: Classification metrics obtained with the different settings in video G19

configuration	*average accuracy*	*error rate*	*precision*	*recall*
R101-box-03	0.89	0.11	0.78	0.67
R101-box-05	0.82	0.18	0.89	0.40
R101-box-07	0.75	0.25	1.00	0.23
X101-box-03	0.84	0.16	0.70	0.53
X101-box-05	0.83	0.17	0.90	0.42
X101-box-07	0.73	0.27	1.00	0.21
R101-mask-03	0.87	0.13	0.67	0.67
R101-mask-05	0.85	0.15	0.67	0.60
R101-mask-07	0.86	0.14	1.00	0.49
X101-mask-03	0.89	0.11	0.77	0.70
X101-mask-05	0.87	0.13	0.92	0.56
X101-mask-07	0.87	0.13	0.96	0.53

5.5 Object tracking

The main parameter to consider is the *tolerance*, i.e. the number of frames before the algorithm concludes that a tracked object is no longer in the sequence. The tolerance value depends on the frame rate of the recording. Values corresponding to half, one, and two seconds were considered in the study, as they represent time windows in which the probability of a identity switch among detections is low. That is, 4, 8, and 16 frames for G8 and G19 videos, and 10, 20, and 40 frames for R8 and R19. The performance of the object tracking module depends on the detection module. The X101-mask-05 detection setting was defined as basis for all tracking tests as it was the best performing detection configuration.

Table 5: Tracking tests results

configuration	*MOTA*	*configuration*	*MOTA*
G8-t04	0.81	R8-t10	0.86
G8-t08	0.83	R8-t20	0.87
G8-t16	0.82	R8-t40	0.89
G19-t04	0.47	R19-t10	0.10
G19-t08	0.49	R19-t20	0.14
G19-t16	0.46	R19-t40	0.13

Results in Table 5 show average MOTA scores of 85% for the daylight cases and significantly lower (30%) for the cases in the night. This indicate that the module performs significantly better in daytime scenarios. In 3 out of 4 cases, the configurations with one second of tolerance (8 frames in G8 and G19, and 20 frames in R8 and R19) had slightly better results than for nighttime scenarios. *MOTA* was mainly affected by *FN* values. In the tracking evaluation this occurs when an existing vehicle is not detected in a given number of frames, therefore it is not tracked. Based on this observation, improving the detection module in bad lighting conditions would also improve the tracker performance.

5.6 Pattern analysis

For the evaluation of the counting module, the vehicle count resulting from the pipeline execution was compared to the number of vehicles obtained using manual counting. The performance of the method to count vehicles was measured using the detection and classification metrics presented above. To perform the evaluation, 30-second video segments were extracted from the four videos in the original test dataset. Four segments were considered for each video, thus having a total dataset of 16 segments from the two studied scenarios, eight taking place during the day and eight during the night.

Table 6 reports the results of the counting module evaluation, using the following configuration: the R101-mask-05 configuration was established for detection; the tracking module uses an IoU of 0.05, and a loss tolerance of eight frames for G8/G19 videos and 20 frames for R8/R19 videos.

Table 6: Counting test results

reference	*average accuracy*	*error rate*	*precision*	*recall*
G8	0.92	0.08	0.92	1.00
G19	0.52	0.48	0.62	0.76
R8	0.87	0.13	0.91	0.95
R19	0.19	0.81	0.27	0.33

Results in Table 6 show an average accuracy of 89% for the daylight cases and 36% for the cases in the night. This indicate that the counting module performs better under good lighting conditions. In this case there is still room to improve the classification of the counted vehicles as the comparison of *precision* and *recall* shows that the main source of error are the *FP*, This means that the counting algorithm correctly counts a vehicle, but classifies it in the wrong class. Under bad lighting conditions the performance is poor, this is also caused by the large number of *FP*, which in this case happens because the counting algorithm counts not existing vehicles. The performance of this module can be mainly improved by using a more precise classification model in the detection module, while a better detection under bad lighting conditions would also improve the performance of the counting module. Improving the classification under poor lightning conditions is one of the main lines for ongoing and future work.

6 Conclusions and future work

This article presented the design and implementation of a system for the analysis of traffic data using computational intelligence techniques.

The proposed system was developed following a flexible pipeline-type architecture built over the modern object detection library Detectron2. The proposed design provides an efficient frame processing, by using independent, configurable functional modules, loosely coupled between them. This feature allows including new methods, modifying existing ones, and evaluate different alternatives and configurations.

The problems of object detection and tracking are solved using the Detectron2 framework and the Node Moving Things Tracker library, respectively. The information generated by these modules from the traffic videos allowed implementing a set of modules for the collection and analysis of traffic data.

The validation of the proposed system is carried out using real videos from two scenarios that include important streets of Montevideo, Uruguay, under different conditions (daylight, nightlight, and different video qualities). These recordings were taken in rush hours and show an interesting flow of vehicles.

Results demonstrate the effectiveness of the system in scenarios with proper lightning conditions. The detection results shown a overall average accuracy of 85%, and better performance using the mask models. In object tracking, the average MOTA scores were 85% in daylight. Results dropped to 30% in nighttime, indicating that improvements are required to deal with bad lightning conditions. Similarly, the counting module performed an average accuracy of 89% in daylight and 36% in nighttime.

The main lines for future work are related to improve the object detection module training the models with bad lightning conditions or bad weather annotated examples. In turn, the experimental evaluation of the proposed system can be extended to consider the analysis of red light runs and no-stop zones modules. Another interesting line of work is related to developing more sophisticated pattern detection methods to capture relevant events such as abrupt lane change or even traffic accidents. We are working on these topics right now.

References

1. Arinaldi, A., Pradana, J., Gurusinga, A.: Detection and classification of vehicles for traffic video analytics. Procedia Computer Science 144, 259–268 (2018)
2. Bochinski, E., Eiselein, V., Sikora, T.: High-speed tracking-by-detection without using image information. In: 14^{th} IEEE International Conference on Advanced Video and Signal Based Surveillance. pp. 1–6 (2017)
3. Chauhan, M., Singh, A., Khemka, M., Prateek, A., Sen, R.: Embedded CNN based vehicle classification and counting in non-laned road traffic. In: 10^{th} Int. Conf. on Information and Communication Technologies and Development (2019)
4. Chavat, J., Nesmachnow, S.: Computational intelligence for detecting pedestrian movement patterns. In: Smart Cities, pp. 148–163 (2019)
5. Dey, S., Kalliatakis, G., Saha, S., Kumar Singh, A., Ehsan, S., McDonald, K.: MAT-CNN-SOPC: Motionless analysis of traffic using convolutional neural networks on system-on-a-programmable-chip. In: NASA/ESA Conference on Adaptive Hardware and Systems (2018)
6. Gilewski, J.: detectron2-pipeline: Modular image processing pipeline using OpenCV and Python generators powered by Detectron2. `https://github.com/jagin/detectron2-pipeline` (2019), [2020-03-15]
7. Girshick, R.: Fast r-cnn. In: Proceedings of the IEEE International Conference on Computer Vision (December 2015)
8. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., Chen, T.: Recent advances in convolutional neural networks. Pattern Recognition 77, 354 – 377 (2018)

9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015)
10. Leal-Taixé, L.: Multiple object tracking with context awareness. CoRR abs/1411.7935 (2014)
11. Leal-Taixé, L., Milan, A., Reid, I., Roth, S., Schindler, K.: MOTChallenge 2015: Towards a benchmark for multi-target tracking. arXiv:1504.01942 [cs] (2015)
12. Lou, L., Zhang, J., Jin, Y., Xiong, Y.: A novel vehicle detection method based on the fusion of radio received signal strength and geomagnetism. Sensors 19 (2019)
13. Mahamkali, N., Vadivel, A.: OpenCV for computer vision applications (2015)
14. Moussy, E., Mekonnen, A.A., Marion, G., Lerasle, F.: A comparative view on exemplar 'tracking-by-detection' approaches. In: 2015 12th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS). pp. 1–6 (2015)
15. Move-lab: Tracking things in object dectection videos. `https://www.move-lab.com/blog/tracking-things-in-object-detection-videos` (2018), [2020-03-15]
16. Nesmachnow, S., Iturriaga, S.: Cluster-UY: Collaborative Scientific High Performance Computing in Uruguay. In: Communications in Computer and Information Science, pp. 188–202. Springer International Publishing (2019)
17. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 91–99 (2015)
18. Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. In: Information Processing Management · July 2009 (2008)
19. Tilkov, S., Vinoski, S.: Node.js: Using javascript to build high-performance network programs. IEEE Internet Computing 14(6), 80–83 (2010)
20. Uy, A., Quiros, A., Bedruz, R., Abad, A., Bandala, A., Sybingco, E., Dadios, E.: Automated traffic violation apprehension system using genetic algorithm and artificial neural network. In: IEEE Region 10 Technical Conference. pp. 2094–2099 (2016)
21. Wu, Y., Kirillov, A., Massa, F., Lo, W., Girshick, R.: Detectron2. `https://github.com/facebookresearch/detectron2` (2019)
22. Yang, H., Qu, S.: Real-time vehicle detection and counting in complex traffic scenes using background subtraction model with low-rank decomposition. IET Intelligent Transport Systems 12, 75–85 (2018)
23. Yang, Z., Pun-Cheng, L.: Vehicle detection in intelligent transportation systems and its applications under varying environments: A review. Image and Vision Computing 69, 143 – 154 (2018)
24. Yangqing, J., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding (2014)
25. Zhang, K., Zhang, L., Yang, M.: Real-time compressive tracking. In: Computer Vision. pp. 864–877. Springer Berlin Heidelberg (2012)
26. Zhang, S., Wu, G., Costeira, J., Moura, J.: Fcn-rlstm: Deep spatio-temporal neural networks for vehicle counting in city cameras. In: International Conference on Computer Vision. pp. 3687–3696 (10 2017)
27. Zheng, M., Li, T., Zhu, R., Chen, J., Ma, Z., Tang, M., Cui, Z., Wang, Z.: Traffic accident's severity prediction: A deep-learning approach-based CNN network. IEEE Access 7, 39897–39910 (2019)
28. Zhou, Y., Nejati, H., Do, T., Cheung, N., Cheah, L.: Image-based vehicle analysis using deep neural network: A systematic study. In: IEEE International Conference on Digital Signal Processing. pp. 276–280 (2016)