UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Inteligencia computacional y aprendizaje para la predicción de incidentes de tráfico

Guillermo Gabrielli Ferreira

Ignacio Rafael Ferreira Urrutia

Pablo Dalchiele González

Programa de Grado en Ingeniería en Computación
Facultad de Ingeniería
Universidad de la República - Universidad del Sur de los Urales

Montevideo – Uruguay
Agosto de 2021

UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Inteligencia computacional y aprendizaje para la predicción de incidentes de tráfico

Guillermo Gabrielli Ferreira

Ignacio Rafael Ferreira Urrutia

Pablo Dalchiele González

Tesis de Grado presentada al Programa de Grado en Ingeniería en Computación, Facultad de Ingeniería de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de Grado en Ingeniería en Computación.

Directores:
D.Sc. Prof. Sergio Nesmachnow
D.Sc. Prof. Luis Ceiter

Montevideo – Uruguay
Agosto de 2021

Agradecimientos

Agradecemos a nuestros tutores, Sergio Nesmachnow y Luis Ceiter, por la asistencia y apoyo invaluables brindados durante la realización del proyecto.

Agradecemos a los profesores que nos han formado durante toda nuestra carrera y a la educación pública.

Agradecemos al Centro de Gestión de Movilidad de la IMM por los datos provistos que resultaron indispensables para la realización del proyecto.

Finalmente, agradecemos a nuestras familias y seres queridos por su apoyo y aliento incondicional.

RESUMEN

Este proyecto trata el problema de patrones de conducción no deseable en el tránsito, más específicamente la detección de eventos de cruce en luz roja, movimiento brusco y estacionamiento en doble fila. Se recolectaron videos de distintas cámaras de avenidas dentro de la zona metropolitana, los cuales fueron provistos por el Centro de Gestión de Movilidad de Montevideo para su análisis. Para la detección de los vehículos se utilizaron redes convolucionales, las cuales detectaban los vehículos en los frames de video. Se utilizaron también algoritmos de tracking y lógica para detectar cada evento. Se utilizaron también redes neuronales para la detección del estado de los semáforos, necesarios para el patrón de cruce en luz roja. Para el patrón de cruce en luz roja, se obtuvieron valores buenos de recall (0.63), si bien la precision no fue tan buena (0.42). Para el patrón de movimiento brusco, se obtuvieron buenos valores de precision y recall (0.82 en ambos casos), aunque fue necesario agregar pruebas sintéticas a las pruebas realizadas dada la dificultad para encontrar instancias del evento en los videos provistos. Para el patrón de estacionamiento en doble fila, se obtuvo un buen valor de recall (0.91) con un nivel aceptable de precision (0.59). A diferencia del patrón de movimiento brusco, en este caso se obtuvo una cantidad importante de datos a partir de los videos provistos por la Intendencia de Montevideo. Las principales contribuciones del proyecto son el entrenamiento de detectores de vehículos en videos de la zona metropolitana de Montevideo, junto con tres patrones de detección de conducción indebida de tránsito. Los patrones realizados permiten ser ejecutados en tiempo real contando con recursos de GPU, logrando buenos resultados en la detección de eventos lo que permite asistir en el análisis de la seguridad vial con asistencia computacional.

Palabras claves:
tránsito vehicular, visión por computadora, inteligencia artificial, redes neuronales convolucionales, sistemas inteligentes de transporte.

Lista de figuras

Tabla de contenidos

Capítulo 1

Introducción

En el área de análisis de tránsito vehicular, nuestro país cuenta con cámaras situadas en distintos puntos del país que permiten obtener información sobre como se comportan los vehículos en determinada zona. Las cámaras también permiten obtener información en caso de que ocurra algún accidente automovilístico, de forma de obtener de forma más objetiva las causas de cada siniestro.

La seguridad vial continúa siendo un problema que resulta de interés para distintos países. En nuestro país la seguridad vial sigue siendo una problemática relevante ya que se siguen realizando acciones para mantenerla controlada como puede ser la instalación de cámaras que detecten el exceso de velocidad, como fue reportado por Subrayado (2019).

A partir de la problemática de la detección de infracciones en el tránsito, se propone un sistema computacional que a partir de videos de cámaras de tránsito, sea capaz de detectar distintos tipos de conductas imprudentes realizadas por los conductores en tiempo real.

Algunas de las principales contribuciones del trabajo son la generación de modelos de detección de vehículos adaptados a videos de la zona metropolitana de Montevideo junto con modelos de detección de eventos de interés para controlar el comportamiento indebido de conductores en tiempo real.

Para la implementación de los detectores de eventos se utilizaron variantes de YOLOv3 (algoritmos de detección de objetos) luego de las cuales se aplica seguimiento de vehículos (tracking). Se calcula la velocidad estimada de cada vehículo a partir de una calibración que relaciona partes de la cámara con su posición física (latitud, longitud). A partir de los datos mencionados, junto a

información etiquetada de donde se encuentran los carriles en cada cámara, se aplica lógica adicional que permite detectar los patrones de interés. Estos patrones son cruce en roja, estacionamiento en doble fila y movimiento brusco. Se utilizó como fuente de datos videos de cámaras de la zona metropolitana provistos por la Intendencia de Montevideo.

El trabajo presenta las secciones de fundamentos teóricos (sección 2), metodología (sección 3), evaluación experimental (sección 4) y conclusiones y trabajo futuro (sección 5). La sección 2 presenta trabajos relacionados a la detección de vehículos, análisis de tránsito, rastreo de objetos y recolección de datos. La sección 3 presenta información sobre los conjuntos de datos generados, funcionamiento de los algoritmos utilizados y generados junto con otros elementos como las métricas de evaluación y herramientas de calibración generadas. La sección 4 presenta los experimentos realizados, incluyendo el entrenamiento de los modelos de detección de vehículos, el proceso de generación de tests y la evaluación de los patrones de detección de eventos. La sección 5 presenta las conclusiones del trabajo junto con posibles trabajos futuros que podrían mejorar los resultados.

Capítulo 2

Fundamentos teóricos

El capítulo de fundamentos teóricos presenta una revisión de literatura o estado del arte de temáticas de interés para el proyecto de grado. En este capítulo en particular, se presenta una revisión de literatura de la detección de vehículos, detectores de objetos, análisis de tránsito, rastreo de objetos y recolección de datos.

2.1. Análisis del estado del arte

Esta sección presenta los trabajos de distintos autores en cuatro temas relacionados a la temática del proyecto. Se realizó la búsqueda con foco en la detección de vehículos, análisis de tráfico, rastreo de objetos y recolección de datos como se presenta a continuación.

2.1.1. Detección de vehículos

Con respecto al problema de detección de vehículos se pueden identificar dos enfoques principales, por un lado métodos basados en procesamiento de imágenes como background substraction y aprendizaje no profundo y por otro lado recientemente han ganado popularidad enfoques basados en redes neuronales convolucionales (CNN). Dentro de los enfoques basados en CNN se pueden identificar dos categorías:

- Detectores basados en proposición de regiones: aquellos que identifican primero regiones que puedan contener objetos de interés para luego procesar las regiones con CNN.

- Detectores de un solo paso: aquellos que procesan la escena en una sola pasada. Los detectores de una sola pasada son suficientemente rápidos permitiendo el procesamiento en tiempo real en hardware modesto equipado con una GPU.

En la primer categoría destaca R-CNN de Girshick *et al.* (2014). Los autores utilizaron una etapa de selección de regiones (selective search) para producir aproximadamente 2000 regiones candidatas. Cada región es redimensionada a un tamaño fijo de 227 × 227 píxeles, de los cuales se extrae un vector de 4096 características utilizando cinco capas convolucionales y dos capas conexas, finalmente utilizando SVM sobre estas características para clasificar la región. En el conjunto de pruebas PASCAL VOC 2010 obtiene una precision de 53.7 %, superando métodos anteriores. Girshick (2015) mejoró considerablemente la velocidad al aplicar las capas convolucionales a toda la imagen una sola vez y obteniendo las regiones de interés a partir de la salida de estas. Ren *et al.* (2017) sustituyen la etapa de selección de regiones por una red convolucional que comparte las características convolucionales con la red de detección.

Kim *et al.* (2018) presentaron un enfoque híbrido que utiliza background substraction y una red convolucional simple para la detección de peatones. Los objetos de interés son detectados utilizando background substraction mediante el algoritmo de GMM, luego son redimensionados y clasificados por una CNN de 64×64 con dos capas convolucionales. El conjunto de prueba son vídeos de CCTV en Corea, provenientes de tres lugares: un sitio de juegos, un callejón y una calle poco transitada. Este algoritmo presentó mejores resultados en la detección de peatones que YOLO a una velocidad comparable. Los resultados no necesariamente se trasladan a detección de vehículos, pues la detección de objetos pequeños es un punto débil conocido de YOLO. Este enfoque tiene similaridades con R-CNN en que cada objeto de interés es procesado independientemente pero utiliza background substraction en vez de una etapa de selección.

Los trabajos principales en la segunda categoría son YOLO de Redmon *et al.* (2016) y SSD de Liu *et al.* (2016). YOLO innovó en utilizar una sola red neuronal para predecir tanto los cuadros delimitadores como la clasificación en etiquetas en una sola evaluación, modelándolo como un problema de regresión, a diferencia de métodos anteriores como R-CNN que realizan una etapa de generación de cuadros delimitadores potenciales para luego correr un clasificador sobre los cuadros delimitadores potenciales, o métodos como DPM

(modelo de partes deformables) que utiliza ventanas deslizables. Al ver toda la imagen permite utilizar información contextual global para la detección de cuadros y clases y aprender representaciones altamente generalizables. La información adicional obtenida redunda en una reducción en el número de falsos positivos y una alta resistencia a modificaciones en el formato de la entrada. Otra consecuencia es permitir el procesamiento en tiempo real al utilizar un pipeline simple. Los resultados experimentales en el test set PASCAL VOC 2007 obtenidos son superiores a otros sistemas en tiempo real, siendo superado solo por sistemas Fast R-CNN y Faster R-CNN combinado con VGG-16, con una velocidad inferior en más de un orden de magnitud.

Redmon y Farhadi (2017) describieron una serie de mejoras, la principal es el uso de *anchor boxes*, que corresponden a cuadros limitadores típicos para las clases entrenadas, su uso mejora significativamente el recall cuando existen múltiples objetos en la misma celda. Estas mejoras permiten obtener resultados competitivos con métodos del estado del arte a una velocidad considerablemente superior. Huang *et al.* (2018) describieron un derivado de Redmon y Farhadi (2017) que permite el procesamiento en tiempo real incluso en ordenadores sin GPU, aunque a costa de una pérdida significativa de precision. Chen *et al.* (2018) describieron ajustes al modelo para mejorar la detección de vehículos en vídeos de tráfico en autopistas. Liu *et al.* (2016) presentaron un detector de objetos con un enfoque similar a Redmon y Farhadi (2017) con resultados competitivos tanto en velocidad como precision.

Liu *et al.* (2018) describieron modelos de degradación de imágenes frecuentes y métodos de mejora de datos para hacer frente a ellos. En pruebas sobre señales de tráfico el modelo entrenado sobre imágenes sin degradación obtuvo una reducción en precision de 2.06 % con degradación de desenfoque, la adición de ruido, ya sea del tipo sal y pimienta (aquel que cambia el valor de los píxeles al mínimo o al máximo valor permitido) o gaussiano (que presenta una distribución continua) la redujo en 12.01 %, una rotación de hasta diez grados resultó en 24.23 %, mientras que el recorte la reduce en 27.96 %. Se repitieron las pruebas aplicando un solo modelo de degradación a las imágenes de entrenamiento, aplicando el mismo al conjunto de prueba. Para desenfoque la mejora fue de 1.54 %, para ruido fue de 11.09 %, para rotación fue de 0.93 %, mientras que para recorte fue de 8.24 %. Aplicar un modelo de degradación compuesto sobre el conjunto de entrenamiento resultó en una mejora

de 2.32 % (correspondiente a ocho imágenes) sobre un conjunto de pruebas de 334 imágenes capturadas para el estudio.

Ooi *et al.* (2018) exploraron el uso de un framework de rastreo de objetos en el cual asocian los objetos entre frames empleando una red profunda para clasificar los objetos en cada frame. Indica que el enfoque de asociación es mejor que el de background sustraction y optical flow ya que no son tan precisos cuando hay oclusión entre los objetos o los objetos se detienen. El análisis experimental se desarrolló sobre el Urban Tracker dataset utilizando las métricas CLEAR MOT. Los resultados más destacables fueron que el uso de un clasificador previo a un rastreador de objetos fue beneficioso para este último en su tarea. Como desventaja, la salida del clasificador en sí tiene muchos errores relacionados con objetos que no son de interés como autos estacionados, aunque se puede usar optical flow y background sustraction para mejorar la precision en casos de objetos estáticos.

2.1.2. Funcionamiento de YOLOv3 para detección de objetos

A partir de la investigación de trabajos relacionados se llegó a la conclusión de que YOLOv3 de Redmon y Farhadi (2018) o en su lugar alguna de sus variantes debían ser utilizadas en el proyecto de grado para obtener los mejores resultados en la tarea de detección de objetos. A continuación se presenta una descripción general de cada parte del proceso que realiza la versión original de YOLOv3 para detección de objetos.

2.1.2.1. Extracción de características

En YOLOv3 se utiliza una nueva red para realizar la extracción de característica llamada Darknet-53. Como su nombre indica, contiene 53 capas convolucionales, cada una seguida de la capa de batch normalization y activación Leaky ReLU. Utiliza bloques residuales para evitar la pérdida de detalle por pooling y el problema del desvanecimiento de gradiente. Un bloque residual (figura 2.1) consiste en dos capas donde la entrada al bloque se suma elemento a elemento (en el caso de Darknet) antes de la función de activación ReLU final. Utilizar la suma permite limitar el número de parámetros, generalmente sin causar pérdida de información, pues la misma ubicación espacial no suele

generar activaciones no relacionadas en distintas capas. Se utiliza una función de proyección lineal W_s para igualar el tamaño de la entrada con el de la salida, pues tienen tamaños distintos. Para reducir la resolución de los mapas de características en la versión tiny se utiliza una operación de pooling, mientras que en la versión completa de YOLOv3 se utiliza una capa convolucional con stride=2 para lograr el mismo efecto, pero con mayor flexibilidad a costa de un mayor número de parámetros.

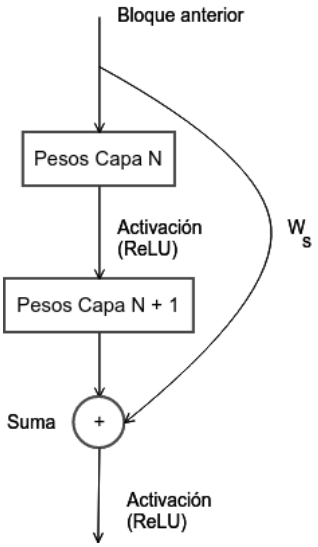

Figura 2.1: Ilustración de un bloque residual. Basado en He *et al.* (2016).

2.1.2.2. Predicción a diferentes escalas

YOLOv3 predice bounding boxes en tres escalas diferentes. Extrae características de esas escalas utilizando un concepto similar a las redes de atributos piramidales (FPN). A partir del extractor de características base los autores agregan varias capas convolucionales. La última capa de cada escala predice un tensor 3D codificando bounding box, objectness y las probabilidades de cada clase. En los experimentos realizados se predicen para cada cuadro de la grilla, para cada una de las escalas tres bounding boxes, un tensor resultante de dimensiones $N \times N \times [3 \times (4 + 1 + C)]$, utilizando cuatro para las cuatro coordenadas de cada bounding box, una para la predicción de objectness y C para predicciones de clase (donde C es la cantidad de clases). A continuación, se utiliza el mapa de características de la capa a distancia 2 hacia atrás de la actual y se les aplica un upsampling $\times 2$ a sus atributos. También se toma el mapa de características de la etapa previa de la red (escala previa) y se fusio-

na con las características a las que se les realizó upsampling, concatenándolas. Luego se aplican algunas capas convolucionales más para procesar el mapa de características combinado para predecir un tensor similar, aunque ahora con el doble del tamaño. Se aplica nuevamente el procedimiento mencionado para predecir las cajas en la escala final. Se utilizó k-means para obtener los anchors iniciales para la detección a partir del dataset COCO.

2.1.2.3. Predicción de bounding boxes

Como se mencionó anteriormente, la red predice bounding boxes como cuatro coordenadas, t_x, t_y, t_w, t_h. Si la celda está desplazada desde la esquina superior izquierda de la imagen por (c_x, c_y) y el bounding box tiene ancho y altura p_w, p_h, entonces las predicciones corresponden a las fórmulas 2.1, 2.2, 2.3 y 2.4 para b_x, b_y, b_w, b_h respectivamente. El símbolo σ representa aplicar la función sigmoide.

$$b_x = \sigma(t_x) + c_x \tag{2.1}$$

$$b_y = \sigma(t_y) + c_y \tag{2.2}$$

$$b_w = p_w * e^{t_w} \tag{2.3}$$

$$b_h = p_h * e^{t_h} \tag{2.4}$$

2.1.2.4. Predicción de objectness

YOLOv3 predice una puntuación de objectness para cada bounding box mediante regresión logística. Este valor debería ser 1 si el bounding box se superpone a un objeto del ground truth más que cualquier otro bounding box. Si alguno de los otros bounding boxes detectados que también estén superpuestos con el bounding box del ground truth no son los mejores pero tienen una intersección mayor que cierto threshold se ignora dicha predicción.

2.1.2.5. Predicción de clase

Cada cuadro de la grilla predice las clases que puede contener el bounding box mediante clasificación de múltiples etiquetas. Los autores no utilizan softmax ya que descubrieron que es innecesario para un buen rendimiento, en su lugar, utilizaron clasificadores logísticos independientes. Durante el entrenamiento, utilizaron como loss entropía cruzada binaria para las predicciones de clase. Esta formulación ayuda cuando se pasa a dominios más complejos como el conjunto de datos de Open Images. El uso de un softmax impone la suposición de que cada cuadro tiene exactamente una clase, lo que a menudo no es el caso. Un enfoque de múltiples etiquetas modela mejor los datos.

2.1.3. Análisis de Tráfico

Rabbouch *et al.* (2018) describieron la utilización de un índice de intensidad de movimiento para cuantificar la cantidad de actividad visual en videos de tráfico, así como un modelo estadístico basado en la noción de wavelet para unificar el número de vehículos con su coeficiente de movimiento. Este artículo resultó de interés para el presente informe ya que muestra muy buenos resultados a la problemática de estimar el nivel de congestión, permitiendo además predecir el nivel de tráfico a corto plazo. Los casos de estudio fueron realizados en carreteras de Riyadh principalmente, con diversas condiciones lumínicas tanto como climáticas, tomando porciones de la imagen en intervalos de cuatro segundos. Los resultados obtenidos en el informe mostraron que se obtuvieron muy buenos resultados prediciendo el nivel de tráfico en relación al bajo costo necesario para su implementación. No obstante, se detectaron picos inesperados del índice de movimiento en los días lluviosos, los cuales parecen ser causados principalmente por la alteración de la luminosidad en el fondo en días nublados, lluviosos o con neblina. Por esta razón una posible mejora al trabajo es tener en cuenta factores ambientales como son la luminosidad, estado climático, buscando prácticas que reduzcan sus efectos negativos.

Mou *et al.* (2019) presentaron una solución para la predicción de tráfico basado en el uso de redes recurrentes (más específicamente LSTM) utilizando información temporal además de información relacionada al tráfico. Se observó que el modelo obtenido obtuvo métricas mejores a todos los otros métodos eva-

luados aplicados a la predicción de tráfico y reparación de datos faltantes de tráfico. El artículo resultó de interés ya que utiliza un método que obtiene muy buenas métricas al momento de predicción y detección de la congestión de tráfico, superando a otros enfoques como las redes neuronales feed-forward. Utilizaron para la representación de instancias tuplas de dos coordenadas en la que cada una representa el tiempo o timestamp y el nivel de congestión, se podrían además utilizar otros atributos como entrada que permitan mejorar los resultados obtenidos. El conjunto de datos empleado fue información de tránsito del puente Shibalidian a Hongyan del este de Beijing en el período del primero de marzo al 30 de agosto de 2014. Se utilizó de marzo a julio para el entrenamiento y agosto para la validación. Se utilizó para la evaluación experimental intervalos de dieciséis minutos. A partir de los resultados obtenidos se puede observar un buen nivel de desempeño del modelo generado, superando otros modelos tanto clásicos (SVM, KNN) como más contemporáneos (GRU, LSTM ambos sin referencia temporal). Resulta útil agregar otro tipo de información a la hora de realizar la predicción como información climática o si es día festivo. Al referirse a las conclusiones se mencionó que un posible campo a explorar es intentar clasificar una red de carreteras y no solo el tráfico en una carretera particular, el cual puede ser de interés investigar en el presente proyecto.

Priambodo y Ahmad (2018) describieron la predicción de niveles de congestión en una carretera utilizando información de carreteras cercanas, utilizando redes neuronales, algoritmo k-means y regresiones múltiples. El método presentado es relevante ya que utiliza información de nodos cercanos para generar una predicción del nivel del tráfico, lo que provee otra dimensión que puede ser de interés en el proyecto. Inicialmente en el trabajo de los autores se realizó un proceso de recabar información para obtener datos sobre la correlación entre las distintas carreteras a utilizar como datos para el modelo generado. Para la evaluación experimental se obtuvo información de sensores IoT de Aarhus, Denmark. Con los datos de los sensores IoT, para horas particulares se determinó la correlación del tráfico entre carreteras cercanas. Los resultados obtenidos muestran una mejora de la performance utilizando clusters en relación a no utilizarlos para la predicción del tráfico. Puede extenderse los resultados utilizando redes LSTM, otros atributos relacionados al clima, distintas arquitecturas de redes además de utilizar esta metodología como complemento a

partir de la predicción del nivel de tráfico en puntos específicos. Si bien utilizó un modelo que considera el comportamiento de nodos cercanos para la predicción, la arquitectura de los algoritmos utilizados podría complejizarse en búsqueda de mejores resultados; dando margen para extender los métodos utilizando otros algoritmos, además de información adicional.

Lin *et al.* (2019) estudiaron el problema de la baja precision de métodos de predicción de tráfico a la hora de predecir escenarios inusuales (picos de cambio). Fue realizada una arquitectura llamada PSN (Pattern Sensitive Networks) la cual utiliza una red GAN para predecir el tráfico. Los autores agregaron además como variante varias funciones de costo adicionales a las redes GAN para mejorar los resultados. Los métodos presentados resultaron de interés ya que si bien son simples, presentan mejores tiempos de entrenamiento y eficiencia que otros métodos del estado del arte (LSTM, etc). Se realizó un análisis a partir de la fórmula de divergencia de Kullback-Leibler observando que al utilizar la probabilidad conjunta en lugar de la condicional, era posible obtener mejores aproximaciones del modelo con la predicción real. Para la evaluación experimental se utilizó un conjunto de datos de sistemas de mediciones de performance de Caltrans (California Department of Transportation) obtenido de detectores de bucle de inducción en autopistas. La información procesada fue la cantidad de autos en intervalos de cinco minutos. Se utilizaron los primeros nueve meses de datos para entrenamiento y los siguientes tres para validación. A partir de los resultados experimentales se puede ver que el modelo presentado obtiene mejores métricas que otros métodos del estado del arte para casos usuales (utilizando un quinto del tiempo de entrenamiento de los mismos). Para los casos inusuales se obtuvo que el método presentado quedó dentro de los tres primeros lugares junto con otros métodos tradicionales que no aplican redes neuronales (SVR, ARMA). Se indica que los resultados pueden deberse a que la mayor parte de los métodos tradicionales no prestan suficiente importancia a los outliers, en cambio los métodos SVR y ARMA si lo hacen. Otra opción es probar el mismo enfoque pero utilizando la idea de redes de tráfico no solo en un punto particular a la vez, lo cual no se ve en el artículo. Uno de los problemas es que los datos con los que más probablemente se dispongan para tener la cantidad de vehículos por ejemplo puedan contener ruido basado en la predicción del nivel de tráfico. En el caso dado los sensores utilizados disminuyen en gran medida este problema, el que de igual forma

puede presentarse con las fuentes de datos accesibles. Para el problema del ruido en la lectura del nivel de tráfico podrían utilizarse redes más complejas en caso de utilizar su arquitectura, tal como se menciona en su trabajo.

Yu *et al.* (2019) estudiaron el problema de predicción de tráfico a corto plazo, con enfoque a puentes marítimos en China. Los datos utilizados corresponden a la autopista Chang Tai. Uno de los aspectos más destacables del trabajo es que se logra una buena exactitud en las predicciones no solo en días normales sino en feriados también. El método utilizado fue analizar el flujo de autos y minibuses en intervalos de cinco minutos. Los datos de vehículos fueron recolectados por un sistema dinámico de detección de vehículos presente en el puente de la autopista Chang Tai. Entre los atributos usados están la fecha y hora, la velocidad, el carril y la aceleración y solamente se consideraron los vehículos en un sentido. Como preprocesamiento de los datos, realizaron un estudio de outliers en los datos, los cuales determinaron comparando la velocidad promedio y el flujo promedio de tráfico. Los outliers fueron corregidos promediando los puntos del día previo, el anterior al previo, una semana antes y dos semanas antes. La arquitectura utilizada fue una red neuronal con dos capas LSTM. Utilizaron 80 % de los datos para entrenamiento del cual apartaron a su vez un 2 % para validación durante el entrenamiento y el restante 20 % para evaluación. La evaluación se realizó usando datos en cuatro escenarios, día de semana, fin de semana, feriados y día lluvioso. Las métricas utilizadas fueron MAPE, MAE Y RMSE. En términos de MAPE el mejor resultado se dio en el caso de día lluvioso (13.85 % aunque todos los resultados dieron similar, en el entorno de 15 %). Uno de los problemas presentados en el trabajo es el relativamente corto horizonte de predicción de cinco minutos. Una posible mejora es extender a quince o 30 minutos con el fin de mejorar la utilidad en la previsión aunque probablemente se degraden las métricas.

Fan *et al.* (2019) presentaron un framework de ensemble en cascada para mejorar la exactitud de las predicciones de tráfico a corto plazo, disminuyendo el sesgo y la varianza. Este trabajo es pionero en la utilización de árboles aleatorizados (extremely randomized trees o extra-trees) mediante el uso de una técnica de boosting llamada EET, para enfrentar el desafío de la no estacionaridad de los datos de flujo de tráfico. El método utilizado consistió en preprocesar los datos con una media móvil ponderada para mejorar la exac-

titud, el cual consideran como la primer predicción de la cascada. Luego utilizaron una capa de entrenamiento con AdaBoost y extra-trees pero entrenan cada extra-tree en un subconjunto distinto y luego toman las predicciones de cada uno. A continuación se utilizan las predicciones de los extra-trees para ajustar la probabilidad de seleccionar los ejemplos para el conjunto de entrenamiento de la siguiente capa de entrenamiento con AdaBoost, de forma tal que los ejemplos que se predicen con más error tengan más probabilidad de ser seleccionados, mejorando así la exactitud en el entrenamiento final. El dataset que usaron fue el PeMS que incluye los atributos de flujo de tráfico, velocidad y tasa de ocupación de la calle, muestreados cada 30 segundos de autopistas de California. Se predijo el tráfico con un horizonte de cinco minutos, tomando intervalos de cinco minutos por lo que dividieron un día en 288 intervalos. Lo más destacable de los resultados es que el algoritmo EET presentado fue el que mejor predijo, por encima de otros métodos, el flujo de tráfico en un caso en el que el flujo de tráfico cambia bruscamente, como en el caso de un embotellamiento, lo cual es un ejemplo de no estacionalidad. Como mejora futura en el trabajo se mencionó el obtener más datos y atributos con el fin de mejorar las predicciones y enfocarse en datos faltantes y ruido en los datos. Una posible mejora es también extender el horizonte de predicción a quince o 30 minutos sin degradar las métricas considerablemente.

2.1.4. Rastreo de objetos

Una vez detectados los objetos de interés se puede realizar un rastreo de los objetos para obtener un seguimiento de su movimiento. Realizar rastreo permite obtener información de dirección y estimación de la velocidad del objeto. El rastreo de objetos permite solucionar casos de oclusión cortos.

Existen dos enfoques de rastreo de objetos:

- Rastreo libre de modelo: una vez identificada la ubicación inicial del objeto el rastreador se encarga de identificar la posición del objeto en cuadros subsecuentes. Generalmente están basados en un filtro que se entrena sobre la ventana inicial, el objeto se busca dentro de una ventana de búsqueda en el cuadro siguiente maximizando la correlación, una vez encontrado se actualiza el filtro para compensar cambios en la apariencia, que pueden ser causados por diferencias en el fondo, rotación, etc. (Bolme *et al.* (2010)).

- Rastreo basado en detecciones: el rastreador recibe detecciones del objeto en cuadros subsecuentes, por ejemplo a partir de un detector de objetos, limitándose a asociar detecciones entre cuadros a un mismo objeto. La ventaja es que si el proceso de detección es suficientemente bueno se evita el costo del seguimiento de objetos.

Dentro del enfoque de rastreo libre existen varios algoritmos. Los más destacados se comentan a continuación. Bolme *et al.* (2010) describieron un algoritmo para rastreo de objetos de baja complejidad, implementado actualmente como el método MOSSE en OpenCV. Es un rastreador basado en filtros. Para obtener la correlación se aplica una convolución entre la ventana y el filtro, la cuál se realiza en el dominio de Fourier por ser más eficiente pues equivale a realizar una multiplicación punto a punto. La conversión al dominio de Fourier presenta el problema de conectar artificialmente los bordes derecho e izquierdo, superior e inferior, lo que puede inducir a errores. Para reducir el problema de bordes se aplicó una función de ventana de coseno que reduce la intensidad de los píxeles cercanos al borde de la ventana hasta alcanzar cero, adicionalmente la imagen en la ventana se normaliza para evitar situaciones de bajo contraste que son más susceptibles a estos problemas. Para inicializar el algoritmo en el cuadro inicial se calcula un filtro en el dominio de Fourier como el que minimiza el error cuadrático entre los valores obtenidos y esperados de la convolución, este filtro se entrena sobre ocho pequeñas transformaciones afines al azar sobre la ventana inicial, generando los valores esperados correspondientes a un pico gaussiano con máximo en el centro de la ventana. En cuadros subsecuentes se aplica un procedimiento similar sin utilizar múltiples transformaciones de entrenamiento, posteriormente se realiza una media móvil exponencial entre el filtro obtenido del cuadro actual y el filtro anterior para adaptarse rápidamente a cambios de apariencia. Durante experimentos en varias secuencias, comparado con otros métodos contemporáneos (UMACE, ASEF, IVT, MILTrack, OAB y FragTrack) obtuvo resultados similares, pero alcanzó una velocidad de 669 fps utilizando un filtro de 64×64 px en un CPU a 2.4GHz, considerablemente superior a los otros rastreadores que obtuvieron velocidades en torno a 25 fps o inferiores.

Henriques *et al.* (2015) describieron el uso de un kernel no lineal rápido sobre matrices circulantes, matrices donde cada fila está rotada en un lugar a la derecha con respecto a la fila anterior. Se pueden generar estas matrices aplicando traslaciones a una imagen original, esto siempre resulta en matrices

diagonales, hecho que permite reducir fuertemente el costo de procesamiento en el espacio de Fourier, por lo tanto una traslación puede modelarse (con ciertos problemas de borde similares a los resueltos por Bolme *et al.* (2010)) como una fila dentro de una matriz circulante donde la fila inicial es la imagen buscada, esto reduce el problema de encontrar la traslación a encontrar la fila que más se asemeja es esa matriz. Este problema se resuelve al aplicar una regresión con un kernel no lineal mediante el método Ridge, por las propiedades de las matrices circulantes la complejidad de esta regresión se puede reducir de cuadrática a lineal. Esto redunda en que evaluar toda la ventana como una única matriz es mucho más eficiente que evaluar cada región por separado. Otra ventaja de este procedimiento es que permite combinar varios canales de características simplemente sumándolos en el dominio de Fourier. Este algoritmo se encuentra implementado en OpenCV bajo el nombre Kernelized Correlation Filter (KCF).

Lukežič *et al.* (2018) obtuvieron resultados cercanos al estado del arte en el VOT 2015 (Kristan *et al.* (2015)). No solo mantiene una velocidad cercana a tiempo real en una sola CPU sino que también introduce el concepto de un mapa espacial de confianza y de confianza de canal. Utilizó múltiples canales de características, veintisiete canales de HoG, diez canales que operan sobre color y un canal sobre escala de grises, que se combinan para obtener el resultado final. El mapa espacial de confianza es una matriz binaria que indica si un píxel forma parte del filtro aprendido por cada canal, el filtro es resuelto de acuerdo a un enfoque iterativo según la ecuación $h \cong h \circ m$, donde h es el filtro, m es el mapa de confianza y $\circ$ es el producto punto a punto, el mapa se estima mediante un enfoque bayesiano con regularización, con una distribución anterior que asigna una probabilidad alta a los filtros centrales y uniforme para los píxeles del borde. La confianza de canal consiste en un factor que indica el poder discriminativo de cada canal, usado como peso al combinar los canales. Este método se encuentra implementado en OpenCV bajo el nombre de CSRT.

2.1.5. Recolección de datos

Se optó por buscar en la literatura trabajos sobre la recolección de datos para conocer como generar datos de calidad y que resulten realistas. A partir de la búsqueda se obtuvo información sobre los siguientes trabajos.

En el trabajo de Arinaldi *et al.* (2018) se presentaron dos métodos aplicados a la tarea de detectar vehículos en el tráfico, junto con su velocidad y clase entre otros atributos. El primer método corresponde a MOG background subtraction y SVM, mientras que el segundo método es Faster Region Based CNN (RCNN). En el artículo se planteó el problema de generar un conjunto de datos etiquetados para utilizar RCNN, herramienta que también puede resultar útil para el trabajo. En el artículo se utilizaron dos conjuntos de datos, uno generado por el equipo que desarrolla el paper y otro de acceso público. En el primero de ellos se utilizan carreteras altamente concurridas en los peajes de Jagorawi, Kapup en Indonesia. Se utilizó una definición de 4096×2160 con veintidós frames por segundo. Para obtener los conjuntos de datos se debió ir hasta un puente de peatones que tenía vista a las carreteras y utilizar una cámara convencional. Se tomaron datos de casi una semana en los dos lugares, incluyendo videos nocturnos. El otro dataset utilizado fue el de MIT, el cual consta de secuencias de video de 90 minutos grabado por una cámara estacionaria, el video tenía una definición de 720×480 píxeles y estaba dividido en veinte clips. Los datasets presentados resultan de interés ya que no solo se utiliza el conjunto de datos generado, los cuales carecen de oclusión, giros en la carretera. En el dataset abierto se puede ver lo contrario, se puede ver clara oclusión, escenarios variantes y múltiples giros que puedan ser realizados, de igual forma los carriles no son del todo claros en ese caso. Se puede observar a través de los resultados obtenidos en el paper que los resultados del modelo presentado fueron mucho peores en los dataset generados por el equipo investigador. Se presenta como posible razón que se toman dos escenarios distintos y diferentes condiciones climáticas.

Temel *et al.* (2019) introdujeron el dataset de detección de señales de tráfico CURE-TSD, que contiene 49 secuencias de vídeo reales provenientes del dataset BelgiumTS y secuencias sintéticas creadas con la herramienta de desarrollo de videojuegos Unreal Engine 4. A cada secuencia se le aplicó doce condiciones de degradación simuladas, cada una en cinco niveles distintos: decoloración, desenfoque de lente, desenfoque gaussiano, error de codec, oscurecimiento, sobre-exposición, suciedad en el lente, ruido, lluvia, sombras, nieve y niebla. Las secuencias de BelgiumTS fueron etiquetadas con la herramienta VATIC, mientras que las etiquetas de las secuencias sintéticas se encuentran disponibles en su metadata. Se utilizó el conjunto de datos en la competencia

VIP Cup 2017, a partir de la cual surgieron en tres algoritmos finalistas con los nombres Neurons, PolyUTS, IIP y Markovians:

- Neurons detectó el tipo de degradación mediante una CNN basada en VGG16, utilizando el algoritmo de equalización local CLAHE para secuencias con bajo contraste, un CNN basado en ResNet para eliminar efectos de lluvia. Localizaron las señales de tráfico con un CNN de arquitectura U-Net desarrollada para el concurso, que luego son clasificadas mediante una CNN no profunda.
- PolyUTS utilizó GoogLeNet para extraer características que se envían a una red neuronal de regresión inspirada en Redmon *et al.* (2016).
- IIP utilizó la arquitectura de Faster R-CNN para la detección de tráfico.
- Markovians utilizó una CNN recurrente para determinar el tipo de degradación, Faster-CNN para localización, luego utilizando un CNN propia para clasificarlos. En el caso de suciedad en lentes se utilizaron filtros de Kalman para rastrear señales de tráfico, mientras que para otras degradaciones utilizaron el algoritmo de Lucas y Kanade (1981) basado en flujo óptico.

Detallando solamente esos cuatro finalistas, el algoritmo Neurons obtuvo los mejores resultados, seguido por PolyUTS y Markovians, obteniendo IIP el cuarto lugar. Al variar el nivel de degradación la perfomance de Neurons es afectada significativamente, pero el valor de F1 sigue superando a los competidores hasta el nivel cuatro, siendo superado por PolyUTS y Markovians en el nivel cinco. Con respecto al tamaño de la señal de tráfico la performance mejora considerablemente al aumentar el tamaño hasta alcanzar unos 90px, manteniéndose casi constante para tamaños superiores. Neurons obtuvo los mejores resultados para todos los tamaños, Markovians superó a PolyUTS para tamaños menores a 90px, mientras que PolyUTS es mejor para tamaños mayores a 90px por su superior recall.

A continuación se detallaran los tipos de modificaciones en las imágenes utilizados y su implicación en los algoritmos participantes. Con respecto al tipo de degradación, las sombras tuvieron el menor impacto para todos los finalistas aunque la nieve tuvo un impacto leve para todos los finalistas. La suciedad en los lentes tuvo un impacto medio para Neurons, mientras que fue leve para los demás. El desenfoque de lente tuvo un impacto medio para Markovians, leve para los demás. El desenfoque gaussiano tuvo resultados similares. El oscure-

cimiento tuvo un impacto medio para IIP y Markovians, leve para Neurons y PolyUTS. La decolorización tuvo un impacto medio para Markovians y leve para los demás. En relación a la sobreexposición, tuvo un impacto medio a elevado para todos los finalistas. Los errores de codec tuvieron un impacto similar a sobrexposición. La lluvia tuvo un impacto medio para PolyUTS, mientras que fue elevado para los demás, afectando especialmente a IIP. La niebla tuvo un impacto medio para IIP, pero tuvo el mayor impacto para los demás. La generación de ruido tuvo un impacto medio para Markovians y leve para los demás. Con respecto a la métrica F_1, Markovians obtuvo los mejores resultados en suciedad en lentes, PolyUTS obtuvo los mejores resultados en lluvia, IIP obtuvo los mejores resultados en niebla, mientras que Neurons mantuvo los mejores resultados para las otras categorías de degradación. En general todos los algoritmos tuvieron degradación de performance considerable bajo condiciones difíciles, por lo que es necesario mejorar la robustez de los algoritmos. Se encontró una correlación de Spearman alta para las métricas de precision y recall entre las secuencias reales y sintéticas, por lo que resulta válido utilizar simulaciones para predecir la robustez bajo condiciones reales. Una conclusión a destacar para el trabajo actual es que los algoritmos suelen ser más robustos ante oscurecimiento que ante sobreexposición, lo que favorece una velocidad de exposición más rápida. Para validar el método se utilizaron cámaras Flea3 con sensor CCD de 640×480 píxeles. Para controlar la exposición se varió el tiempo de exposición hasta alcanzar un máximo de 25.5 ms $\left(\frac{1}{39.2}s\right)$, aumentando la ganancia en caso de necesitar incrementar aún más la exposición. Se evaluó la performance de detección de peatones usando dos tipos de detectores, un modelo de partes deformables y SSD basado en CNN. Ambos métodos mostraron un aumento en la tasa de recuperación frente al número de falsos positivos frente a EA o EM. Asimismo se evaluó el efecto en odometría visual, que consiste en estimar la trayectoria del automóvil mediante vídeo proveniente de una cámara montada en el automóvil, la estimación a partir de la cámara controlada mediante este método fue considerablemente más precisa que en el caso de EA o EM.

Gauen *et al.* (2017) compararon diferentes datasets y frameworks comúnmente usados en aprendizaje automático y propusieron un método de creación de datasets usando imágenes obtenidas en tiempo real y georeferenciadas de cámaras alrededor del mundo. Como elementos para la compara-

ción se toma la ubicación y la relación tamaño del objeto/tamaño de imagen. Un aspecto a tener en cuenta es que la comparación la realiza para la clase "$persona$" y no la hace para vehículos. Los datasets que compara son PASCAL VOC, ImageNet, SUN, INRIA, KITTI, Caltech Pedestrian y COCO. La principal conclusión a la que llegan es que las personas suelen aparecer en el centro de la imagen. Con respecto a la relación objeto/tamaño de imagen se concluye que en general los objetos detectados no superan el 10 % del tamaño de la imagen, con la excepción de los datasets PASCAL VOC e ImageNet. En el caso de los datos de cámaras en red los objetos suelen estar menos centrados que en los datasets antes mencionados.

El método propuesto de creación de datasets consiste en una primera instancia generar una base de datos de cámaras IP alrededor del mundo, guardando la ubicación aproximada basándose en la IP. Las cámaras de esta base de datos pueden generar y analizar 97 millones de imágenes únicas en veinticuatro horas, sin embargo la anotación de estas imágenes exige mucho trabajo humano por lo cual proponen un método de anotación automático. El método consiste en aprovechar el hecho de que las cámaras capturan el mismo entorno bajo diferentes escenarios como día/noche, estación del año, etc. Además al conocer la ubicación se puede cruzar esta información para generar etiquetas, es decir, se crean etiquetas a partir de los metadatos de las imágenes, como "$tiene - árboles$" para una imagen de un parque. Dos de los principales problemas de este método es que en el caso de cámaras que cambien el ángulo de grabación, pueden aparecer objetos presentes que no se correspondan con la anotación generada por el método, y por otro lado puede haber mucha redundancia de información ya que en varias imágenes el fondo se mantiene igual.

Capítulo 3

Metodología

El objetivo de este capítulo es justificar el diseño metodológico elegido. La finalidad de una metodología bien descrita es explicitar los pasos mediante los cuales se obtienen los resultados, y por tanto, el cumplimiento (o no) de los objetivos establecidos, de manera tal que pueda ser replicado por otro investigador. En este capítulo se describe el tipo de investigación realizada, el modo de recolección de datos junto con una descripción de los algoritmos utilizados y el proceso de desarrollo de los algoritmos.

3.1. Estudio de bibliotecas de detección de objetos

Se realizó un análisis con el fin de seleccionar la biblioteca de detección de objetos a utilizar en el proyecto de grado. En la tabla 3.1 se presentan los atributos evaluados y los distintos valores para cada una de las bibliotecas consideradas.

	OpenCV	Keras	TensorFlow	PyTorch
	Rápida	Rápida	Baja	Media
Ecosistema	Sitio dedicado	Bastante bueno	Muy bueno	Muy bueno
Flexibilidad	Baja	Baja	Alta	Alta
Documentación	Buena	Buena	Buena	Buena
Eficiencia	Moderada	Muy buena	Muy buena	Bastante buena

Tabla 3.1: Atributos analizados para distintas bibliotecas que permiten realizar detección de objetos.

Para el análisis de las bibliotecas de detección de objetos se tuvieron en cuenta atributos tales como velocidad de prototipado, ecosistema, flexibilidad, documentación y eficiencia. Se puso mayor foco en velocidad de prototipado y en la eficiencia de las herramientas. Teniendo en cuenta la velocidad de protitipado y eficiencia Keras se presenta como la mejor alternativa. Dado que su ecosistema era bueno tanto como su documentación, se optó por elegir Keras como biblioteca para los modelos de detección de objetos.

Una vez tomada la decisión de que biblioteca utilizar se procedió a investigar repositorios que utilicen Keras junto con YOLOv3 para la detección de objetos. Se realizó una prueba de concepto de clasificación de imágenes para probar las bibliotecas disponibles, se utilizó primeramente YOLOv3 convencional utilizando una implementación en Keras Brownlee (2019). La velocidad de la implementación utilizada resultó baja ya que contiene operaciones implementadas de forma ineficiente, especialmente el código de Non Max Suppresion (NMS). Se intentó corregir las operaciones pero se llegó a la conclusión que realizar los cambios que mejoren el tiempo de ejecución requiere un esfuerzo muy grande, por lo que se optó por buscar una implementación base más eficiente.

Se analizó la implementación de YOLOv3 de Ponnusamy (2018) que utiliza operaciones de OpenCV que son muy eficientes para el procesamiento de imágenes. Se observó que el detector con un modelo YOLOv3 completo presentaba una velocidad de inferencia inferior a un fps en una computadora personal, por lo que se intentó utilizar una arquitectura tiny que obtuvo 6.24 veces menos tiempo de ejecución en comparación con la versión YOLOv3 original. Utilizar detectores con tiempo de procesamiento alto es una problemática ya que se busca con el proyecto de grado que la solución presentada pueda ejecutar en tiempo real. El inconveniente de la implementación es que el módulo de detección de objetos DNN de OpenCV basado en OpenCL, si bien era bastante eficiente en CPUs, no era eficiente en GPUs (la implementación de DNN en CUDA en su momento presentaba serias dificultades de compilación), por lo que finalmente se utilizó el módulo de detección de objetos basado en la implementación de Xiaochus (2018) en Keras, que es compatible tanto con GPUs de Nvidia como de AMD.

Asimismo se consideró el modelo Slim YOLOv3 Pengyi Zhang (2019), que funciona aproximadamente dos veces más rápido que el modelo YOLOv3 convencional. Se encontraba entrenado previamente para tráfico en imágenes to-

madas desde drones. Lo que significa que el modelo estaba más adaptado a la tarea para la cual se eligió utilizar, dado que las imágenes utilizadas en el entrenamiento eran similares a las obtenidas en el proyecto de grado. Si bien los resultados en la detección de objetos del modelo tiny resultaban inferiores a los de YOLOv3 en un análisis visual, los resultados fueron suficientes como para utilizarlo en la práctica.

3.2. Fuentes de datos obtenidas

Previo a contactar al centro de gestión de movilidad se realizaron pruebas con una cámara provista por Facultad de Ingeniería. Luego de las pruebas fue posible realizar grabaciones con la cámara a su máxima definición utilizando ZoneMinder. El setup estaba constituido de una laptop, un switch y la cámara. Para la recolección de los datos, la zona circundante a 21 de Setiembre y Gonzalo Ramírez presentó las cualidades de adquisición de datos que se buscaban para entrenar los algoritmos. Se contactó un comercio cercano a la zona mencionada para poder realizar grabaciones utilizando la cámara de vigilancia provista por Facultad de Ingeniería. Se intentó contactar el establecimiento en reiteradas ocasiones para poder grabar allí pero no fue posible. Dadas las complicaciones ocurridas para tomar datos de calidad se decidió contactar al centro de gestión de movilidad.

Para el etiquetado de datos se obtuvieron videos de ocho cámaras fijas de la Intendencia Municipal de Montevideo, los videos fueron grabados el 27 de Febrero del 2020 entre las 18:00 y 20:00 horas. Las cámaras fueron elegidas ya que están ubicadas en intersecciones relevantes de la zona metropolitana (zona de estudio). La ubicación resulta de interés para el proyecto de grado ya que las ubicaciones que presenten altos volúmenes de tráfico aportan mayor cantidad de información sobre tráfico. A continuación se detallan las calles de las cuales se obtuvieron videos:

- Bulevar Artigas y 21 de Setiembre
- Bulevar Artigas y Canelones
- Bulevar Artigas y Avenida Brasil
- Bulevar Artigas y José Enrique Rodó (mirando al norte)
- Bulevar Artigas y José Enrique Rodó (mirando al sur)

- Bulevar España y Libertad
- Avenida Sarmiento y 21 de setiembre
- Bulevar España y Avenida Sarmiento

Se obtuvo también una grabación de Bulevar Artigas y Avenida Italia, la cual no pudo ser utilizada para la detección de eventos ya que el video fue grabado con una cámara pivotante. Los nueve videos mencionados fueron utilizados para el etiquetado de datos. Se presenta en la siguiente imagen el mapa de la zona metropolitana con las ubicaciones de las cámaras de las cuales se utilizaron sus grabaciones 3.1. El mapa fue generado utilizando la capa base de OpenStreetMap.

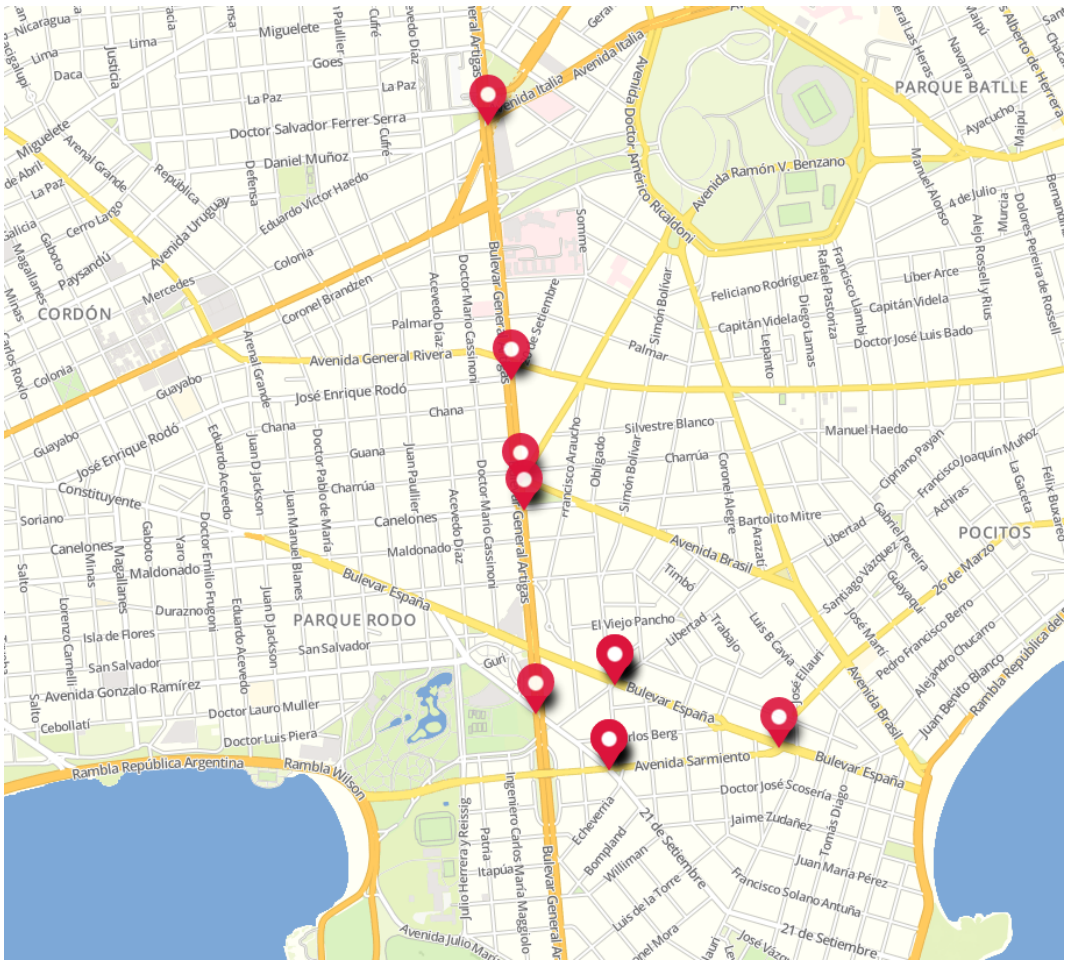

Figura 3.1: Mapa de la zona metropolitana de Montevideo con las ubicaciones de las cámaras utilizadas para generar las grabaciones.

3.3. Análisis de conjuntos de datos abiertos de vehículos

Se analizaron las anotaciones de los dataset VOC por Everingham *et al.* (2010), COCO por Lin *et al.* (2014) y VisDrone por Zhu *et al.* (2018), de forma de evaluar que cantidad de anotaciones se tienen de cada tipo de vehículo. Visdrone es el conjunto que posee más cantidad de vehículos etiquetados aunque los vehículos son muy pequeños en comparación al tamaño de la imagen. La distribución de los tamaños de los vehículos en los distintos datasets ya mencionados se puede observar en la figura 3.2.

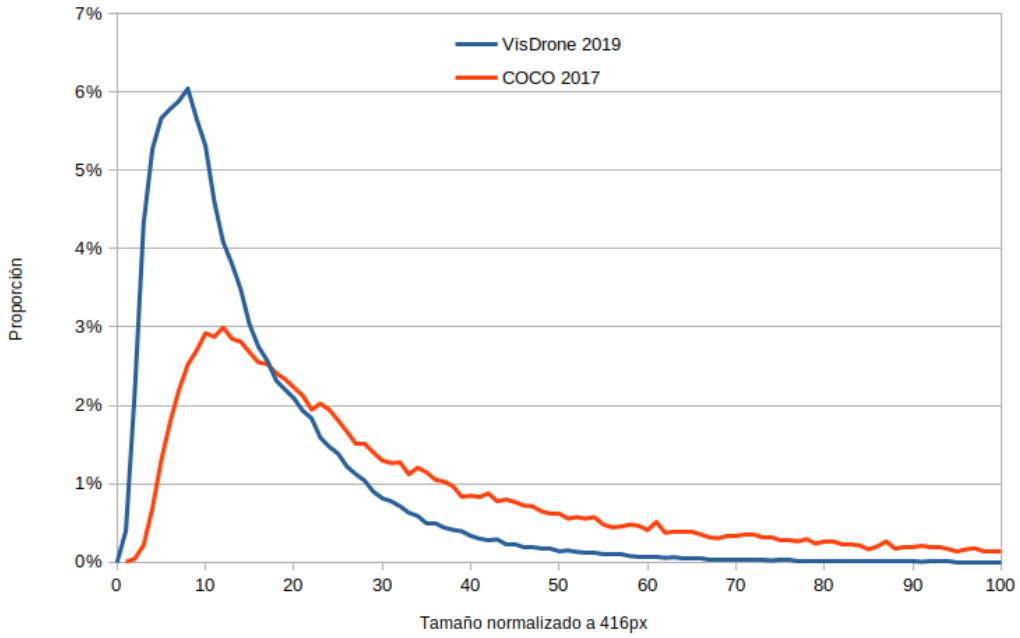

Figura 3.2: Distribución de las dimensiones (largo, ancho) para la clase car en ambos datasets (VisDrone, COCO).

Se entrenó la red YOLO-tiny-pr sobre COCO con ocho clases para evaluar el tiempo necesario para entrenar el modelo. Se obtuvo con una GTX 1050 mobile y resolución de 608x608 una duración de 9.5h para 10000 batches. Las clases utilizadas para el entrenamiento fueron: person, bicycle, car, motorbike, bus, truck, traffic light, stop sign.

Se detectó un problema en el entrenamiento ya que el programa utilizado para convertir las anotaciones de COCO no cambiaba los índices de las clases. Ejecutando YOLO preentrenado en COCO sobre las imágenes de VisDrone se

obtiene una asociación aproximada de las clases que no coinciden. La asociación se realiza eligiendo según la medida intersección sobre unión obteniendo las siguientes asignaciones.
De los triciclos (33 % detectados de los triciclos totales) se obtiene que el algoritmo los detectó como car 67 %, person 13 %, motorbike 9 %, truck 8 %. De forma similar para las vans (46 % detectados) el algoritmo las detectó como car 81 % truck 17 %.

3.4. Prototipo de capa de objetos

Para detección se eligió YOLO tiny ya que permite procesar más rápidamente los frames de los videos según Reyes *et al.* (2018). Las pruebas realizadas para YOLOv3 y YOLOv3 tiny se realizaron utilizando la implementación de YOLOv3 de Xiaochus (2018) y se realizó el entrenamiento con la implementación de AlexeyAB (2017).

El tracker Mosse obtuvo muy buenos valores de FPS y una eficacia de tracking razonable. Para el tracker Deep Sort tanto los FPS como la eficacia resultaron desfavorables. El tracker Sort presentó las mejores prestaciones tanto en FPS como en eficacia, por lo que fue seleccionado para el prototipo inicial. Sort usa intersection over union mientras que DeepSort además utiliza una red neuronal profunda con otras features extraídas de los objetos, lo que causa gran impacto en el tiempo de ejecución. El prototipo inicial combina la detección de objetos con el tracking de los objetos para su procesamiento en etapas posteriores.

3.5. Métrica de evaluación de detección de objetos

Para comparar distintos modelos se utiliza la métrica average precision (AP) por cada clase y la medida mean Average Precision (mAP) descripto por Henderson y Ferrari (2016), que consiste en el promedio de AP para todas las clases consideradas. AP se define como el área bajo la curva de precisión-recuperación Everingham *et al.* (2010).

La curva de precisión-recuperación es la curva en la cual se grafica la precisión y recuperación de los modelos en función de las detecciones generadas.

Tiene una pendiente descendente porque a medida que disminuye la confianza, se hacen más predicciones (beneficiando el recall) y predicciones menos precisas (perjudicando la precision). En muchos modelos incluyendo las redes neuronales se puede asignar a cada detección un nivel de confianza, que es una puntuación numérica, solamente se emite la detección si esta confianza supera cierto umbral configurable.

Para calcular la curva de precisión-recuperación, se deben tomar todas las detecciones, las cuales se ordenan según su confianza de forma descendente. Se procede iterativamente tomando una detección a la vez, se calcula precision y recall tomando todas las detecciones hasta ese punto y se genera un punto en el gráfico con esos valores de recall y precision. Dado que es posible obtener distintos números de falsos positivos para el mismo número de verdaderos positivos y por lo tanto, varios valores de precision para el mismo recall (por ejemplo cuando reducir el umbral solo aumenta las falsas detecciones), se toma la precision más alta obtenida para cada recall. Una vez calculados, se interpolan los puntos para obtener la curva de precision y recall. Para calcular el AP a partir de esta curva se puede aplicar integración obteniendo la métrica área bajo la curva (AUC), o se puede utilizar otra métrica a efectos similares. En este proyecto se utilizó el AUC con un umbral de IoU de 0.5, que es el método más simple. Everingham *et al.* (2010) utilizaron el promedio de precisión para los once valores de recuperación de 0.0 a 1.0 con incrementos de 0.1, sin embargo en iteraciones posteriores de la competencia VOC aplicaron una métrica similar a AUC. Lin *et al.* (2014) utilizaron 101 valores de recuperación y promedia entre diez distintos umbrales de IoU entre 0.5 y 0.95 para dar mayor peso a las detecciones que se asemejan más a las verdaderas. En la práctica los resultados obtenidos por los distintas métricas no resultan en ordenamientos significativamente distintos por lo que se consideró que la métrica AUC es adecuada.

El AP no depende del valor numérico de confianza que se le asigne a las detecciones sino de su orden relativo. Esta propiedad evita el problema en el que un nivel de confianza en un modelo puede corresponder a otro muy distinto en otro modelo. La curva de precisión-recuperación no requiere estrictamente que se pueda modificar el umbral de confianza, si la sensibilidad del modelo no es ajustable la curva resultante es una función constante. Se presenta a continuación un ejemplo de curva de precisión-recuperación para la clase vehicle utilizando el modelo tiny inicial. La figura 3.3 muestra un ejemplo de curva

de precisión-recuperación generada utilizando una herramienta de cálculo de la métrica mAP.

Una observación importante es que reducir el umbral mínimo de detección solamente puede mover detecciones del conjunto negativo al positivo y aumentarlo solamente del positivo al negativo, la recuperación es monótonamente creciente al reducir el umbral pues se define como verdaderos positivos sobre la suma de verdaderos positivos y falsos negativos, de esta manera el umbral controla la recuperación. En la práctica esta propiedad sobre la recuperación y el umbral de confianza permite obtener la curva mediante una única ejecución, no es necesario correr el modelo varias veces con distintos umbrales. Para obtener la métrica mAP de forma correcta es necesario configurar el umbral mínimo al mínimo valor posible.

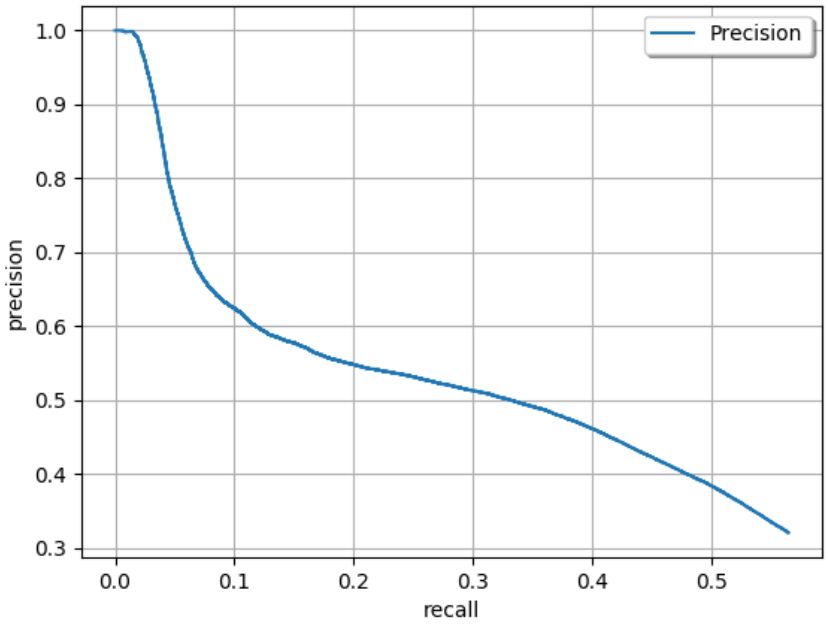

Figura 3.3: Curva de precision-recall para la clase vehicle del modelo tiny inicial 4.5 generada por la herramienta desarrollada por Padilla *et al.* (2020)

3.6. Etiquetado de datos

En la etapa de etiquetado de datos se utilizan los videos obtenidos en la etapa de recolección de datos, se analizan herramientas de etiquetado, eligiendo la más apta para la tarea y se describe el proceso por el cual se genera el conjunto de datos a utilizar en etapas posteriores.

3.6.1. Comparativa de herramientas de etiquetado de datos

Se evaluaron las herramientas de etiquetado Labelbox, YOLO_Mark y CVAT. Se decidió utilizar CVAT principalmente por la capacidad de interpolar imágenes. CVAT permite adicionalmente subir video, el cual es automáticamente segmentado en frames para ser etiquetados en la interfaz. YOLO_Mark fue descartado ya que no resultaba amigable para la tarea, ya que era necesario etiquetar imagen por imagen, no es colaborativa y no permite subir videos, su utilidad es principalmente para inspeccionar que el formato de etiquetado compatible con Darknet sea el correcto.

Se realizó una comparación en la velocidad de etiquetado entre Labelbox y CVAT etiquetando vehículos en el video de la zona de Tres Cruces. A partir de las pruebas obtenidas con ambas herramientas se obtiene que el tiempo promedio de etiquetado por imagen es 20.875 para Labelbox y 2.02 para CVAT. A partir de los resultados obtenidos en la comparación de velocidad de etiquetado se puede apreciar que si bien el soporte colaborativo de CVAT no es bueno, la velocidad de etiquetado con CVAT resultó diez veces mayor que la obtenida utilizando Labelbox. CVAT posee además la posibilidad de etiquetar de forma automática, la cual no fue posible utilizar en conjunto con la interpolación (no es compatible con interpolación en la práctica). A partir de los resultados de tiempo de etiquetado, CVAT fue la herramienta seleccionada para la tarea de recolección de datos. En la práctica se utilizó CVAT con interpolación de frames para etiquetar las imágenes de los videos.

Un atributo importante de CVAT es que no solo permite realizar etiquetado de bounding boxes con su respectiva clase sino que también permite etiquetar identidades en el tiempo (necesarias también para poder etiquetar con interpolación). El etiquetado con tracking de vehículos permitió que sin muchos cambios al conjunto de datos ya generado, se utilice la información de tracking resultante del etiquetado de detecciones también para la tarea de detección de eventos. Una de las razones que permitió reutilizar los datos para la detección de eventos fue que como los detectores de objetos pueden cometer errores en la detección, al momento de verificar si el evento detectado es correcto, resultaba necesario utilizar un ground truth para tracking de vehículos, el cual fue generado sin esfuerzo adicional para poder optimizar la velocidad de etiquetado

de detecciones. Se agrega a continuación una imagen de la interfaz de CVAT la cual se utilizó para realizar la tarea de etiquetado 3.4.

Figura 3.4: Screenshot de CVAT al etiquetar un video de 21 de Setiembre y Bulevar Artigas.

3.6.2. Desglose del conjunto de datos etiquetado

A partir de los videos obtenidos de las cámaras mencionados en la sección 3.2, se generó un conjunto de entrenamiento y validación. Para el conjunto de entrenamiento se utilizaron los videos de Bulevar Artigas y Avenida Italia, Bulevar Artigas y 21 de Setiembre. Para el conjunto de validación se utilizó también la porción restante de frames no utilizados para entrenamiento de Bulevar Artigas y 21 de Setiembre y los frames etiquetados de los videos adicionales mencionados previamente en la sección 3.2. A continuación se presenta un desglose de los datos etiquetados para entrenamiento y validación. Se incluyó para cada conjunto de datos los videos etiquetados junto con la cantidad de frames etiquetados:

- Etiquetado para entrenamiento
  - Bulevar Artigas y 21 de Setiembre (6000 frames)
  - Bulevar Artigas y Avenida Italia (1200 frames)
- Etiquetado para validación
  - Bulevar Artigas y Canelones (200 frames)
  - Bulevar Artigas y Avenida Brasil (200 frames)

- Bulevar Artigas y José Enrique Rodó (mirando al sur) (1000 frames)
- Bulevar España y Libertad (300 frames)
- Avenida Sarmiento y 21 de setiembre (200 frames)
- Bulevar España y Avenida Sarmiento (200 frames)

3.7. Data augmentation

Data augmentation es una estrategia que permite aumentar significativamente la diversidad de los datos disponibles al entrenar un modelo sin requerir datos nuevos. Se describen a continuación las herramientas de data augmentation utilizadas para los fines marcados en cada apartado.

3.7.1. Mejorar los datos de entrenamiento para obtener más variedad (mejorar entrenamiento)

Se utilizó en una primera instancia el código de Ultralytics Jocher *et al.* (2019) para realizar data augmentation ya que se vio que tenía buenas propiedades para realizar data augmentation. Como comparativa se utilizó a su vez el código de Darknet AlexeyAB (2017) para realizar el entrenamiento. Darknet poseía los mismos elementos de data augmentation como Ultralytics, poseía cambios en luminosidad, color, jitter (trasladar la imagen y tomar una parte de la imagen). A partir de los resultados obtenidos en el entrenamiento utilizando ambos métodos, se llegó a la conclusión de que utilizando Darknet se obtenían mejores resultados, por lo que continuó siendo la herramienta elegida para la tarea.

Se aplicó un preprocesamiento al dataset de VisDrone2019 con el fin de mejorar la calidad de los datos de entrenamiento y reducir el número de falsos positivos de tamaño pequeño que se observó inicialmente. Se calculó la mediana del tamaño de anotaciones de vehículos de la imagen en relación a la resolución de la imagen. Cuando la mediana es inferior al 3 % se aplica un procedimiento para obtener hasta tres imágenes ampliadas, se presenta un ejemplo del resultado en la figura 3.5. El procedimiento para mejorar la calidad de los datos de entrenamiento aumentando la cantidad de píxeles por bounding box se presenta a continuación:

1. Se aplica el algoritmo de K-means a los centros de cuadro delimitadores de cada anotación para obtener tres clusters.
2. Se calculan los centros de los clusters. Se genera una ventana sobre la imagen centrada en cada uno de los centros, correspondiente a un 50 % de la resolución de la imagen manteniendo el aspecto, sin reducir la resolución a un tamaño inferior a 416 píxeles en la dimensión menor.
3. Se desplaza la ventana de manera que la ventana quede completamente en la imagen.
4. En caso de que queden objetos fuera de la ventana, correspondientes al cluster asociado, se aumenta el tamaño de la ventana hasta alcanzar un máximo de 80 % de la resolución inicial.
5. En caso de que los centros obtenidos sean muy cercanos, lo que resulta en ventanas muy similares, se repite el procedimiento reduciendo el número de clusters.
6. Las anotaciones que hayan quedado truncadas se marcan como tales, omitiendo aquellas que hayan quedado mayormente truncadas, pues el dataset no contiene anotaciones con un truncamiento mayor al 50 %. En caso de que la anotación ya estuviera marcada como truncada se estima un truncamiento inicial de 25 %.
7. Se emiten las imágenes correspondientes a cada ventana si la mediana del tamaño de anotaciones en la ventana es superior al 2 % de la resolución de la ventana, de lo contrario se descarta la ventana.

3.7.2. Conversión de imágenes de día a noche modificando propiedades de la imagen

Se analizó como posibilidad utilizar herramientas de procesamiento de imágenes de OpenCV que modifiquen cualidades de las imágenes. De esta forma generando a partir de imágenes de día, imágenes análogas en la noche para realizar el entrenamiento. Se realizaron pruebas cambiando las propiedades de alpha, beta, gamma a distintos valores utilizando OpenCV. Se compararon los resultados obtenidos con imágenes de noche tomadas de los videos mencionados anteriormente. Si bien este método modificaba las imágenes de forma que el resultado fuera relativamente similar a las imágenes nocturnas, las luces de los autos aparecen mucho más brillantes en las imágenes nocturnas reales, las

Figura 3.5: Ejemplo de preprocesamiento sobre imagen de Visdrone, donde los rectángulos de color verde, rojo y azul representan las imágenes ampliadas correspondientes a cada cluster.

cuales no pudieron ser emuladas de forma fidedigna con este enfoque. Ya que modificar las imágenes de la forma descripta generaría datos de entrenamiento distintos a la realidad, se abandona el enfoque.

3.7.3. Conversión de imágenes de día a noche utilizando redes neuronales

Se exploró el uso de redes neuronales del tipo GAN las cuales, mediante el aprendizaje del espacio latente de las imágenes de día y de noche, permitirían generar imágenes nocturnas a partir de las diurnas que ya fueron etiquetadas. El uso de estas redes es de interés para aumentar la cantidad imágenes nocturnas etiquetadas sin requerir trabajo de etiquetado manual adicional. Para la tarea de generar imágenes nocturnas se investigaron las siguientes redes neuronales Unsupervised Image-to-image Translation Networks (UNIT) de Liu *et al.* (2018) y CycleGAN de Zhu *et al.* (2017) las cuales se detallan a continuación. Ambas redes se descartaron debido a que no se encontraron pesos pre-entrenados y el coste computacional de entrenarlas de cero a cualquiera de ellas era demasiado elevado.

3.7.3.1. Unsupervised Image to image Translation Networks (UNIT)

Los autores presentan la idea de un espacio latente compartido entre los dos dominios que supone que un par de imágenes correspondientes en ambos dominios pueden ser convertidas a una misma representación en un espacio latente compartido. Un aspecto que demuestran los autores es que idea de un espacio latente compartido implica la consistencia circular (cycle-consistency). La arquitectura contiene en su estructura Variational Autoencoders (VAEs) y Redes Generativas Adversarias (GANs) y consiste en 6 subredes: dos encoders de imágenes a representaciones de espacio latente E_1 y E_2, dos generadores de imágenes de cada dominio G_1 y G_2 y dos discriminadores de imágenes de cada dominio D_1 y D_2. Dichas subredes se conectan entre sí en distintas capas, lo cual se puede observar en la figura 3.6. Las redes $\{D_1, G_1\}$ forman una GAN para generar imágenes del dominio 1 y las redes $\{D_2, G_2\}$ lo hacen para el dominio 2. Las redes $\{E_1, G_1\}$ forman un VAE para el dominio 1 y las redes $\{E_2, G_2\}$ forman un VAE para el dominio 2 (en el caso actual podría considerarse que un dominio son las imágenes de día y el otro dominio las imágenes de noche). Para vincular los dos VAEs se comparten los pesos de las últimas capas de E_1 y E_2 y de las primeras capas de G_1 y G_2. Las redes $\{E_1, G_2\}$ convierten imágenes del dominio 1 al 2 y las redes $\{E_2, G_1\}$ lo hacen del dominio 2 al 1. De esta forma, G_1 puede generar dos tipos de imágenes: imágenes del dominio 1 reconstruidas por el VAE e imágenes del dominio 2 convertidas al dominio 1. Por lo tanto, el discriminador D_1 debe retornar verdadero para imágenes del dominio 1 y falso para ambas imágenes generadas por G_1 pero dado que el VAE del dominio 1 puede ser entrenado de forma supervisada, solo aplican entrenamiento adversario a las imágenes generadas por las redes $\{E_2, G_1\}$. De forma análoga sucede con el discriminador D_2. Finalmente, el entrenamiento de esta arquitectura consiste en resolver un problema minimax entre los encoders y generadores $\{E_1, E_2, G_1, G_2\}$ y los discriminadores $\{D_1, D_2\}$ donde la optimización busca encontrar un punto de equilibrio (punto silla). Los autores reportaron una accuracy de 0.9053 en la tarea SVHN→MNIST (traducir imágenes de números de puerta a dígitos).

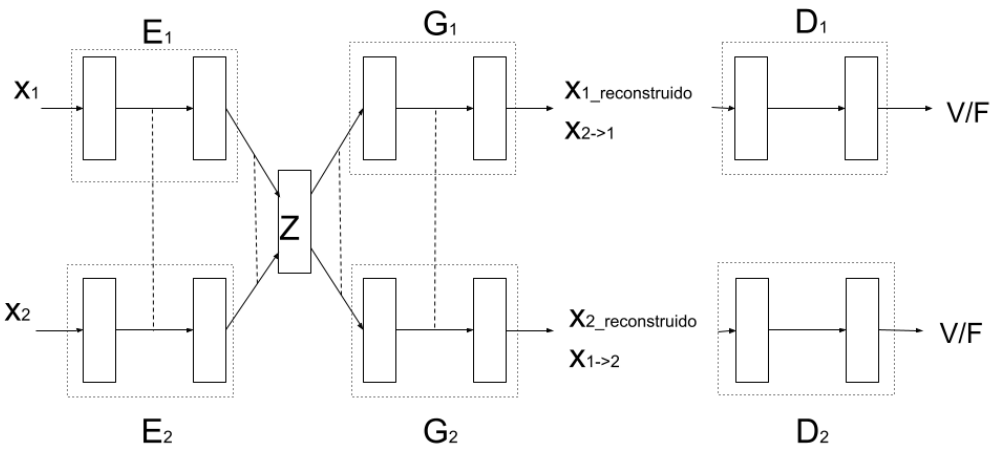

Figura 3.6: Esquema de la arquitectura UNIT. E_1, E_2, G_1 y G_2 son los encoders y generadores respectivamente. El espacio latente se representa en el esquema como Z, las líneas punteadas entre E_1 y E_2 representan los pesos compartidos entre las últimas capas de estas redes y las líneas punteadas entre G_1 y G_2 representan los pesos compartidos entre las primeras capas de estas redes. En ambos casos los pesos compartidos corresponden a capas con features de alto nivel. $X_{1\_reconstruido}$ y $X_{2\_reconstruido}$ son imágenes reconstruidas del dominio 1 y 2 respectivamente por sus respectivos VAEs y $X_{2\to1}$ y $X_{1\to2}$ son imágenes traducidas del dominio 2 al 1 y del 1 al 2 respectivamente. D_1 y D_2 son discriminadores de las GANs, que evaluan si las imágenes convertidas son realistas, para el dominio 1 y el dominio 2 respectivamente.

3.7.3.2. CycleGAN

A diferencia del trabajo de UNIT, los autores de CycleGAN no utilizan un espacio latente para la generación de imágenes. Los autores definieron dos funciones en donde X e Y son los dominios utilizados para cada imagen (en el caso actual podría ser X imágenes de día e Y imágenes de noche), la función $G : X \to Y$ y la función $F : Y \to X$. Definieron también dos discriminadores D_X y D_Y para los dominios X e Y respectivamente. Se definen dos funciones de pérdida de consistencia circulares (cycle-consistency loss), una para el caso $X \to Y \to X$ y otra para el caso $Y \to X \to Y$. La idea propuesta por los autores es que al transformar una imagen a otro dominio y luego transformarla al dominio original, debe ser lo más parecida posible a la imagen original, pero la imagen intermedia debe ser realista en su dominio (es considerada como de su dominio por el discriminador del dominio). En la figura 3.7 se puede ver el proceso de pasar de un dominio a otro y el proceso de calcular las funciones de pérdida al transformar la imagen desde cada dominio origen. Los autores reportaron una accuracy de 0.58 en la tarea etiquetas→foto (traducir de la

etiqueta de una foto a una foto que corresponda a la etiqueta) del dataset Cityscapes.

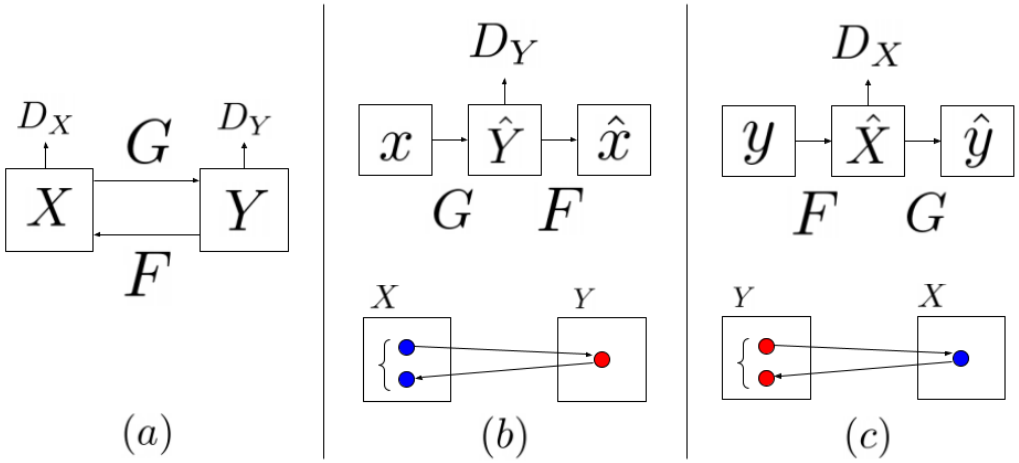

Figura 3.7: (a) Modelo que contiene las dos funciones anteriormente mencionadas (F y G), tanto como los discriminadores Dx y Dy. Dy motiva a G a traducir elementos de X en imágenes indistinguibles del dominio Y y viceversa para Dx y F. (b) función de pérdida de consistencia cíclica hacia adelante: $x \rightarrow G(x) \rightarrow F(G(x))$ y (c) función de pérdida de consistencia cíclica hacia atrás: $y \rightarrow F(y) \rightarrow G(F(y))$.

3.8. Anotaciones automáticas

Se entrenó inicialmente solo con los conjuntos de datos etiquetados. Posteriormente se realizaron entrenamientos con datos etiquetados automáticamente (corriendo el modelo YOLOv4 desarrollado por Bochkovskiy *et al.* (2020)), esto también se conoce como pseudo-labeling. Según hyun Lee (2013) el pseudo-labeling se basa en la hipótesis del clúster que afirma que las clases son significativamente distintas, es decir, existen pocos objetos que se encuentran cerca de la frontera entre clases. Cuando se cumple esta condición, es efectivo, pues ayuda a delimitar las clases al incrementar el número de ejemplos, pero en caso de no cumplirse puede introducir un mayor número de errores. En el problema de detección además existe la frontera entre los ejemplos positivos y los negativos, así que existe la posibilidad de que el pseudo-labeling mejore la detección pero afecte negativamente la diferenciación entre clases de ejemplos positivos. En el caso de YoloV3 existe sensibilidad al tamaño de los objetos por lo que se estimó que había potencial de mejora. Se utilizaron umbrales de confianza para cada clase elegidos manualmente a efectos de reducir el número

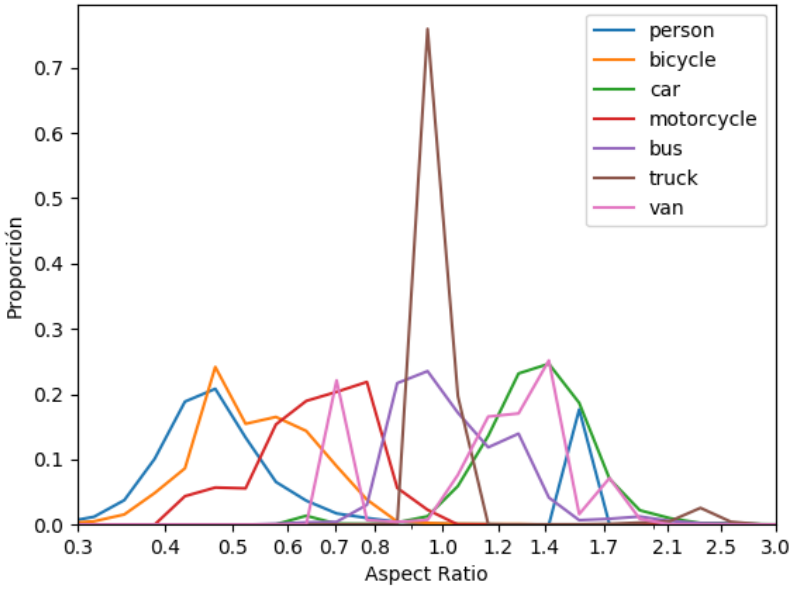

Figura 3.8: Histograma de aspect ratio para las clases person, bicycle, car, motorcycle, bus, truck, van para los primeros minutos de 21 de Septiembre y Bulevar Artigas (en una sola perspectiva)

de falsos positivos en los datos etiquetados automáticamente. En la tabla 3.2 se presentan los valores de confianza elegidos para cada clase en los videos de Tres cruces y Rodó al sur.

3.9. Corrección de detecciones

Examinando las detecciones del modelo YOLO tiny, se observó que el modelo presenta problemas para distinguir entre algunas clases. Se presenta a continuación el proceso realizado para la corrección de la problemática mencionada.

3.9.1. Aspect ratio

Se intentó en una primera instancia obtener información del aspect ratio de las clases problemáticas de forma de comprobar si es posible distinguirlas. En la figura 3.9 se presentan los aspect ratios obtenidos para el dataset Visdrone, mientras que en la figura 3.10 se presentan para los primeros minutos del video de 21 de Septiembre y Bulevar Artigas.

Clase	Tres cruces	Rodó al sur
bus	0.20	0.15
car	0.40	0.33
motorcycle	0.20	0.20
person	0.35	0.40
traffic light	0.20	0.15
truck	0.20	0.25
[otras]	0.20	0.20

Tabla 3.2: Umbrales de confianza mínimos para generación de anotaciones automáticas utilizando modelo YoloV4 COCO para las imágenes tomadas de los videos de Tres cruces y Rodó al sur.

En base a los histogramas obtenidos de los aspect ratios, se obtuvo que solo es posible dividir las observaciones entre las clases más pequeñas y las más grandes, como por ejemplo para persona, moto, bicicleta (con ciertas diferencias pero aún solapándose entre si) y por otro lado las demás clases (en estas el solapamiento de aspect ratios no permite discernirlas). La causa observada del problema consiste en que si bien algunos bounding boxes tienen tamaños distintos, la profundidad y la forma en que el objeto se presenta frente a la cámara impacta de gran forma el aspect ratio. En una sola perspectiva es posible clasificar a partir del aspect ratio hasta cierto punto, pero no es generalizable a múltiples perspectivas y direcciones de calle.

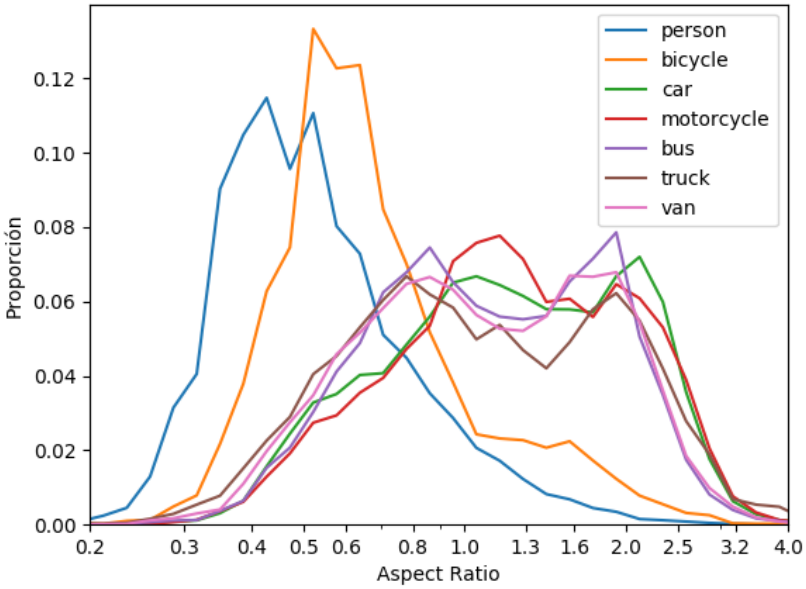

Figura 3.9: Histograma de aspect ratio para las clases person, bicycle, car, motorcycle, bus, truck, van para el dataset Visdrone.

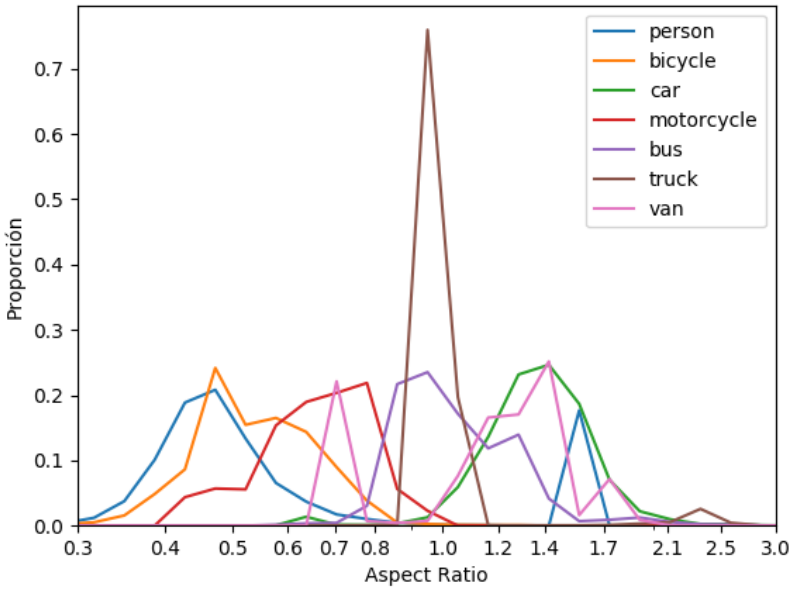

Figura 3.10: Histograma de aspect ratio para las clases person, bicycle, car, motorcycle, bus, truck, van para los primeros minutos de 21 de Septiembre y Bulevar Artigas (en una sola perspectiva)

3.9.2. Estimar tamaños de blobs (background sustraction)

Se evaluó utilizar background sustraction y operaciones de envoltura convexa de OpenCV para aproximar de forma más acertada la forma real de los objetos detectados. Utilizando esta técnica, se obtuvieron en ocasiones una mejor aproximación real del objeto (los detectores de objetos no proveen información de segmentación) pero el método no resultaba muy robusto. La baja robustez del método se debió a que se obtuvieron en algunas pruebas siluetas de elementos muy grandes resultado de unir varios objetos en uno. Finalmente se descartó utilizar este procedimiento, se optó por utilizar la velocidad distintiva de los objetos pequeños para refinar de forma más precisa su clase.

3.9.3. Corrección de clases utilizando velocidad máxima

En esta sección se presenta el procedimiento utilizado para mejorar la clasificación de vehículos utilizando su velocidad máxima. Adicionalmente se presenta el método inicial utilizado para estimar la velocidad, junto con elementos adicionales necesarios para la clasificación y el criterio de evaluación empleado.

3.9.3.1. Procedimiento de clasificación

Se obtuvieron datos de las velocidades máximas para las diversas clases, de forma de ver si era posible mejorar la clasificación. Se vio que si bien no era posible diferenciar la mayor parte de las clases a partir de la velocidad, las clases person, bicycle y motorcycle podían distinguirse a partir de su velocidad máxima. Estas clases en varias ocasiones eran confundidas por el detector. Se realizó un clasificador en base a datos estadísticos de las velocidades de cada clase. El clasificador toma la clase detectada y la velocidad máxima y retorna la clase corregida para cada objeto. Este clasificador como se mencionó previamente solo es aplicado a las clases separables.

3.9.3.2. Grilla de profundidad

Como forma de obtener más datos de los vehículos detectados, se optó por generar una grilla que a partir de los tamaños de los bounding boxes de la pantalla, estime la profundidad que se tiene en ese punto. Estimar la profundidad a partir de video fue posible ya que se conoce el tamaño real de

los autos, el que suele ser muy similar entre distintos autos. Se utilizó la clase 'car' ya que representa los vehículos más comunes y que pueden aportar más datos. A partir del dataset de 6000 frames etiquetado, se obtuvo la cantidad de objetos únicos. Había 162 autos sobre 217 vehículos totales, lo que conformaba el 75 % de los bounding boxes. A continuación en la tabla 3.3 se muestra la cantidad de objetos etiquetados en el video de 21 de Setiembre y Bulevar Artigas.

Etiqueta	# Objetos únicos
car	162
bicycle	8
motorcycle	8
bus	7
truck	7
van	25
Total	217

Tabla 3.3: Cantidad de objetos etiquetados en el dataset de 6000 frames del video de 21 de Setiembre y Bulevar Artigas

A partir de una imagen de entrada, se genera una grilla de $n \times n$ celdas, en la que para cada una de las celdas se guarda el tamaño en y de cada uno de los bounding boxes por clase. Para cada clase se guarda una cantidad k de bounding boxes. Se utiliza para cada celda, los tamaños de bounding boxes obtenidos y el tamaño real aproximado de los autos para estimar la profundidad real en ese punto.

3.9.3.3. Normalización de vector de desplazamiento utilizando mapa de profundidad

Utilizando el tracker, para cada objeto al cual se aplica tracking puede obtener su vector de desplazamiento en dos frames distintos. Se quiere a partir de este desplazamiento estimar la velocidad real. Fue necesario considerar que distintos desplazamientos significan distintas velocidades dependiendo de la posición del vehículo en la calle. La figura 3.11 muestra como la información de profundidad es actualizada al pasar los vehículos por la calle de 21 de Setiembre y Bulevar Artigas. Se puede apreciar como al aumentar el valor de la profundidad, el color de los valores se vuelve más oscuro, permitiendo evaluar el método de forma visual.

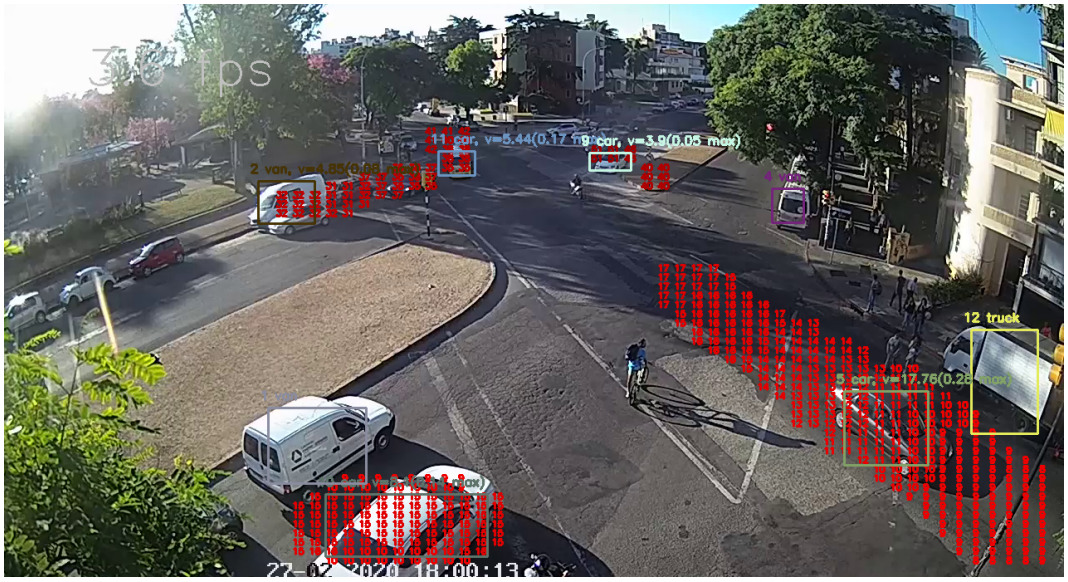

Figura 3.11: Grilla de profundidad para el video de 21 de Setiembre y Bulevar Artigas

En relación al tamaño de los autos se utilizó el recurso Auto-Data.net (2010) para obtener el tamaño promedio. El tamaño estándar se encontraba entre 1.40m-1.60m, a partir del rango de longitudes obtenido se utilizó el valor medio (1.50m). Se debió agregar luego un factor experimental viendo el tamaño de las personas, aumentando en 40 % para tener en cuenta que la coordenada y es mayor por la perspectiva. La razón de este factor se debe a que en las imágenes obtenidas por las cámaras de los autos, se debe tener en cuenta la parte del techo al considerar su altura por la perspectiva. Teniendo en cuenta los cambios relacionados a la perspectiva se utilizó el tamaño de las personas como validación para realizar el ajuste de parámetros.

La aproximación utilizando la grilla de profundidad resultó inestable al utilizar las alturas de los autos, lo que generaba que la velocidad estimada oscilara de forma brusca. Para solucionar la problemática se utilizan las profundidades circundantes, las cuales se interpolan para obtener el valor resultante.

Se decide también aumentar la densidad de la grilla para disminuir esta problemática. Dado que que se estaba utilizando el centro del bounding box para actualizar la grilla, al haber más puntos en la grilla la cantidad relativa de datos era menor. Para combatir el problema de la falta de datos para completar la grilla se decide que las actualizaciones de la grilla de profundidad sean utilizando el área completa del vehículo (se actualizan todas las celdas que intersectan con el vehículo).

La cantidad de datos recabados por clase en cada celda de la grilla fue elegida de forma que sea representativa (la cantidad no debe ser muy baja) y no debe generar problemas de performance (exceso de memoria almacenada). Los cambios realizados permitieron entonces sin tener gran impacto en la performance, obtener un estimativo de la profundidad para cada vehículo de la pantalla.

3.9.3.4. Historial de vehículos

Para cada vehículo se mantiene un historial, el cual permite obtener la clase más común de todas las detecciones provistas por el detector de objetos. A partir de la clase más común es posible, utilizando la corrección por velocidad, obtener mejores resultados de clasificación en relación a la clase asociada. Para cada historial, el cual representa un vehículo en el tiempo, se mantiene cual es su velocidad promedio y su velocidad máxima, lo que se utiliza para luego corregir su detección.

3.9.3.5. Método de evaluación

Para evaluar los resultados de las correcciones en las clases se opta por ejecutar el algoritmo sin la corrección de clases y otra instancia con el corrector basado en velocidad. Luego de terminada la ejecución del segundo algoritmo, se corrige la clase generada utilizando los historiales. Con los dos conjuntos de detecciones y las anotaciones generadas para el video, se comparan utilizando el repositorio de Padilla *et al.* (2020) para obtener el mAP de ambas anotaciones.

3.10. Herramientas generadas para etiquetado de datos

Se desarrollaron herramientas auxiliares para realizar el etiquetado de información útil para los detectores de eventos. A continuación se detalla su guía de uso. Los elementos etiquetados fueron carriles, semáforos, cruces, mapeo de puntos de la cámara a un plano cartesiano, entre otros.

3.10.1. Herramienta para dibujar carriles, semáforos y cruces

Esta herramienta consiste en una interfaz gráfica en la cual se pueden dibujar los polígonos de los carriles, semáforos y cruces de cada cámara. Para ello, primero se carga el archivo .png o .jpg de un frame de la cámara mediante el botón *Cargar Frame*. Luego se definen los carriles seleccionando el radio button *Carriles* y dibujando los distintos carriles en la ventana abierta con el frame que se cargó, haciendo click en los puntos que definen el polígono, señalando el último punto mediante doble click. Para eliminar el polígono de un carril, se hace click derecho sobre un punto interior de este polígono y se presiona la tecla suprimir. De forma análoga se definen los semáforos y cruces. Para agregar los carriles que están asociados a un semáforo, o sea los que el semáforo habilita el paso con la luz verde, se selecciona el semáforo y el carril y se hace click en el botón *Agregar* de *Carriles asociados a semáforos*, de forma similar se hace para un carril perpendicular a un semáforo, es decir, los carriles a los que el semáforo no les habilita el paso al estar en verde. Por último se pueden definir agrupaciones de carriles, por ejemplo, los carriles de un sentido de una calle o avenida. Para agregar una agrupación de carriles se debe agregar primero los grupos de carriles con el botón *Agregar grupo de carril* y luego seleccionando el carril del combo Carril y el botón *Agregar carril a grupo*. Finalmente se puede exportar la información a un archivo .json para su uso en los patrones con el botón *Exportar a json* el cual permite elegir la ubicación de guardado. Para cerrar la herramienta es necesario seleccionar la pantalla con el frame cargado y presionar escape y luego cerrar la ventana de la herramienta. En las figuras 3.12 y 3.13 se puede ver la herramienta generada para poder etiquetar de forma manual los carriles semáforos, cruces, grupos carriles entre otros.

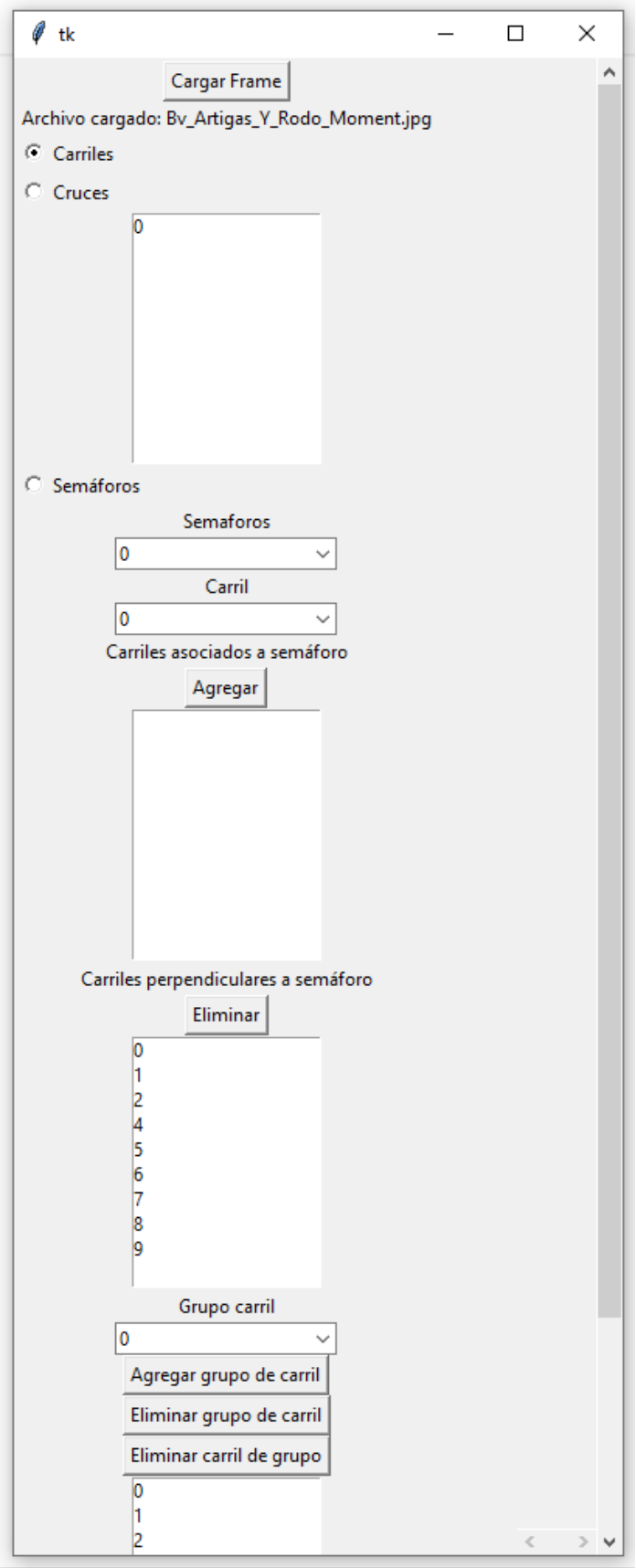

Figura 3.12: Herramienta para dibujar carriles, semáforos, cruces y grupos carriles.

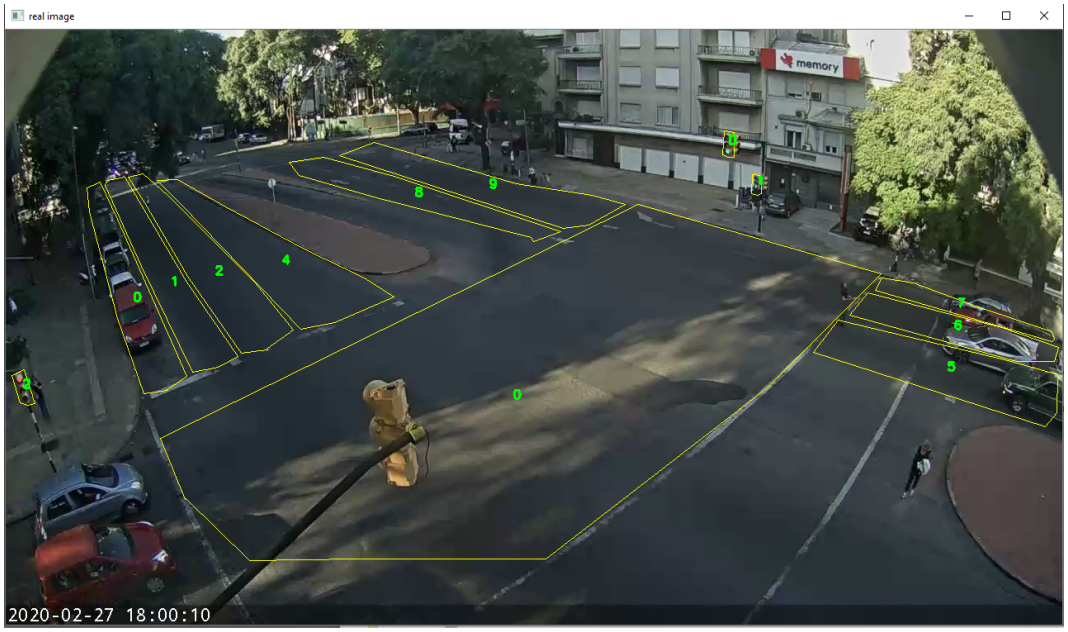

Figura 3.13: Ventana de herramienta para dibujar carriles, semáforos, cruces y grupos carriles

3.10.2. Herramienta para calibrar mapeo de coordenadas de la pantalla a coordenadas geográficas

Esta herramienta consiste en una interfaz gráfica similar a la anterior, en la cual se dibujan los puntos para el mapeo de coordenadas geográficas a píxeles. Para la tarea de calibración, primero se carga el archivo .png o .jpg de un frame de la cámara mediante el botón *Cargar Frame*. Después se carga el archivo .kml previamente exportado de Google Earth que contiene los cuatro puntos necesarios para la transformación. Luego se definen los puntos en el frame cargado mediante clicks. Notar que los puntos definidos deben formar un polígono que cubra en lo máximo posible el área de la cámara. En el frame se pueden visualizar las coordenadas en píxeles de los puntos que se van definiendo. Para eliminar los puntos, estos se pueden seleccionar en el combo *Puntos en OPENCV* y clickear el botón *Eliminar punto*. Por último se puede asociar y desasociar los puntos geográficos a los puntos en píxeles mediante el botón *Agregar asociación* y *Eliminar asociación* respectivamente. Finalmente, con el botón *Exportar a json* se puede exportar la información a un archivo .json para ser usada en los patrones. Para cerrar la herramienta es necesario seleccionar la pantalla con el frame cargado y presionar escape y luego cerrar la ventana de la herramienta en si. En la figura 3.14 se muestra la vista de Google

Earth en la que se seleccionan los puntos geográficos que serán utilizados luego para realizar el mapeo. En la figura 3.15 se muestra el proceso de exportar los puntos geográficos etiquetados a un archivo KML. En las figuras 3.16 y 3.17 se muestra la herramienta generada para realizar el mapeo de puntos de la pantalla a puntos geográficos utilizando la información generada previamente con Google Earth.

Figura 3.14: Ejemplo de selección de puntos en Google Earth, captura de pantalla de parte de la ventana del programa

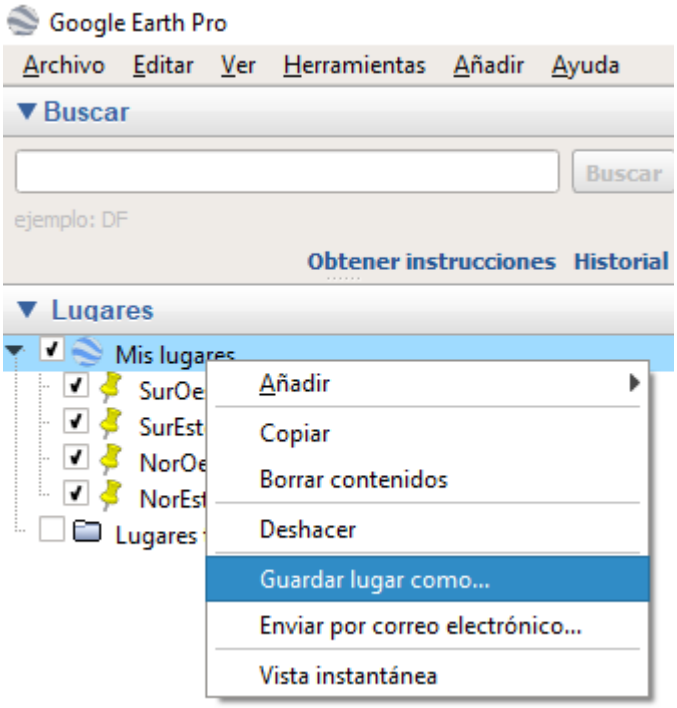

Figura 3.15: Ejemplo de guardado de archivo kml con los puntos en Google Earth, se debe seleccionar en formato ".kml"

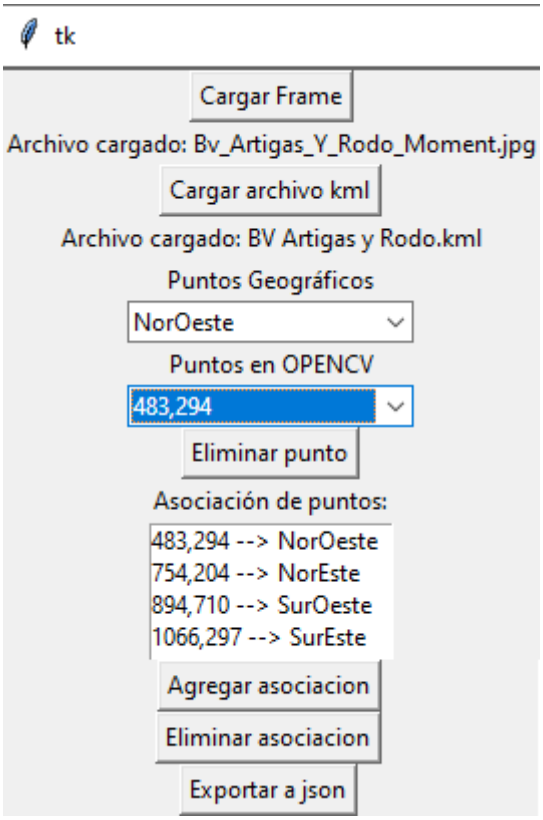

Figura 3.16: Herramienta para seleccionar coordenadas

Figura 3.17: Ventana de herramienta para seleccionar coordenadas

3.11. Detección de eventos

En esta sección se presentan los algoritmos de detección de eventos desarrollados, su funcionamiento, los desafíos ocurridos durante su implementación y como fueron solucionados.

3.11.1. Movimientos bruscos en el tránsito

En esta sección se presenta el proceso llevado a cabo para la creación del algoritmo de detección de movimientos bruscos en el tránsito. Un movimiento brusco de un vehículo en el marco del proyecto de grado representa la situación en la que un vehículo realiza una maniobra brusca a alta velocidad.

3.11.1.1. Grilla de vectores

Se generó inicialmente para la clasificación de movimientos bruscos una grilla de vectores. La grilla utiliza los vectores de movimiento promedio de los vehículos cercanos en cada celda de la grilla para obtener el vector representativo de cada celda. La intuición detrás fue que si un vehículo se desplaza de forma muy diferente a la forma del flujo normal, el movimiento podía considerarse brusco o anormal.

En la grilla solo se utilizan los vectores de movimiento si los vectores sobrepasan cierta norma (para evitar que los vectores cercanos a cero en módulo modifiquen el flujo normal registrado). Cada n frames se obtiene una muestra calculando para cada cuadro de la grilla el promedio entre los últimos K datos obtenidos. En la figura 3.18 se presenta un render de los vectores generados utilizando la grilla de vectores en un video de 21 de Setiembre y Bulevar Artigas.

No se logró con este método clasificar los vehículos dado que no fue posible obtener un método de clasificación a partir de los datos generados con la grilla de vectores. Debido a este resultado, la grilla de vectores no fue utilizada en la solución final.

3.11.1.2. Derivada del ángulo de movimiento, aceleración del vehículo

El segundo enfoque utilizado fue el de utilizar la derivada de ángulo de movimiento y la aceleración de cada vehículo. Luego de realizar los cambios

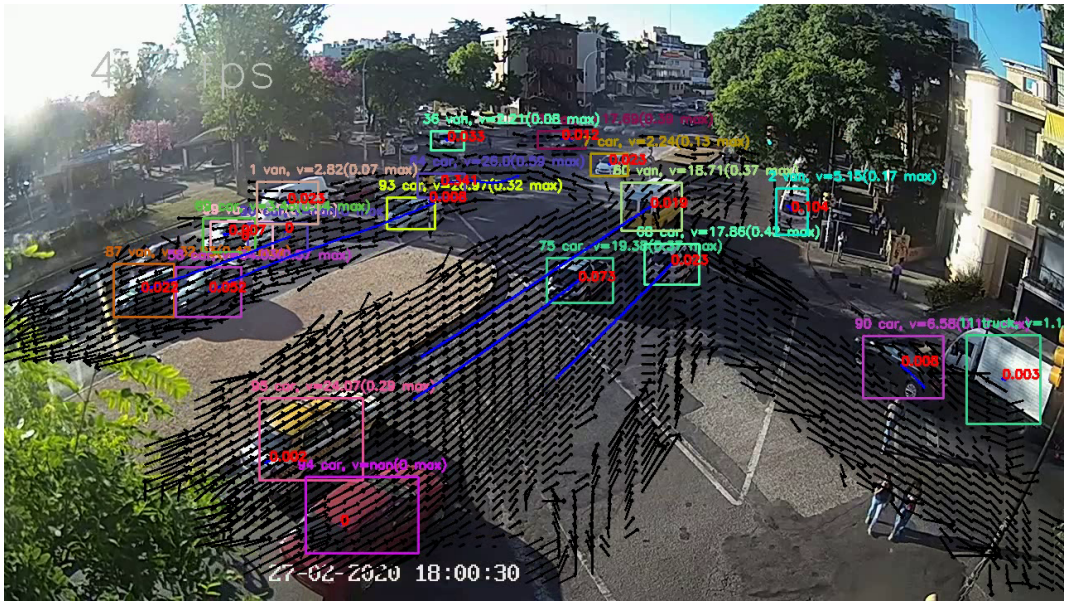

Figura 3.18: Grilla de vectores en video de 21 de Setiembre y Bulevar Artigas.

para poder obtener ambos valores para un vehículo en el tiempo, se llegó a que los datos obtenidos resultaban tener mucho ruido para la clasificación. Por esta razón fue necesario utilizar los parámetros que se describen a continuación, que resultaron ser los utilizados en la solución final.

3.11.1.3. Detección de giro, velocidad mayor a la permitida, cambio de carril para la clasificación

A continuación se menciona cual fue el procedimiento para obtener cada uno de los atributos utilizados en la predicción del evento movimiento brusco. Se agregan también problemas ocurridos, su mitigación y como se combinan los atributos obtenidos para realizar la clasificación final.

3.11.1.4. Detección de giro

Para detectar cuando un vehículo se encuentra realizando un giro se realizaron distintas pruebas utilizando varios enfoques. En una primera instancia se utilizaron las detecciones para n frames. Se calcula para las detecciones la suma de la variación de ángulos detección a detección. Si la variación de ángulos superaba los 90 grados entonces el vehículo se encontraba realizando un giro.

Este enfoque no presentó resultados positivos ya que los datos resultaban muy ruidosos. Por ello se eligió utilizar un enfoque en el que se obtuviera para una trayectoria, a través de mínimos cuadrados, el polinomio de grado uno que mejor interpola los puntos. De esta forma si el error se distancia en gran medida de los valores habituales de una trayectoria recta se detecta como un giro. De igual forma se realizaron pruebas aproximando un polinomio de grado dos, comparando la diferencia del error entre el polinomio de grado uno y el de grado dos para mitigar el ruido en las detecciones, pero sin obtener resultados satisfactorios. En ambos casos el ruido en la trayectoria generaba valores de error mayores a los esperados.

Para mitigar el problema de los datos ruidosos (debido al funcionamiento del detector de objetos) se utilizó un filtro de paso bajo (low pass filter) de forma de corregir la trayectoria, obteniendo datos con menor nivel de ruido. Luego del cambio, se obtuvo que utilizar un polinomio de grado uno lograba mejores resultados al evaluar el error, pero fue necesario normalizar el error en relación a la profundidad del vehículo en la imagen y la cantidad de datos disponibles para calcular el error (la cantidad de detecciones utilizadas). La normalización del error se realizó utilizando la siguiente fórmula 3.1.

$$NormalizedError = \frac{\left(\frac{Error}{VehicleHeight}\right)}{\#datapoints} \tag{3.1}$$

Utilizando el error normalizado continuaban apareciendo falsos positivos (de detección de giro) al realizar un análisis visual, por lo que se optó por utilizar la profundidad estimada y no la altura del vehículo. Si el vehículo era pequeño el error incrementaba de igual forma, obteniendo valores incorrectos. En la ecuación 3.2 se muestra la fórmula para calcular la profundidad estimada utilizando el tamaño promedio de los autos. En la fórmula 3.3 se muestra la fórmula para calcular la altura aproximada utilizando la profundidad estimada. En la fórmula 3.4 se presenta la nueva forma de calcular el error utilizando la altura aproximada en lugar de la altura del vehículo.

$$ApproximateDepth(i,j) = \frac{1}{MedianCarHeight(i,j)} \tag{3.2}$$

$$ApproximateHeight(i,j) = \frac{1}{ApproximatedDepth(i,j)} \tag{3.3}$$

$$NormalizedError(i,j) = \frac{\left(\frac{Error}{ApproximatedHeight(i,j)}\right)}{\#datapoints} \tag{3.4}$$

Para calibrar cual es el error mínimo para poder detectar un giro, se realizó un análisis de 10000 datos de errores de la trayectoria de vehículos en relación a la recta que mejor la aproxima, permitiendo de esta forma obtener los casos anómalos (cuando se encuentra girando). La mayor parte de los errores se concentra en cero, disminuyendo su cantidad de ocurrencias de forma drástica al aumentar el valor de error. Aún utilizando un threshold de error que deja a los outliers fuera de la distribución, no fue posible distinguir los casos de giro de los que no lo eran a partir de los datos, siendo la razón principal que muy pocos valores se comportan como outliers en los datos obtenidos. A partir del comportamiento de los datos se eligió un threshold que deja a la izquierda del gráfico (elementos con un valor de error inferior) la mayor cantidad de vehículos. A partir de los datos obtenidos se obtuvo que el mejor threshold para la detección era 13. El nuevo valor de threshold obtuvo mejores resultados. Se agrega a continuación una imagen del histograma obtenido. Se aplicó escala logarítmica a la cantidad de errores por valor de error para facilitar la visualización. Se muestra en el gráfico 3.19 los resultados obtenidos de los errores luego de aplicar escala logarítmica al valor del error.

El método resultaba ruidoso aún con el nuevo threshold y no mostraba buenos resultados, por lo que se optó por utilizar otro enfoque, el cual mejora los resultados obtenidos. El método consta del siguiente procedimiento:

1. Se utilizan los últimos n frames de trayectoria detectados con n fijo.
2. Con los n frames, se seleccionan los primeros k y los últimos k.
3. Para cada conjunto de k frames se toman los vectores de movimiento uno a uno (desplazamiento entre i e i+1 para i entre 1 y k-1), los que luego se promedian.
4. El ángulo de giro final es el ángulo más pequeño entre ambos vectores promedio (uno por cada conjunto de k frames).

Una vez aplicado el método de detección de giro, se detectó que aparecían detecciones aleatorias y por cortos períodos de tiempo que interferían con la detección correcta de giro. Para mitigar las detecciones aleatorias se agregó un contador de detecciones, que solo tiene en cuenta las detecciones que cumplan

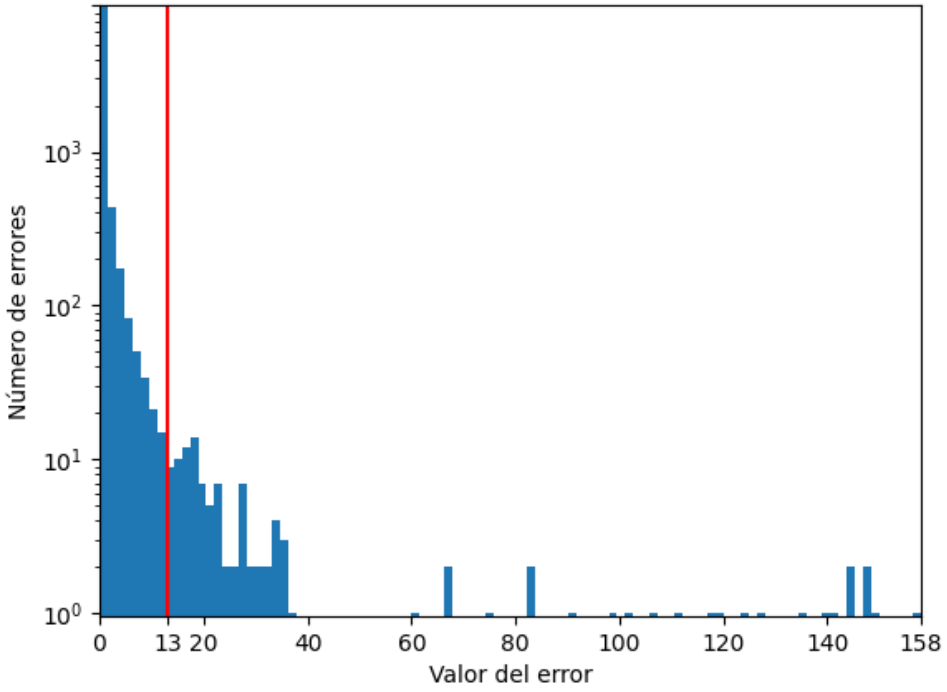

Figura 3.19: Histograma de errores de trayectoria en relación a su aproximación lineal de mínimos cuadrados aplicando escala logarítmica a la cantidad de veces que se alcanza cada error. Se detalla con rojo el valor de threshold utilizado para distinguir casos de giro.

las características buscadas; debiendo detectarse un ángulo en el rango de giro por p frames para que se detecte realmente como un giro. El cambio realizado permitió mitigar los casos en los que se detectaba que un vehículo estaba realizando un giro a causa de falsas detecciones. El cambio añadido provee una detección de giro más robusta.

De esta forma la detección de giro consiste en detectar cuando el ángulo de giro obtenido se encuentra dentro de un rango de ángulos definido por p frames. Los valores de n y k se obtuvieron de forma experimental observando que valores permiten que se tuviera suficiente información de detecciones (n) y que no se utilizara todo el segmento de detecciones para realizar los cálculos; distinguiendo el inicio y el final de forma correcta (k). Para obtener los umbrales de ángulos de giro se realizó una calibración en la que se tomaron datos de un video de cinco minutos con gran flujo de tráfico. En la calibración se tomaron los ángulos de los vehículos con velocidades relativamente altas (mayores

o iguales a 40 km/h). Se muestran a continuación gráficos que muestran los datos obtenidos para velocidad, ángulos 3.20 y ángulos 3.21 respectivamente.

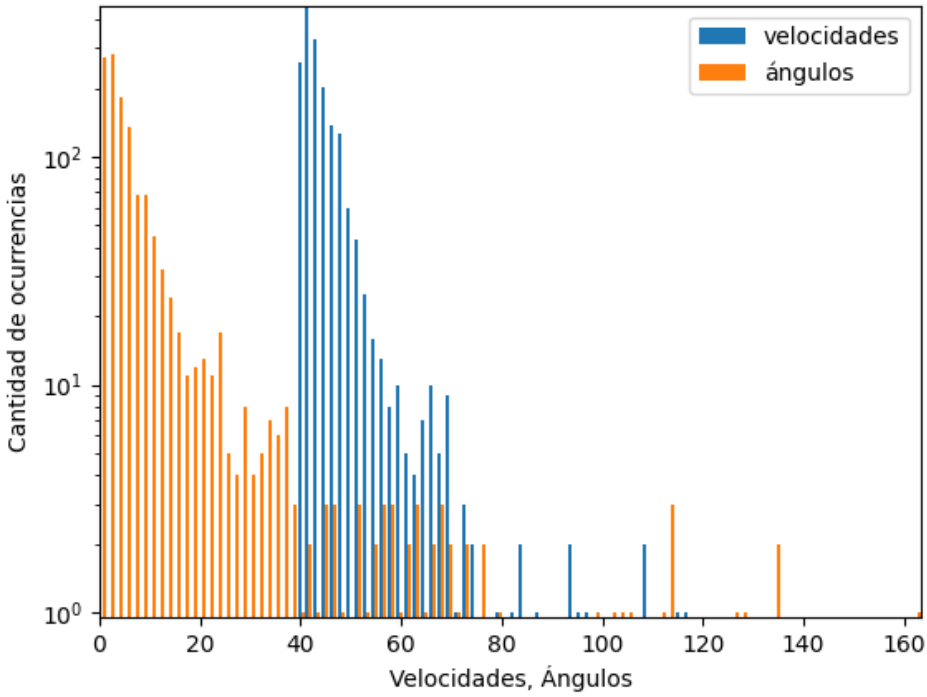

Figura 3.20: Histograma de ángulos obtenidos para velocidades mayores o iguales a 40km/h y las velocidades obtenidas aplicando escala logarítmica a la cantidad de veces que se alcanza cada valor.

A partir de los resultados obtenidos y pruebas realizadas en video se obtuvo que a altas velocidades ángulos bajos suelen ser poco comunes ya que ángulos pequeños cambian en gran medida la dirección de los vehículos. En los histogramas de ángulos mayores a quince grados una baja cantidad de vehículos alcanzan valores superiores al umbral; Lo cual es comprobado con pruebas experimentales. De esta forma quince grados es elegido como threshold mínimo para detección de giro. Luego, para el threshold máximo, se selecciona el valor 70, que se indica en la gráfica como el valor a partir del cual ya no se obtiene un número de datos considerable. Con ese valor se evita considerar valores mayores de ángulo que indiquen que el vehículo se encuentra doblando hacia la derecha por ejemplo. Se consideran valores mayores a 70 grados como anómalos, los cuales son descartados en la detección de giro.

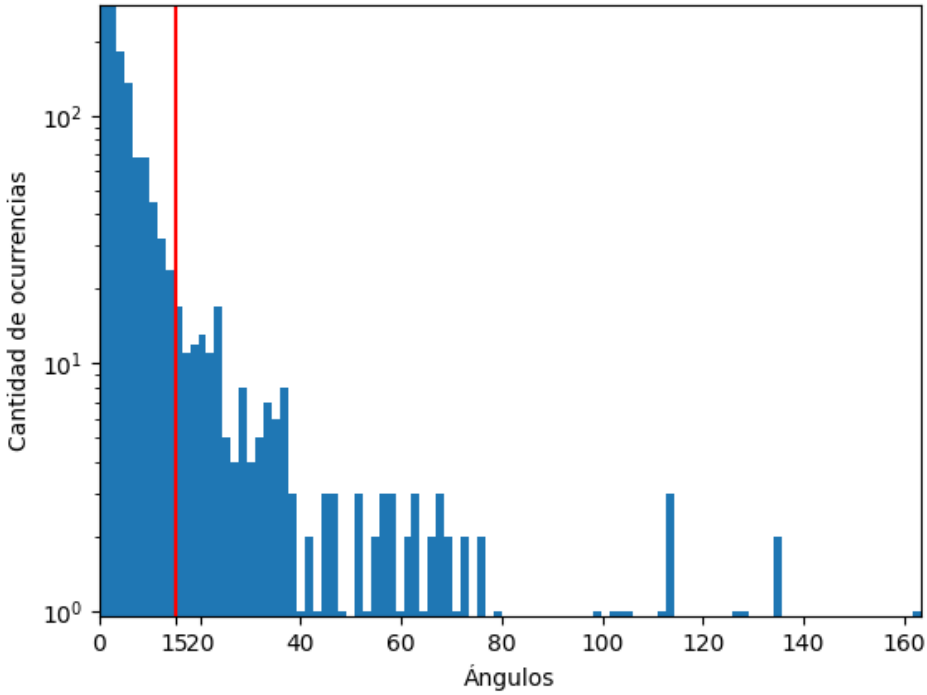

Figura 3.21: Histograma de ángulos obtenidos para velocidades mayores o iguales a 40km/h aplicando escala logarítmica a la cantidad de veces que se alcanza cada valor. Se detalla con rojo el valor de threshold utilizado para distinguir casos de giro.

La cantidad de frames en la que debe ocurrir el evento de giro se obtuvo realizando pruebas en video. Se realizó una búsqueda con el fin de obtener la cantidad de frames que minimiza los falsos positivos maximizando las detecciones de giro real. Al realizar pruebas se comenzó con el valor de cinco, obteniendo que al aumentar o disminuir su valor generaba resultados inferiores en las pruebas realizadas. En base a la información recabada, la condición de giro se define como la variación de ángulo de entre 15 y 70 grados durante cinco frames consecutivos.

3.11.1.5. Velocidad mayor a la permitida

Se observó que al igual que los datos de posición, la velocidad estimada de los vehículos también presentaba niveles altos de ruido (su valor oscilaba de forma no deseada). Se realizaron pruebas utilizando dos enfoques iniciales:

1. Utilizar como velocidad el promedio de la velocidad en los últimos n frames

2. Aplicar filtro de paso bajo y luego realizar un promedio de la velocidad en los últimos n frames.

El primer enfoque presentó mejores resultados, por lo que se decidió continuar utilizando el enfoque. La velocidad promediada se calcula en cada frame, utilizando los últimos n valores. La cantidad de valores utilizados no debe ser muy extensa o la información generada se encontrará desactualizada. Se utilizó un valor de diez fotogramas considerando ambas restricciones. Fue necesario normalizar la velocidad de forma que se asemeje lo más posible a la velocidad real del vehículo. Para la tarea se utilizaron los FPS del video, una constante experimental a determinar y el multiplicador 3.6 para convertir de m/s a km/h. Se obtiene la siguiente expresión 3.5.

$$NormalizedSpeed = VelocityMultiplier \times Speed \times FPS \times 3.6 \tag{3.5}$$

Para obtener la constante experimental VelocityMultiplier se realizó un experimento en el que se obtienen las coordenadas (x,y) de la porción delantera de un auto en la pantalla. Se registra la velocidad en el punto y el número de frame. Luego, cuando la parte trasera del auto supera el punto, se registra de igual forma la velocidad en el punto y el número de frame. Dado que se conoce la longitud aproximada de un auto (cuatro metros), es posible saber cual es la velocidad real con alta certeza. Se promedian las velocidades estimadas en el punto inicial y final y se resuelve un sistema de una variable para obtener el coeficiente experimental buscado. Para el experimento se utilizaron las dimensiones reales de un trayecto de la calle utilizando Google Earth 3.22.

A partir de esta información es posible saber que una vez que un auto cruza la zona indicada, el auto recorrió siete metros. Luego, utilizando la diferencia de frames en ese intervalo de tiempo es posible obtener la velocidad real en m/s. Se utilizan para el vehículo las velocidades estimadas al inicio y al final del trayecto (luego de salir de la zona en la que se recorren los siete metros). Se realiza un promedio entre ambas para comparar con la velocidad real.

Al terminar de procesar un vehículo, se cuenta entonces con la velocidad estimada y la velocidad real. Se recolectaron datos para cada vehículo que atravesó el trayecto y a partir de los valores obtenidos para cada caso (velocidad real y estimada), se calcula el coeficiente que mejor aproxima la velocidad estimada a la real. Para el calculo se promedian de los coeficientes para cada vehículo utilizando la fórmula 3.6, en la que $\#datapoints$ refiere a la cantidad

Figura 3.22: Distancia hasta el cruce de la calle Bulevar Artigas y 21 tomada de Google Earth

de vehículos de los que se obtuvieron datos. La figura 3.23 muestra el histograma generado a partir de los datos obtenidos del experimento de calibración de velocidad. Una vez que se tiene la velocidad normalizada, es posible indicar que si se sobrepasa la velocidad establecida, se considera que el vehículo está a una velocidad peligrosa.

$$VelocityMultiplier = avg\left(\frac{RealSpeed(i)}{EstimatedSpeed(i)}\right) \text{ for i in 1..\#datapoints} \quad (3.6)$$

Los datos de la velocidad aproximada no resultaron ser correctos en varios casos, dado que no se contaba con información de la profundidad del vehículo en algunos puntos. Para resolver el problema se agrega interpolación de los datos de profundidad cada 30 frames. Se optó por que la interpolación se realice cada 30 frames de forma que se obtengan suficientes datos nuevos y no calcularla en todos los frames dado el costo computacional asociado. A continuación se presenta en la figura 3.24 una imagen ilustrando la interpolación de profundidad aplicada.

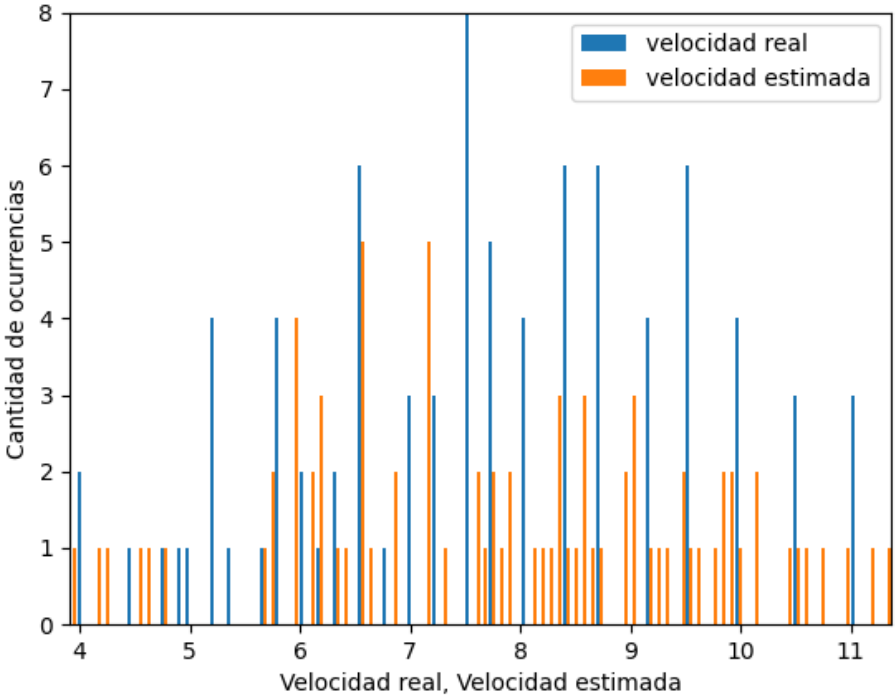

Figura 3.23: Histograma de las velocidades estimadas y reales obtenidas luego de remover los outliers para velocidades reales y estimadas en m/s. Se utiliza la sección de la calle referenciada en la figura 3.22.

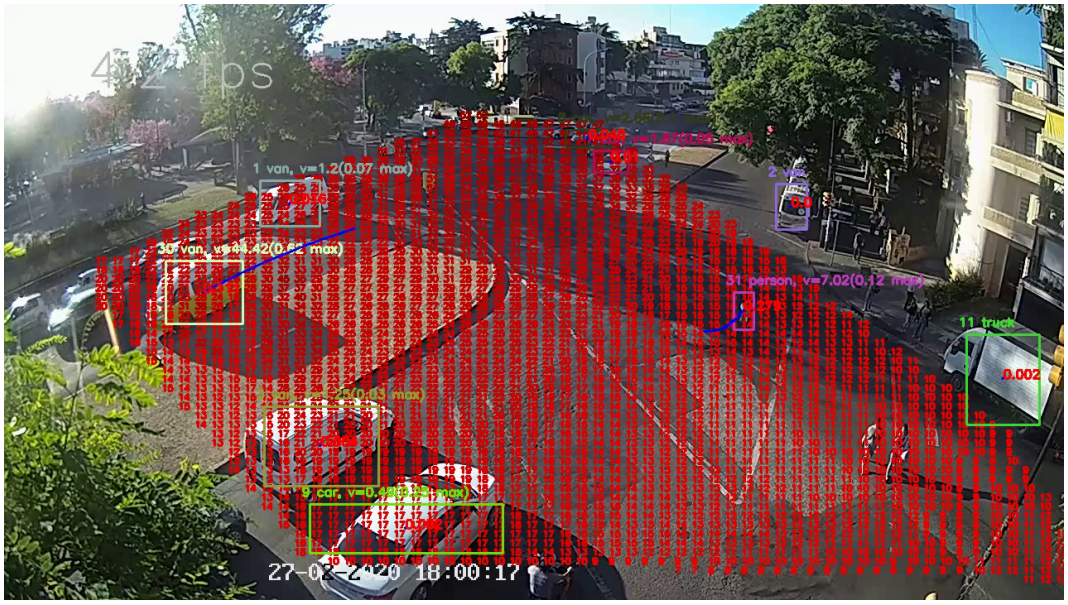

Figura 3.24: Interpolación de la profundidad

Al realizar pruebas adicionales se obtuvo que el enfoque no realizaba una estimación de velocidad certero en la práctica (en ciertas zonas del video los valores de velocidad eran mayores a los reales). Como posible primer solución se optó por utilizar dos coeficientes de normalización (uno aplicado a la parte baja y otro a la parte alta de la pantalla) que permitieran disminuir el efecto de la problemática (la diferencia ocurría principalmente en la parte más alta de la pantalla). Se realizó la calibración en dos puntos adicionales de la pantalla para un video de cinco minutos para obtener los coeficientes que mejor normalizan la velocidad estimada a la real. Se obtuvo que los coeficientes obtenidos eran incompatibles, invalidando la idea de definir coeficientes por zona de la pantalla e interpolar utilizándolos.

3.11.1.6. Mapeo de coordenadas en la pantalla a coordenadas geográficas

Dados los problemas enfrentados con el enfoque de grilla para la estimación de la profundidad, se optó por realizar una investigación sobre normalización de velocidades. Fue posible encontrar información de un proyecto ya realizado Blake (2019) en el que se presentó un método para convertir coordenadas de una cámara a coordenadas geográficas utilizando cuatro puntos de referencia (conversión de puntos de una cámara a coordenadas geográficas). Se integró el código presentado al repositorio del proyecto y se realizaron cambios al código para ser utilizado en Uruguay. A partir de los cambios añadidos se pudo obtener una mejor estimación de la velocidad. Se realizó una prueba en la que se toma la velocidad real y la estimada, logrando una aproximación mucho más acertada. El cambio no solo mejoró el tiempo de ejecución del algoritmo sino que también los resultados de la estimación de la velocidad, aunque requirió que el esfuerzo de calibración inicial requerido para poder utilizar una cámara aumente. Se utilizó la herramienta presentada en la sección 3.10.2 para la calibración de los parámetros de la conversión.

Para obtener la transformación, se mencionaba en el proyecto que fue utilizado como base la utilización de un perspective transform o transformación de perspectiva. Lo que realiza una transformación de perspectiva es una operación matricial que transforma un conjunto de puntos de un plano 2D a otro. La operación genera una matriz de dimensiones 3×3 que al multiplicarse por las coordenadas de origen obtiene coordenadas correspondientes en el plano

destino. En la figura 3.25 se presenta una imagen que representa la operación de aplicar una transformación de perspectiva.

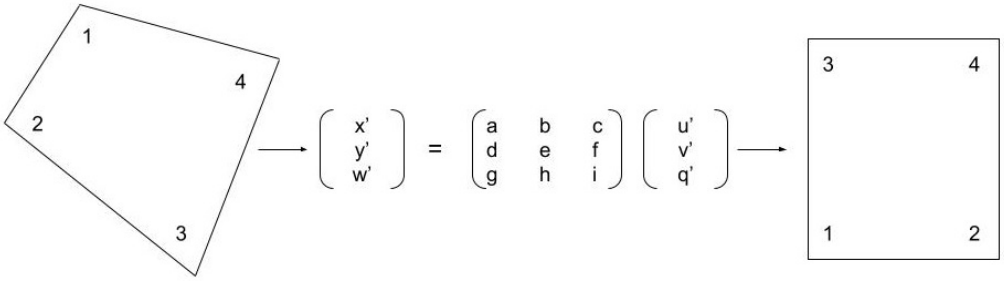

Figura 3.25: Imagen representativa de una transformación de perspectiva

De esta forma se deben realizar dos etapas a partir de una coordenada x,y de la cámara para normalizar las posiciones:

1. Se utiliza la transformación de perspectiva para convertir píxeles a latitud longitud del sistema de referencia con el que se realizó la conversión (4326 de Google Maps).
2. A partir de la latitud longitud se utiliza una transformación que transforma coordenadas de un sistema geográfico a otro para pasar del 4326 al 32721 (Sistema cartesiano 2D de la zona de Uruguay). El sistema es utilizado en lugar de uno geoide para poder utilizar las coordenadas geográficas resultantes en un plano cartesiano de dos coordenadas.

Dado que OpenCV utiliza representación punto flotante 32 bits y el programa utilizado maneja 64 bits para su representación punto flotante, fue necesario, para evitar pérdida de precision que puede resultar en una cancelación catastrófica durante la transformación, restar el punto mínimo de calibración de la cámara en coordenadas geográficas a cada punto que es procesado por el algoritmo.

Una vez que cuenta con la posición normalizada, se utilizan las últimas cinco posiciones para cada vehículo en el plano cartesiano (cinco para que el método sea más robusto y la información de velocidad no sea desactualizada). De esta forma se obtiene la norma del desplazamiento obtenido y se divide sobre la cantidad de frames transcurridos. Luego se multiplica por los FPS para realizar la conversión a metros por segundo y se multiplica por 3.6 para pasar a km/h como se muestra en la ecuación 3.7. Realizar el mapeo utilizando

coordenadas geográficas obtuvo los mejores resultados, por lo que fue utilizado en la solución final. No obstante, en casos en los que el terreno no es plano (colinas por ejemplo), el método tiene un nivel de error mayor ya que aproxima la transformación utilizando un plano.

$$speed = \left(\frac{\sqrt{(dx \times dx + dy \times dy)}}{\#frames} \right) \times fps \times 3.6 \tag{3.7}$$

3.11.1.7. Identificación de carriles, semáforos, cruces

Fue necesario agregar la identificación de carriles, semáforos y cruces para los algoritmos de detección de patrones de tránsito. Se realizó una herramienta que permite generar polígonos para una imagen de una cámara, anotando de forma manual los carriles, semáforos y cruces de una imagen como se mencionó anteriormente. Se muestra a continuación una imagen de los carriles, semáforos y cruces que fueron etiquetados utilizando la herramienta de etiquetado generada 3.26.

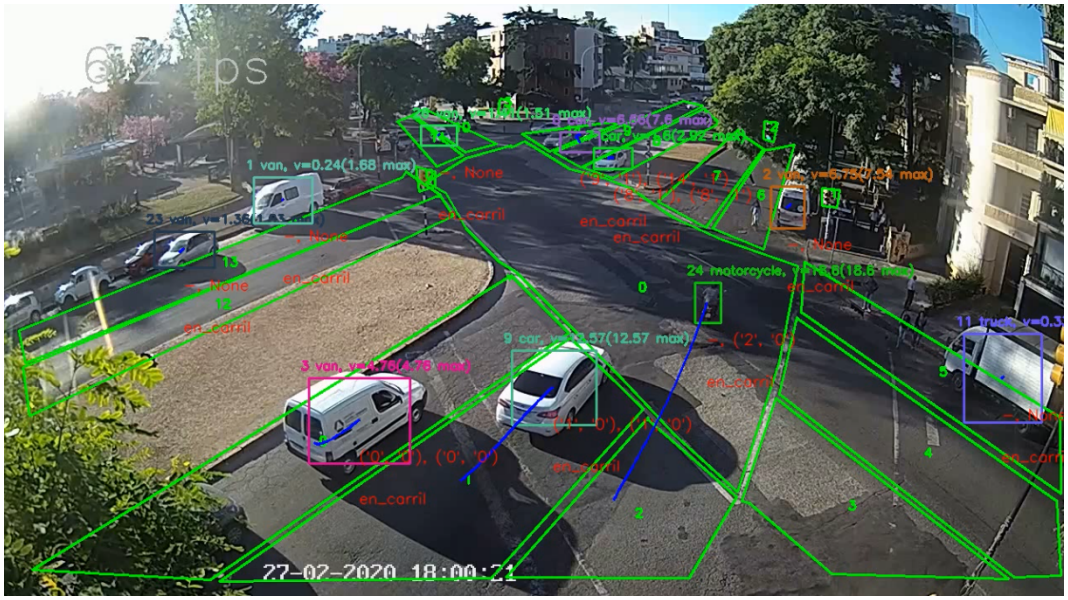

Figura 3.26: Carriles, semáforos y cruces para el video de Bv Artigas y 21 de Setiembre

Particularmente para el patrón de movimiento brusco resultaba de interés detectar los casos en los que se detectaba movimiento brusco y el vehículo luego doblaba para descartar esos casos. Se presenta la idea de grupo carril, el cual para el patrón de movimiento brusco se entiende como un identificador

que los carriles de la misma vía tienen. En el ejemplo los carriles 0,1,2,6 y 7 pertenecen al mismo grupo carril. Se agrega información en el registro de cada vehículo que registra cual fue el último grupo carril válido visitado y si transitó o no por un cruce previamente. A partir del último carril válido visitado y si transitó o no por un cruce, se generan tres posibles estados del vehículo en relación a los carriles. A partir de estos tres estados se genera el diagrama de flujo presentado a continuación 3.27. Los estados utilizados para el diagrama de flujo son:

- NO_ROAD_CHANGE (no hubo cambio de grupo carril)
- MOVED_TO_DIFFERENT_ROAD (el vehículo se movió a un grupo carril diferente)
- MOVED_TO_SAME_ROAD_AFTER_INTERSECTION (indica que un vehículo pasó por una intersección y volvió al mismo grupo carril)

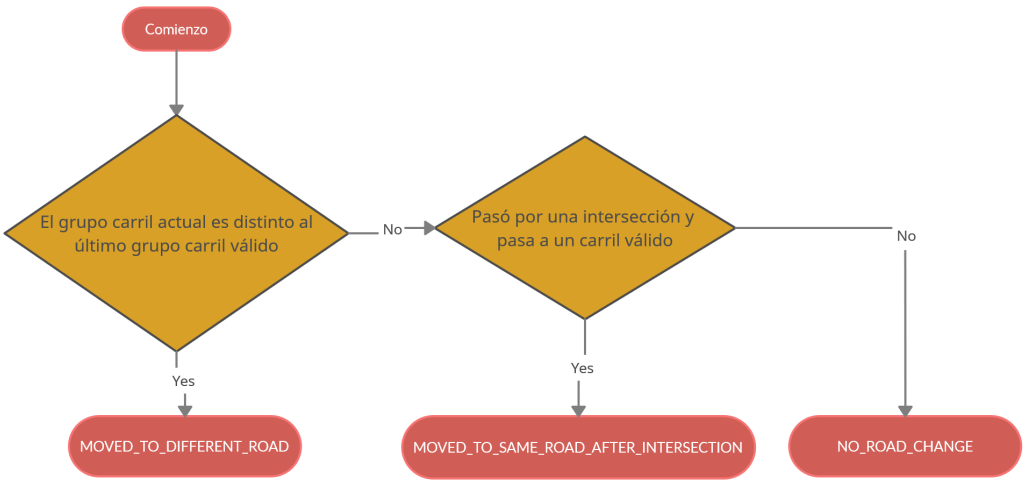

Figura 3.27: Diagrama de flujo de detección de estado de carril de vehículo

3.11.1.8. Clasificación de movimiento brusco

Una vez que se posee el estado del vehículo en los grupos carriles, información de velocidad de forma precisa y se tiene información de giro, se utiliza esta información para detectar el patrón de movimiento brusco. Se procede a utilizar un enfoque de máquina de estados para poder obtener cuando se está realizando un movimiento brusco. La información del cambio de carril se utiliza para diferenciar cuando se está doblando hacia izquierda o derecha para

evitar registrar esos casos como movimiento brusco. Se agrega a continuación el diagrama de máquina de estados utilizada en la clasificación 3.28.

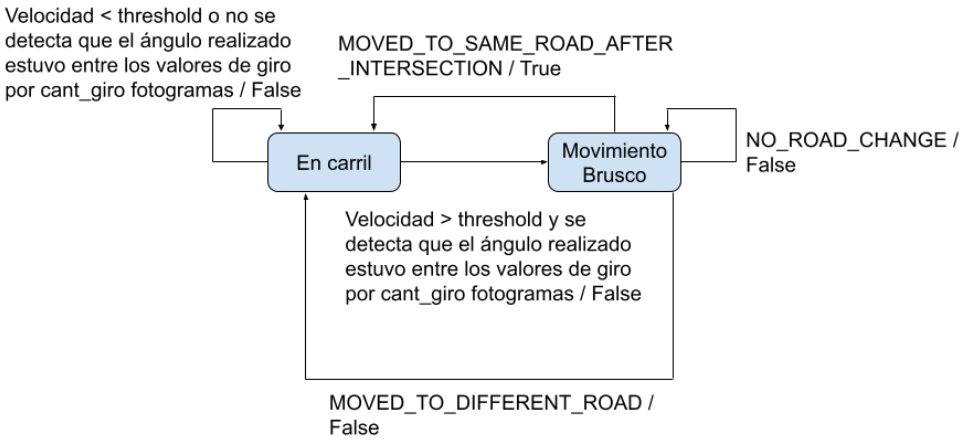

Figura 3.28: Máquina de estados encargada de la detección de movimiento brusco. *cant_giro* refiere a la cantidad de frames en la que debe detectarse el patrón como se menciona en la sección 3.11.1.4.

3.11.2. Cruce en luz roja

En esta sección se presenta el procedimiento realizado para la implementación del detector de cruce en luz roja en el tránsito. Un cruce en luz roja de un vehículo en el marco del proyecto de grado representa la situación en la que un vehículo avanza cuando hay luz roja en el semáforo que indica que debe detenerse.

3.11.2.1. Detección de estado del semáforo mediante OpenCV

Se detecta el estado del semáforo mediante una máscara de color en el espacio HSV, donde el espacio BGR se transforma a tuplas de tono (H), saturación (S) y valor (V) en rangos 0-179, 0-255 y 0-255 respectivamente. La conversión es de baja complejidad al ser operaciones lineales sobre los componentes RGB. El componente V corresponde al máximo de los tres componentes RGB. El componente S se calcula como la diferencia entre el máximo y el mínimo de los componentes RGB dividido entre el máximo. El cálculo del componente H se divide en tres segmentos según el componente RGB máximo como se observa en la tabla 3.4, el rango dentro del segmento depende de la relación entre los

otros dos componentes RGB. En la figura 3.29 se muestran los colores obtenidos al variar el componente de tono en HSV manteniendo la saturación y el valor en 255.

$$
\begin{aligned}
V &\leftarrow \quad max(R, G, B) \\
S &\leftarrow \quad \begin{cases} 255 \cdot \dfrac{V - min(R, G, B)}{V} & \text{si } V \neq 0 \\ 0 & \text{si } V = 0 \end{cases}
\end{aligned}
\tag{3.8}
$$

$max(R, G, B)$	fórmula	caso	rango
R	$30(G - B)/(V - M)$	$B \geq G$ $G \geq B$	150-179 0-30
G	$60 + 30(B - R)/(V - M)$	$R \geq B$ $B \geq R$	30-60 60-90
B	$120 + 30(R - G)/(V - M)$	$G \geq R$ $R \geq G$	90-120 120-150
R = G = B	0		

Tabla 3.4: Fórmula y rangos posibles del componente de tono H, donde $V = max(R, G, B)$ y $M = min(R, G, B)$.

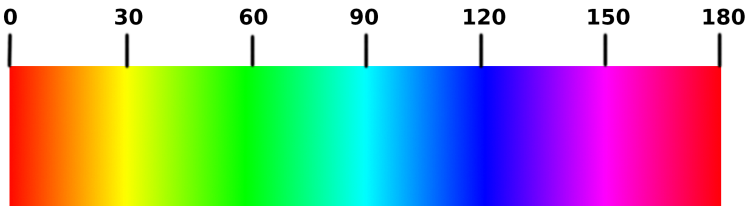

Figura 3.29: Colores obtenidos al variar el componente de tono manteniendo saturación y valor en 255.

Para detectar el estado del semáforo se utilizan los rangos descriptos en la tabla 3.5. El rango inferior de la saturación se calcula en función de la saturación promedio del área del semáforo para adaptarse mejor a las condiciones de iluminación y a las diferencias entre cámaras.

Color	Tono	Valor
Rojo	158 - 14	128 - 255
Verde	60 - 95	180 - 255
Amarillo	15 - 32	240 - 255

Tabla 3.5: Rangos HSV para los colores de luz de semáforos. Tono en rango 0-179, Valor en rango 0 - 255.

Se restringe para cada semáforo las máscaras de color al cuadro delimitador del semáforo. Para las regiones detectadas de cada color específico se calcula el cociente del área de la región sobre el área de una elipse generada al aplicar la función *fitElipse* de OpenCV sobre la región. Si bien la función no garantiza que la elipse generada contenga completamente el blob, la gran mayoría de los píxeles quedan contenidos. Se utilizó un umbral de 0.6 para este cociente (en base a pruebas manuales), un valor inferior indica que el blob de color no se ajusta a una elipse por lo que se trata de un candidato falso con un color similar. El aspect ratio máximo de la elipse considerado es 2.5, este último valor es mayor al aspect ratio real a efectos de mayor tolerancia a variaciones de iluminación, pero es suficientemente bajo para reducir falsas detecciones que puedan deberse a objetos de color similar a las luces del semáforo dentro del cuadro delimitador del mismo.

Para la detección del color amarillo debido a ser contiguo al rojo para algunas condiciones de iluminación se agregó una condición adicional, que consiste que el centro de la detección debe estar dentro del tercio central del área del semáforo.

3.11.2.2. Entrenamiento de red neuronal para detectar estado de semáforo

Se encontró que los resultados no eran satisfactorios sobre semáforos con area reducida o en las condiciones de iluminación desfavorables antes citadas, por lo tanto se exploró utilizar una red neuronal para mejorar los resultados. Se priorizó que esta red sea suficientemente ligera para no tener efectos adversos en la velocidad de procesamiento del algoritmo.

Para el entrenamiento se utilizó el conjunto de entrenamiento del Bosch Small Traffic Lights Dataset (5093 fotogramas en color) introducido en Behrendt y Novak (2017), de donde se extraen 10756 bounding boxes de los semáforos. Contiene categorías 'Red', 'Yellow', 'Green' así como una categoría 'Off'

cuando contiene un semáforo pero el color no es visible o directamente el semáforo está apagado. No se utilizó el conjunto de datos DriveU Traffic Light Dataset descripto en Fregin *et al.* (2018), el cuál es de mayor tamaño, por dificultades en el procesamiento de imágenes ya que las imágenes no se encuentran demosaicadas, paso necesario para obtener imágenes de color, lo que supone un tiempo de implementación adicional para procesar dichas imágenes.

De los fotogramas sin semáforos se extrajo el mismo número (10756) de ejemplos negativos, de 24 píxeles de ancho por 48 de alto, a partir de fotogramas que no contengan ejemplos positivos, potencialmente obteniendo varios ejemplos negativos de la misma imagen. Por cada imagen negativa se corrió el detector de puntos notables FAST de OpenCV con parámetro threshold=20, el cuál detecta principalmente puntos en los bordes. Para obtener cada ejemplo negativo se eligió un punto al azar de la imagen, con una probabilidad del 25 % se toma directamente el área centrada en ese punto, el 75 % restante se elige como centro el punto notable más cercano. Adicionalmente se generan 2000 imágenes negativas con ruido uniforme. A las imágenes negativas se les asignó una categoría 'Negative' para diferenciarlas de la categoría 'Off'. El objetivo de agregar estas imágenes negativas es evitar overfitting y la sobresimplificación del modelo (por ejemplo, decir que el semáforo esté en verde debido a que hay un árbol en el fondo).

Se utilizó como arquitectura la descripta por Plotka (2018), la que consiste en unidades de dos capas convolucionales con activación RELU, seguidas de una capa de reducción con Max pooling; seguida de una capa de regularización mediante Dropout, repetidas dos o tres veces para alcanzar cuatro o seis capas, finalizando con una capa fuertemente conexa (con 128-512 neuronas) antes de la capa de clasificación. Adicionalmente se realizaron pruebas utilizando el modelo descrito por Haque *et al.* (2021), dirigido a la clasificación de señales de tráfico, pero los resultados fueron inferiores (con la salvedad que no puede garantizarse que la reproducción haya sido fidedigna pues no se cuenta con el código asociado). Para entrenar los modelos se utilizó data augmentation para aplicar cambio de brillo, contraste, saturación, corrimiento y cizallamiento (para simular el cambio de ángulo).

3.11.2.3. Detección de cruce en roja para vehículos

En la figura 3.30 se presenta el diagrama de flujo para el algoritmo de detección de cruce en roja para un vehículo. Solo se consideran los objetos que estén dentro de un cruce y se encuentren en movimiento. El procedimiento dependerá de la clase del objeto:

- Si el objeto siendo procesado se trata de un vehículo (definido como todos los objetos que no sean personas), para considerarlo cruce en luz roja debe haber empezado a atravesar el cruce cuando el semáforo asociado al carril donde se donde se desplazaba se encontrara en rojo. La duración del evento solo comprende el tiempo durante el cuál el semáforo continúa en rojo, por lo tanto cesará si cambia a verde durante el cruce.
- Si es una persona se busca el carril más cercano y se aplica el algoritmo de cruces de personas como si estuviera sobre ese carril.

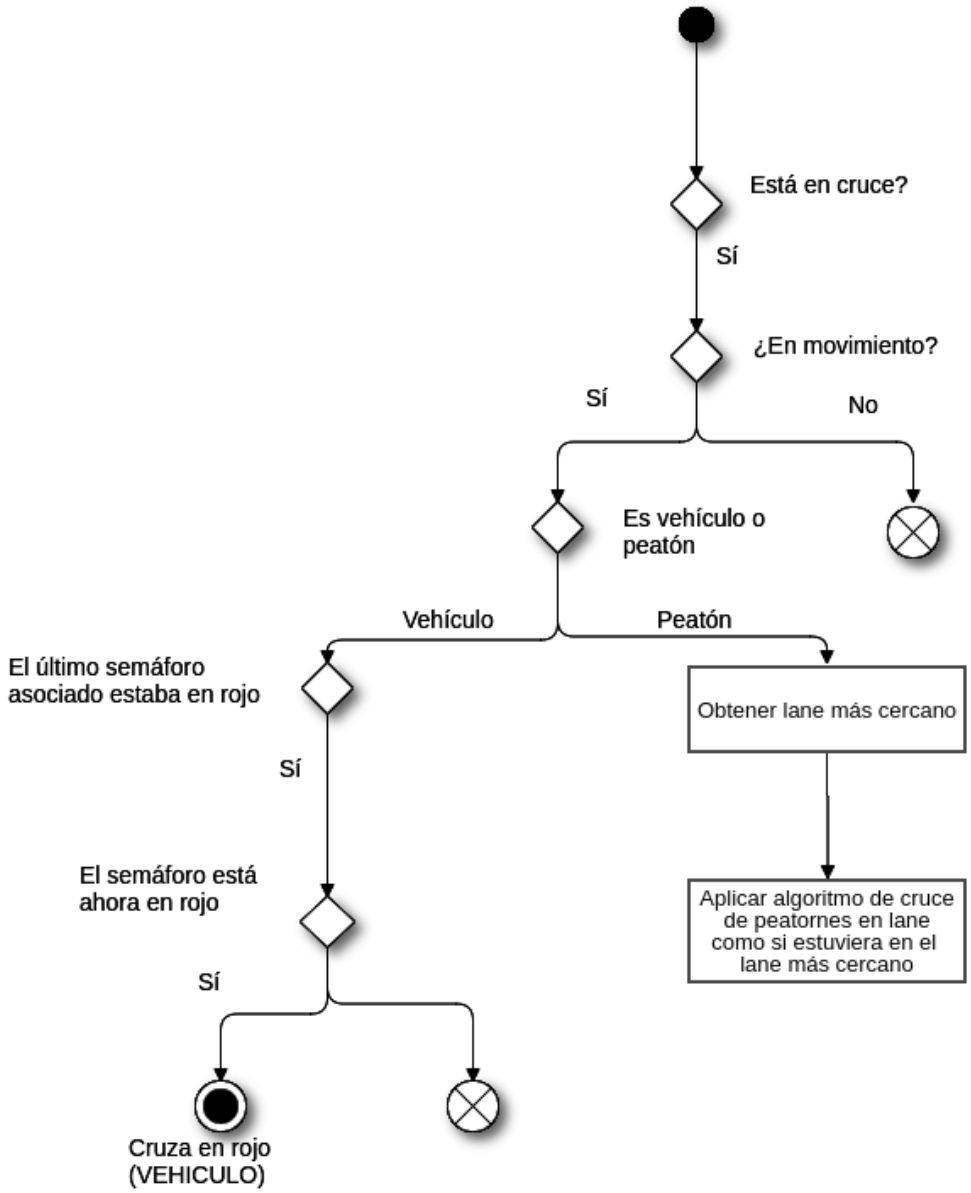

Figura 3.30: Diagrama de flujo de detección de cruce en luz roja para vehículos y personas

3.11.2.4. Detección de cruce en roja para personas

En la figura 3.31 se muestra diagrama de flujo para el algoritmo de detección de cruce en roja para personas junto con la descripción de su funcionamiento.

La detección de personas que cruzan con luz roja cuenta con los siguientes pasos:

- Si la persona se encuentra dentro de un cruce se busca el carril más cercano y se realiza el procesamiento como si la persona estuviera en ese carril. El procedimiento mencionado reemplaza a la versión anterior donde se intentaba determinar el semáforo destino como el más cercano en ángulo con respecto al vector de movimiento y con respecto a la distancia a la persona, este procedimiento tiende a causar selección errónea del semáforo.
- Se calcula el ángulo entre el vector de movimiento de la persona (en base al movimiento en el pasado reciente) y la dirección inferida del carril (en función de los ejes mayores de los otros carriles de la calle).
- Si el ángulo obtenido supera los 40 grados se asume que la persona está cruzando la calle. En caso contrario no se considera un cruce (por ejemplo está caminando en la acera pero la detección ubica a la persona en la calle).
- En caso de que el carril tenga algún semáforo asignado como *perpendicular* (es decir en sus costados), se considera que hay cruce en rojo cuando este semáforo esté en rojo, pues es el que gobierna el cruce de la calle al mirar directamente a la persona.
- Si no se pudo resolver en el item anterior, en caso que el carril tenga algún semáforo asignado en dirección directa se considera que hay cruce en rojo cuando el semáforo esté en verde, lo que implica que el semáforo perpendicular debe estar en rojo, pues los semáforos en ambas direcciones no pueden estar permitiendo el paso. Por una cuestión de confiabilidad de la detección se decidió no considerar los casos donde el semáforo directo esté en amarillo, a efectos de reducir el número de detecciones falsas.
- En caso de existir varios semáforos candidatos se pondera la distancia de la persona al semáforo con la confiabilidad del estado del semáforo.

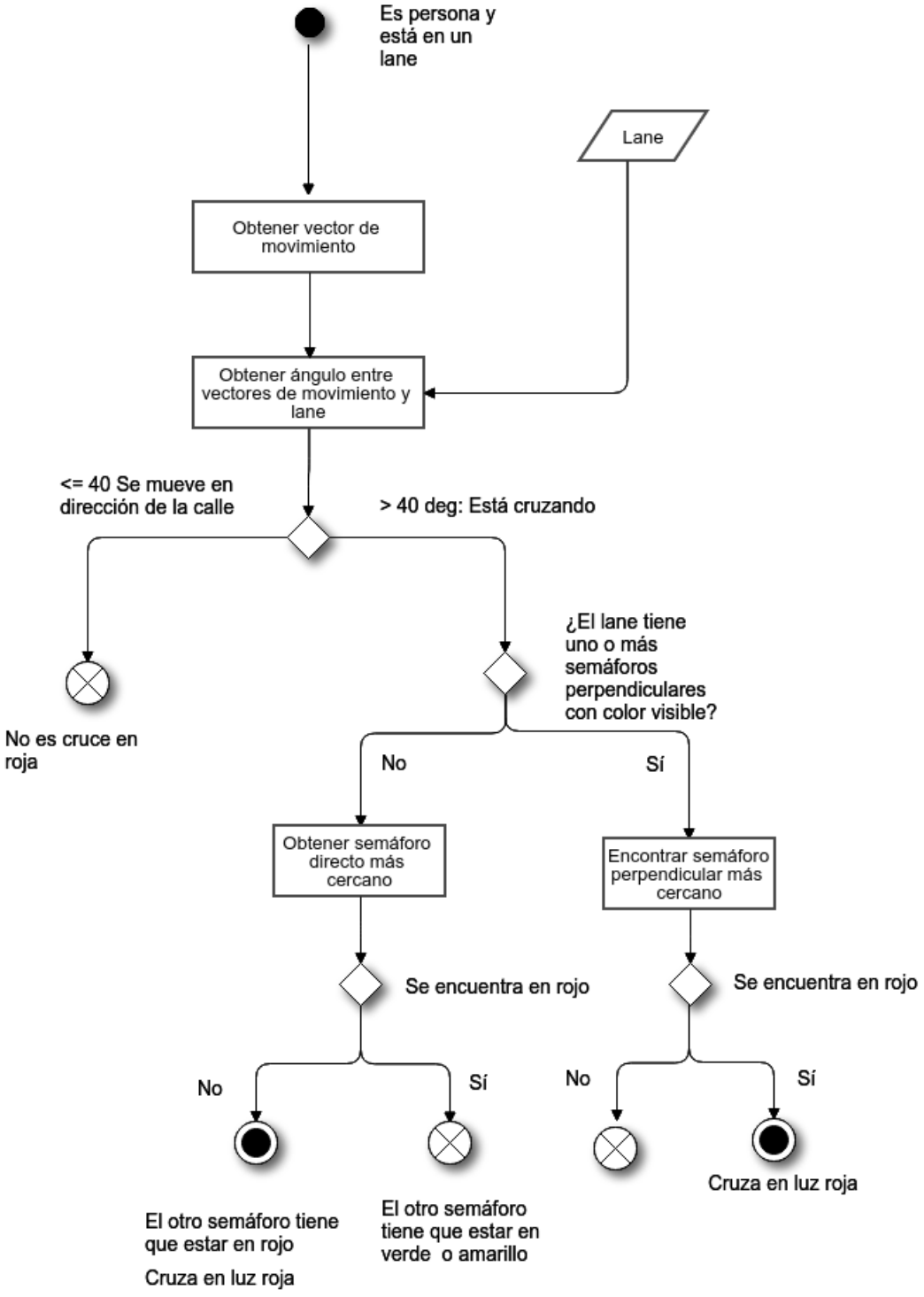

Figura 3.31: Diagrama de flujo de detección de cruce en luz roja para personas

3.11.2.5. Inferencia de la dirección de calle

Se detalla a continuación el procedimiento que realiza el algoritmo de detección de dirección de calle. Conocer la dirección de la calle permite determinar cuando una persona está cruzando la calle en vez de moverse de forma paralela a la calle (ya sea en la calle misma o en la vereda adyacente).

1. Por cada carril se obtiene el rectángulo contenedor mínimo orientado (OBB).
2. Por cada OBB se obtiene el eje mayor y menor.
3. Se ordenan los carriles de acuerdo al eje más largo de forma descendente. Se considera como vector inicial de la calle el del primer carril en el orden mencionado, pues se asume que el carril más grande es representativo de la dirección de la calle.
4. En los demás carriles se considera el eje mayor y el menor, en ambos sentidos. Se elije el eje y sentido que maximiza el producto interno entre el eje y el vector actual de la calle (el producto interno se maximiza cuando dos vectores tienen misma dirección y sentido, por otro lado será nulo cuando sean ortogonales y negativo cuando tengan sentidos opuestos, minimizándose cuando compartan dirección pero no sentido). Luego se actualiza el vector actual sumando el eje elegido.
5. Se utilizan los ejes elegidos en el punto anterior para determinar ángulos de cruce.

3.11.3. Estacionamiento en doble fila

En esta sección se presenta el procedimiento realizado para la implementación del detector de estacionamiento en doble fila en el tránsito. Un estacionamiento en doble fila de un vehículo en el marco del proyecto de grado representa la situación en la que un vehículo estaciona en un carril adyacente al carril de estacionamiento, bloqueando el flujo de tráfico.

3.11.3.1. Consideraciones previas

Para el estudio de este evento se definen "grupos de carriles"que son conjuntos de carriles de una calle que poseen el mismo sentido de circulación, estos carriles no atraviesan cruces de calles. Los carriles se dibujan con la herramienta de dibujar carriles y se exportan en un archivo .json. En la figura

3.32 se puede ver una imagen de los carriles etiquetados para una captura de la cámara de 21 de Setiembre y Bulevar Artigas.

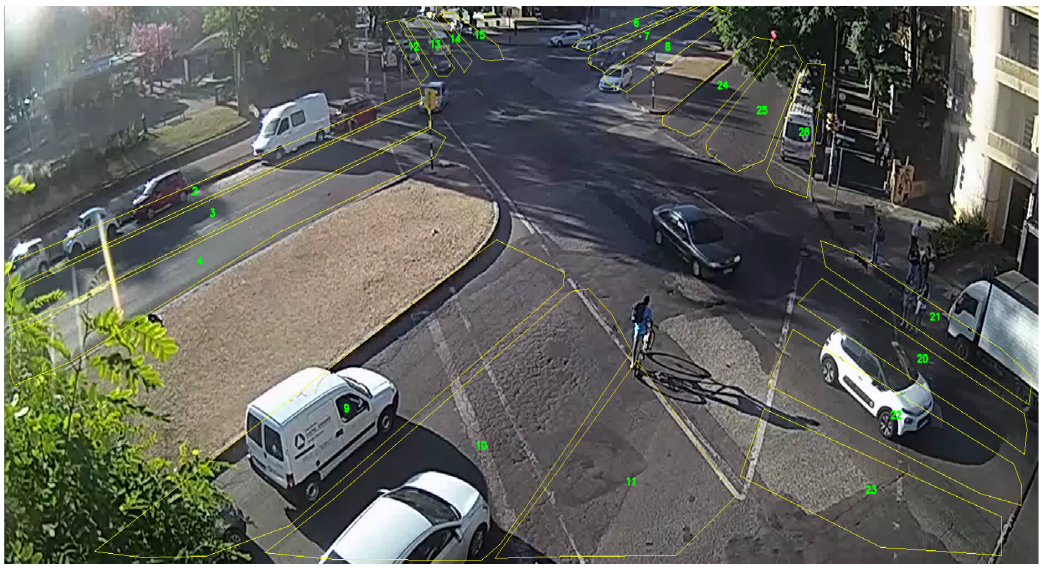

Figura 3.32: Ejemplo de dibujo de carriles para una captura de la cámara de 21 de Setiembre y Bulevar Artigas.

Por otro lado se consideran los carriles externos de cada grupo carril, los cuales son los carriles que limitan con una vereda o con un cantero. Los carriles externos son determinados por el programa automáticamente.

3.11.3.2. Definición del evento

El evento estacionamiento en doble fila se define como el evento en el que un vehículo que se detiene durante un período considerable de tiempo en un carril que es adyacente a un carril externo. El intervalo de tiempo para la detección del patrón es un parámetro determinado por el operador del sistema, el parámetro puede depender de las características de la calle, como puede ser la cantidad de carriles que tiene, el flujo de tránsito, la hora del día, etc.

La lista completa de parámetros del evento es:

- PERIODO_PATRON : Tiempo en segundos durante el cual un vehículo debe estar estacionado en doble fila para considerarlo como una ocurrencia del evento. Este parámetro es definido por el operador del sistema.
- INTERVALO_VEL_INSTANTANEA : Es el intervalo de tiempo en segundos usado para calcular la velocidad instantánea.

- VEL_MINIMA : La velocidad mínima en m/s para considerar que el vehículo está detenido.
- PUNTOS_REPRESENTATIVOS : Conjunto de puntos representativos del bounding box de un vehículo.
- UMBRAL_DETECCIONES_ESTACIONADO: Umbral mínimo que debe alcanzar la proporción de detecciones del vehículo en un carril externo con respecto al total de detecciones para determinar que el vehículo se encuentra estacionado de forma correcta, es decir, en un carril externo.
- UMBRAL_SEMAFORO_VERDE : Umbral mínimo que debe alcanzar la proporción de detecciones del semáforo en color verde sobre el total de detecciones del color del semáforo en un segundo determinado para considerar que el semáforo estuvo en verde ese segundo.
- MINIMO_DETECCIONES : Número mínimo de detecciones del evento, para determinar que el vehículo estuvo estacionado en doble fila efectivamente.

3.11.3.3. Detección de estacionamiento en doble fila

Para la detección del evento se sigue un proceso que se puede observar en el diagrama de flujo en la figura 3.33. Las etapas son:

1. Determinar si el objeto detectado es una persona.
2. Determinar si el tamaño es anómalo.
3. Determinar si está en un carril que es doble fila.
4. Determinar si la cantidad de detecciones es mayor a un mínimo.
5. Determinar si la velocidad promedio es menor a la velocidad mínima para considerar que el vehículo estuvo detenido.
6. Determinar si el vehículo está estacionado.
7. Determinar si estuvo la mayor parte del tiempo con el semáforo que lo habilita a avanzar en verde.
8. Determinar si la cantidad de detecciones en las que estuvo estacionado en doble fila es mayor a un mínimo.

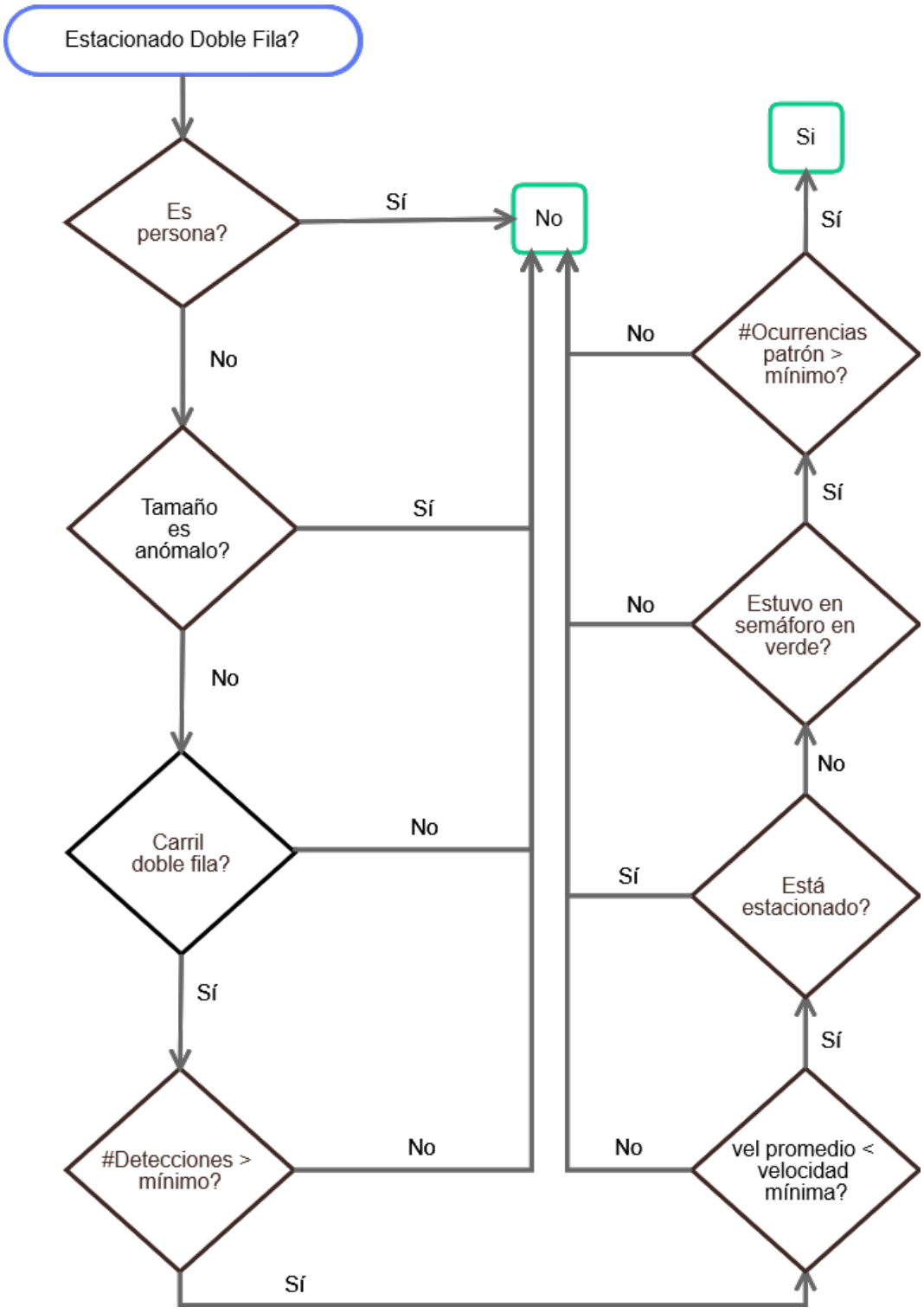

Figura 3.33: Diagrama de flujo de la detección del evento estacionamiento en doble fila.

Para determinar si el objeto detectado es una persona se consulta la clase del objeto detectado. En caso afirmativo se retorna que no estuvo estacionado en doble fila, en caso negativo se continúa con la evaluación de tamaño anómalo.

Para determinar si el tamaño del vehículo es anómalo se comprueba si el área del bounding box del vehículo es menor al percentil 5 de las áreas de los bounding box de vehículos de su clase. El histograma para determinar el percentil se determina previo a la ejecución del test. En las figuras 3.34 y 3.35 se puede observar los histogramas para las clases "car" y "bus", los histogramas de las clases restantes se presentan en el anexo 1. En caso que el tamaño sea menor se considera un error del detector y por lo tanto se retorna que no está estacionado en doble fila. En caso que el tamaño no sea menor se prosigue a comprobar si el vehículo está en un carril doble fila.

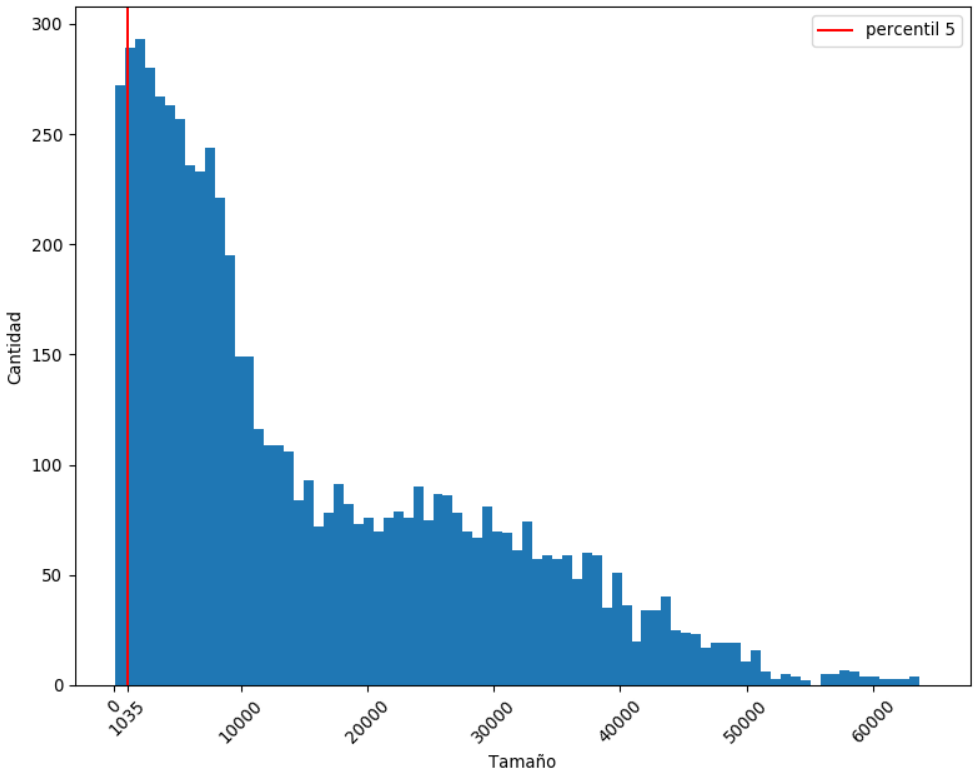

Figura 3.34: Histograma de tamaños de vehículos para la clase 'car'. Se indica el tamaño del percentil 5 de los tamaños, el cual se usa para determinar tamaños anómalamente pequeños.

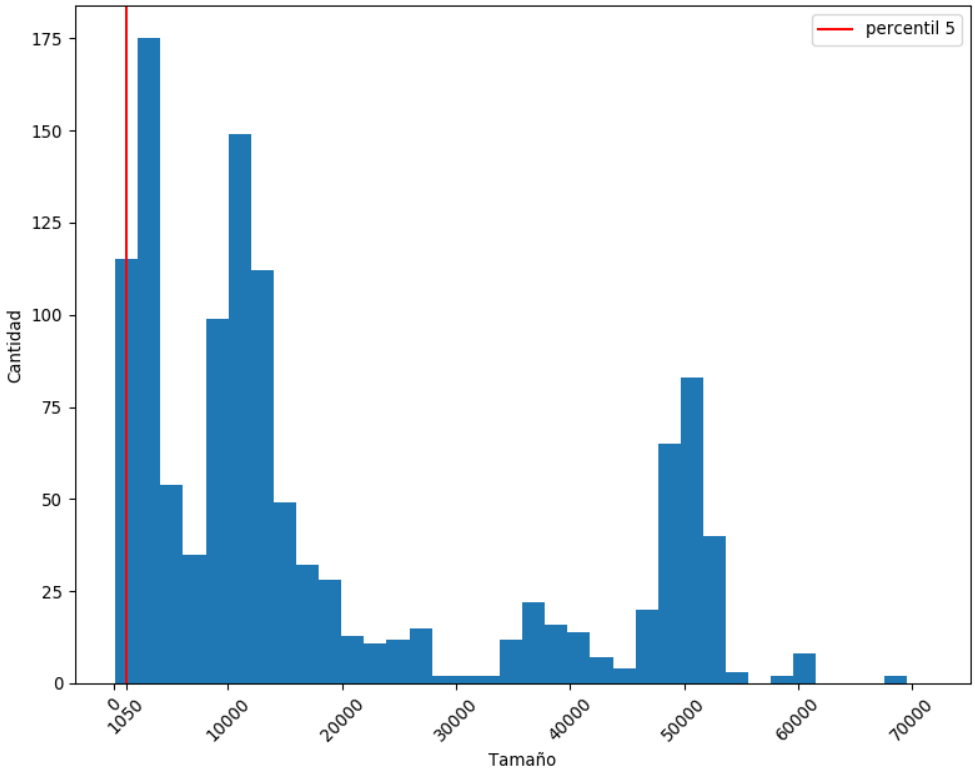

Figura 3.35: Histograma de tamaños de vehículos para la clase 'bus'. Se indica el tamaño del percentil 5 de los tamaños, el cual se usa para determinar tamaños anómalamente pequeños.

Para determinar si el vehículo está en un carril doble fila se determinan unos puntos representativos dentro de su bounding box. Estos puntos están dados de forma genérica por las coordenadas (x_{medio}, y) , donde x_{medio} es el punto medio en el eje horizontal del bounding box, e $y \in \{0.1, 0.25\}$ es un punto en el eje vertical, situado a 0.1 ó 0.25 de distancia respecto al borde inferior del bounding box. Esta distancia es relativa al tamaño del bounding box, por ejemplo distancia 1 es el borde superior del bounding box, distancia 0.5 es el punto medio en el eje vertical, etc. Estos puntos se eligieron dado que en diferentes perspectivas el borde inferior del vehículo suele intersectar el carril en el que el vehículo se encuentra, se eligieron dos puntos de forma de disminuir el ruido en la detección. En las figuras 3.36 y 3.37 se pueden observar ejemplos de ambos puntos. En caso que ninguno de estos puntos intersecte un carril que sea adyacente a un carril externo, se devuelve que el vehículo no está estacionado en doble fila, en caso contrario se procede a comprobar el número de detecciones del vehículo.

Figura 3.36: Vehículo con punto representativo situado a 0,25 del borde inferior de su bounding box.

Figura 3.37: Vehículo con punto representativo situado a 0,1 del borde inferior de su bounding box.

Para determinar el número de detecciones de un vehículo, se compara si la cantidad de detecciones del vehículo es menor que el parámetro PERIODO_PATRON multiplicado por la cantidad de frames por segundo del video, en caso afirmativo se devuelve que no está estacionado en doble fila, ya que el tiempo que pudo haber estado en doble fila no supera el exigido por el parámetro

PERIODO_PATRON. En caso negativo, se prosigue a comprobar la velocidad promedio.

Para determinar si la velocidad promedio es menor a la velocidad mínima debajo de la cual se considera que el vehículo está detenido, primero se calcula la velocidad instantánea consultando la posición en coordenadas geográficas en el frame actual y en el frame anterior dado por el párametro INTERVALO_VEL_INSTANTANEA. Si x_n es la posición en coordenadas geográficas del vehículo en el instante actual, x_{n-1} es la posición en el frame anterior, x_{n-j} es la posición en j frames anteriores al actual, i es el parámetro INTERVALO_VEL_INSTANTANEA y fps son los fps del video, entonces la velocidad instantánea se calcula mediante la fórmula 3.9.

$$velocidad_n = \frac{x_n - x_{n-i \times fps}}{i \times fps} \tag{3.9}$$

Estas velocidades instantáneas se promedian durante el período dado por el parámetro PERIODO_PATRON. Si el promedio de estas velocidades es menor al parámetro VEL_MINIMA se prosigue a comprobar si el vehículo está estacionado, en caso contrario se retorna que no está estacionado en doble fila.

Para determinar si el vehículo está estacionado se calcula la cantidad de detecciones de dicho vehículo en las cuales estuvo en un carril externo. Si la proporción de detecciones en las cuales estuvo en un carril externo con respecto a la cantidad de detecciones totales supera el parámetro UMBRAL_DETECCIONES_ESTACIONADO, se considera que está estacionado y se retorna que no está estacionado en doble fila. En caso contrario se procede a comprobar si estuvo en semáforo en verde.

Para determinar si el vehículo estuvo en semáforo en verde, en primera instancia, con el fin de evitar ruido en las detecciones, estas se agrupan en segundos en lugar de en frames. La forma de agrupar es que si las detecciones del color verde superan el parámetro UMBRAL_SEMAFORO_VERDE, se considera que el semáforo estuvo en verde en ese segundo, en caso contrario se considera que estuvo en rojo. En segunda instancia se consideran las detecciones por segundo de los colores en el período en el que se considera el estacionamiento en doble fila. Si la mayoría de detecciones por segundo es del color verde, se considera que estuvo en semáforo en verde, se suma uno a un contador de detecciones provisorias y se prosigue comprobando si la cantidad de ocurrencias del patrón es mayor a un mínimo de detecciones. Si la mayoría

de detecciones no es de color verde se devuelve que no estuvo estacionado en doble fila.

Para determinar si las ocurrencias del patrón son mayores a un mínimo con el fin de evitar detecciones espúreas, se comprueba si el contador de detecciones provisorias recientes es mayor o igual al parámetro MINIMO_DETECCIONES. En caso afirmativo se devuelve que está estacionado en doble fila, en caso contrario se devuelve que no está estacionado en doble fila.

Capítulo 4

Evaluación experimental

En este capítulo se presentan los experimentos realizados asociados a distintas etapas requeridas para alcanzar la solución final. Se evaluarán los datos etiquetados, los anchors elegidos, los detectores de objetos entrenados junto con los algoritmos de detección de eventos. A continuación se presenta el análisis experimental realizado sobre las etapas mencionadas.

4.1. Descripción de la metodología de evaluación

El capitulo de evaluación experimental presenta el entrenamiento realizado de los modelos de detección de objetos con los datos etiquetados que fueron mencionados en la sección 3. Gran parte del entrenamiento realizado fue llevado a cabo en la plataforma Cluster UY descripto por Nesmachnow e Iturriaga (2019). El resto de las pruebas y entrenamientos fueron realizados en computadoras personales. Para la evaluación de los algoritmos de detección de eventos se realizaron casos de prueba a partir de casos reales y sintéticos para cada patrón. Dadas las características de cada evento, se buscó generar una cantidad de pruebas suficientemente representativas para su evaluación.

4.2. Análisis de datasets considerados

Se analizaron las anotaciones de los conjuntos de entrenamiento para los datasets de COCO (versión 2017) por Lin *et al.* (2014), VOC por Everingham *et al.* (2010) , VisDrone (versión 2019) por Zhu *et al.* (2018) y OpenImages

(versión v5) por Kuznetsova *et al.* (2018). Se analizó el porcentaje de imágenes que contienen anotaciones de vehículos terrestres y el porcentaje de las anotaciones que corresponden a esos vehículos.

VOC (combinando sus ediciones 2007 y 2012) contiene la menor cantidad de anotaciones de vehículos con 7368 anotaciones en 3821 imágenes, representando estas clases el 13.37 % de las anotaciones totales y 17.31 % de las imágenes del dataset. COCO es el dataset más grande en cuanto al número de imágenes, con 75747 anotaciones de vehículos en 19759 imágenes, representando el 8.81 % de las anotaciones totales y 16.86 % de las imágenes del dataset. VisDrone no tiene tantas imágenes como COCO, pero consiste en imágenes de alta resolución de zonas urbanas con mayor cantidad de vehículos en cada imagen, por lo que posee el mayor número de anotaciones de vehículos. Considerando solamente anotaciones de la clase 'car', el tamaño promedio de los bounding boxes es de 3.55 % de la imagen para VisDrone y 10.32 % para COCO. En una secuencia obtenida en la intersección de Bulevar Artigas y Avenida Italia el tamaño promedio de los bounding boxes estuvo entre 4 % y 5 %, valor que se encuentra contenido entre las cantidades de VisDrone y COCO respectivamente.

Open Images es un dataset que contiene en total 15.4 millones de bounding boxes para 600 clases en 1.9 millones de imágenes con anotaciones de localización de objetos en su versión actual disponible en Google (2020). Las clases de interés para este proyecto son *Car*, *Truck*, *Van*, *Bus*, *Motorcycle*, *Bicycle*, *Ambulance* y *Taxi*. Debido al gran número de imágenes de automóviles para la clase *Car* se utilizó solamente un subconjunto de aproximadamente 20000 imágenes a efectos de reducir el tiempo de entrenamiento y mejorar el balance de clases. Se utilizó el software OIDv4_ToolKit de Vittorio (2018), para realizar la descarga del dataset. Un inconveniente encontrado es que existen algunos objetos anotados como múltiples clases, por ejemplo Van y Truck, por lo que se deduplica la anotación en esos casos para asignarle una sola clase.

A continuación se presenta el análisis realizado de los dataset abiertos disponibles que tienen etiquetado de vehículos. De los conjuntos de datos se analizan atributos como la cantidad de imágenes, el porcentaje por clase, la cantidad de anotaciones, su porcentaje por clase y la cantidad promedio de anotaciones por imagen por clase. Los datasets utilizados son VOC 2007 y VOC 2009 (tabla 4.1), COCO (tabla 4.2), VisDrone (tabla 4.3) y Open Images (tabla 4.4).

	Imágenes		Anotaciones		
Clase	Total	%	Total	%	por imagen
car	2059	9.33	4033	7.32	1.96
bicycle	853	3.86	1226	2.22	1.44
motorbike	853	3.86	1170	2.12	1.37
bus	650	2.94	939	1.70	1.44
Total vehículos	3821	17.31	7368	13.37	1.93
Total	22077	100.00	55114	100.00	2.50

Tabla 4.1: Análisis de anotaciones por clase para VOC2007+VOC2019

	Imágenes		Anotaciones		
Clase	Total	%	Total	%	por imagen
car	12251	10.45	43867	5.10	3.58
truck	6127	5.23	9973	1.16	1.63
bus	3952	3.37	6069	0.71	1.54
motorcycle	3502	2.99	8725	1.01	2.49
bicycle	3252	2.77	7113	0.83	2.19
Total vehículos	19759	16.86	75747	8.81	3.83
Total	117226	100.00	860001	100.00	7.34

Tabla 4.2: Análisis de anotaciones por clase para COCO

	Imágenes		Anotaciones		
Clase	Total	%	Total	%	por imagen
car	6133	94.78	144867	42.21	23.62
van	4948	76.46	24956	7.27	5.04
motor	4237	65.48	29647	8.64	7.00
truck	3551	54.88	12875	3.75	3.63
bicycle	2755	42.57	10480	3.05	3.80
bus	2023	31.26	5926	1.73	2.93
Total vehículos	6318	97.64	228751	66.65	36.21
Total	6471	100.00	343205	100.00	53.04

Tabla 4.3: Análisis de anotaciones por clase para VisDrone

	Imágenes		Anotaciones		
Clase	Total	%	Total	%	por imagen
car	23798	38.10	62823	43.63	2.64
truck	5689	9.11	8301	5.76	1.46
van	4055	6.49	5805	4.03	1.46
bus	5314	8.51	8760	6.08	1.43
motorcycle	4899	7.84	8598	5.97	1.76
bicycle	10553	16.90	19932	13.84	1.89
Total Vehículos	54308	86.95	114219	79.32	2.10
Total	62457	100.00	143995	100.00	2.31

Tabla 4.4: Análisis de anotaciones por clase para Open Images (solo subconjunto empleado)

A partir de los resultados obtenidos se obtuvo que VisDrone tiene mayor cantidad de anotaciones de vehículos por imagen que los otros datasets. Una razón para la gran diferencia en relación a los demás es que VisDrone es específico para tráfico mientras que los otros datasets no lo son. En relación a la cantidad de anotaciones VisDrone obtuvo la mayor cantidad de anotaciones.

4.3. Evaluación de pesos pre-entrenados para detección de vehículos

Se evaluaron distintos pesos pre-entrenados sobre un video de 21 de Septiembre y Bulevar Artigas, para determinar el efecto que la estructura de la red y el conjunto de entrenamiento tiene sobre la detección. En cuanto al dataset de entrenamiento para modelos livianos, los entrenados sobre VisDrone tuvieron mejores resultados que los de COCO, mientras que para modelos completos los de COCO obtienen resultados ligeramente mejores. Se observó que los pesos de YOLOv3 completos (pertenecientes al conjunto de modelos completos) obtuvieron mejores resultados que métodos más recientes como SPP. Una posible razón de la diferencia de resultados es que los cambios recientes que mejoraron los resultados del modelo sobre COCO también causaron mayor sobreajuste al dataset de COCO.

Debido a que pesos pre-entrenados para COCO y Visdrone tienen distintas clases (en particular Visdrone tiene una clase *van* que se divide entre *car* y *truck* de COCO, además Visdrone divide las personas entre *person* y *peatons*),

no es posible compararlos con exactitud mediante la medida AP. Para poder realizar una comparación válida se calculó los mAP simplificando las clases de personas, motocicletas y bicicletas a clase *person* y demás clases a una clase *vehicle*. Los valores de AP no son comparables a los obtenidos sin simplificar las clases.

Para mejorar la eficacia de modelos livianos se entrenó sobre un dataset combinado conformado por COCO y VisDrone. En la tabla 4.5 se observan los resultados de cada modelo evaluados en el conjunto de validación referido en la sección 3.6.2.

	Vehicle	Person
COCO (pre-entranado)	24.77 %	4.20 %
Visdrone (pre-entrenado)	26.04 %	4.29 %
COCO+Visdrone (entrenado inicial)	30.54 %	19.35 %

Tabla 4.5: AP con umbral IoU de 0.5 y clases simplificadas para arquitectura tiny.

Utilizar una mezcla de ambos datasets mejoró los resultados, pero la detección de clases minoritarias, bicicletas, motocicletas y buses no fue satisfactoria, para reducir este problema se incluyó imágenes de clases minoritarias múltiples veces en la lista de entrenamiento, lo que al aplicar data augmentation resulta en imágenes distintas.

4.4. Elección de Anchors

Los anchors son bounding boxes que se generan para algunas redes neuronales convolucionales utilizadas en la detección de objetos. Los anchors representan combinaciones de escala y aspect ratio, lo que permite representar la mayor parte de los bounding boxes que pueden ocurrir en una imagen de entrada. Los anchors se ubican a lo largo de la imagen de forma de cubrir los posibles bounding boxes. A partir de los anchors se generan Δx_{center}, Δy_{center}, $\Delta width$, $\Delta height$ que son offsets a partir de los anchors de base. Para cada convolución realizada, los anchors tienen como centro la ventana de convolución, lo que distingue a los anchors base de los de cada convolución. Se realizaron pruebas con los siguientes conjuntos de anchors:

- Anchors por defecto.
- Anchors derivados del conjunto de vehículos de COCO y VisDrone.
- Anchors derivados del conjunto de MIO-TCD.

Para cada prueba se entrenó un detector sobre 32000 batches de 64 imágenes (tamaño por defecto y recomendado para un batch), sin utilizar entrenamiento multi-resolución, sobre los conjuntos de Visdrone y OpenImages (excluyendo imágenes con vehículos sin clasificar). En cada prueba realizada se entrenó un modelo YOLOv3 tiny con uno de los tres conjuntos de anchors. En la tabla 4.6 se muestran los valores de AP obtenidos con distintos conjuntos de anchors para las distintas clases de vehículo sobre un conjunto de fotogramas de Tres Cruces y de 21 de Septiembre y Bulevar Artigas.

	Anchors Origen		
Clase	COCO+Visdrone	COCO	MIO-CTD
AP person	19.49 %	18.60 %	**19.94 %**
AP bicycle	10.22 %	**11.79 %**	8.49 %
AP car	**58.97 %**	50.14 %	52.36 %
AP motorcycle	2.33 %	**2.48 %**	2.36 %
AP bus	43.91 %	**45.89 %**	39.85 %
AP truck	13.19 %	14.91 %	**15.56 %**
AP van	7.83 %	5.53 %	**7.86 %**
mAP Vehículos	**22.74 %**	21.79 %	21.08 %

Tabla 4.6: AP para clases relevantes para los anchors obtenidos a partir de una combinación de COCO y Visdrone, COCO (anchors por defecto de YOLOv3) y utilizando el dataset MIO-CTD. Las métricas fueron obtenidas con umbral de confianza 0.25.

A partir de los resultados obtenidos se puede observar que los anchors de COCO+Visdrone obtienen mejor mAP en vehículos que los otros modelos, lo que se debe principalmente a que obtiene mejores valores para la clase car (la que tiene mayor representación en el dataset). Para las otras clases consideradas los otros modelos obtienen mejores valores de mAP.

4.5. Resultados de entrenamiento de detectores de objetos

Para el entrenamiento de los algoritmos de detección de objetos se utilizó el conjunto de entrenamiento con los fotogramas etiquetados manualmente en la sección 3.6.2. Para la evaluación de la corrección de los modelos se utilizó el conjunto de validación. Los conjuntos de entrenamiento y validación se detallan en la sección 3.6.2. Al evaluar en el conjunto de evaluación es posible medir la habilidad de generalización de cada algoritmo a información no vista previamente. A continuación se presentan los modelos que fueron entrenados, junto con su dataset de entrenamiento correspondiente. Para algunos de los modelos presentados se agrega a su nombre 30k, lo que significa que se utilizó un dataset de 6000 imágenes de 21 de Septiembre y Bulevar Artigas y Tres Cruces en su conjunto de entrenamiento y que el modelo fue entrenado durante 30000 iteraciones, por otro lado los modelos que se presentan como retrained se realizó un entrenamiento en dos fases: en una primera fase se entrenó sin incluir esas imágenes y luego se refinó el modelo incluyendo esas imágenes como segunda fase. Se realizaron entrenamientos con diferentes datasets con el objetivo de conocer qué algoritmo obtiene mejores resultados en el conjunto de validación. Los modelos generados son los siguientes:

- tiny inicial: modelo tiny de dos capas, el cual fue entrenado sobre COCO y Visdrone.
- tiny+: modelo tiny de dos capas, el cual fue entrenado sobre OpenImage y Visdrone.
- tiny+ (30k): modelo tiny de dos capas, el cual fue entrenado sobre OpenImage y Visdrone. El modelo luego se entrenó con datos de 21 de Septiembre y Bulevar Artigas y Tres Cruces.
- tiny m: modelo tiny de dos capas, el cual fue entrenado sobre OpenImage y Visdrone aplicando mejora de datos de mosaico. El modelo es luego entrenando sobre datos de 21 de Septiembre y Bulevar Artigas y Tres Cruces.
- tiny auto: modelo tiny de dos capas, el cual fue entrenado sobre Visdrone y sobre anotaciones generadas automáticamente. Las anotaciones automáticas fueron generadas a partir de los videos de Tres Cruces y de Rodó mirando al norte (videos que no son usados para validación). Adi-

cionalmente se utilizaron anotaciones manuales sobre Tres Cruces junto con las automáticas.

- tiny auto (30k): modelo tiny auto, el cual fue entrenado sobre datos de 21 de Septiembre y Bulevar Artigas y Tres Cruces.
- tiny 3ln: modelo tiny de tres capas, el cual fue entrenado sobre OpenImage y Visdrone sin activar la mejora de datos de mosaico de Darknet.
- tiny 3lm: modelo tiny de tres capas, el cual fue entrenado sobre OpenImage y Visdrone. En el entrenamiento se utilizó la mejora de datos de mosaico.
- tiny 3l auto: modelo tiny de tres capas, el cual fue entrenado sobre Visdrone y anotaciones automáticas.
- tiny 3l auto retrained: modelo tiny 3l auto el cual fue re-entrenado sobre datos de 21 de Septiembre y Bulevar Artigas y Tres Cruces.
- tiny 3l var2: modelo tiny el cual fue modificado con anchors derivados sobre anotaciones automáticas y aumento de resolución para objetos de tamaño reducido.
- tiny 3l var2 retrained: modelo tiny de tres capas entrenado sobre OpenImage y Visdrone con ajustes sobre los conjuntos de datos que luego se entrena en datos de 21 de Septiembre y Bulevar Artigas y Tres Cruces. AlexeyAB (2019)

Una forma de medir la complejidad computacional de un algoritmo es midiendo la cantidad de operaciones en punto flotante requeridas para utilizarlo. Resulta de interés conocer para los modelos utilizados el costo computacional para conocer su utilidad en aplicaciones en tiempo real. Se analiza para algunos de los modelos utilizados la cantidad de operaciones de punto flotante necesarias para su uso por segundo. La tabla 4.7 muestra la comparación del costo computacional para algunos de los modelos utilizados. Todos los modelos entrenados para detección de objetos utilizan alguna de las arquitecturas definidas en la tabla como base, por lo que solo se muestran las más representativas.

A partir de la tabla comparativa se puede ver que el modelo tiny tiene cerca de la mitad de operaciones por ejecución que el modelo tiny 3l var2, lo que lo hace mejor para ejecución en tiempo real.

Las tablas 4.8, 4.9, 4.10, 4.11 y 4.12 muestran los valores de AP por clase para los modelos entrenados para variaciones de los modelos tiny, junto con su continuación, además de los modelos tiny3l en conjuntos de datos abiertos,

Arquitectura	FLOPS
tiny	8.276×10^9
tiny 3l	10.809×10^9
tiny 3l var2	15.792×10^9

Tabla 4.7: Cantidad de operaciones de punto flotante por segundo para los modelos tiny, tiny 3l y tiny 3l var2.

tiny3l y tiny3l var2 respectivamente. En todos los casos los resultados obtenidos corresponden al conjunto de validación descripto en la sección 3.6.2.

Clase	tiny inicial	tiny+	tiny+ (30k)
AP person	1.78 %	2.72 %	**4.41 %**
AP bicycle	0.11 %	0.89 %	**2.05 %**
AP car	34.27 %	42.48 %	**45.20 %**
AP motorcycle	0.07 %	4.75 %	**10.73 %**
AP bus	26.90 %	22.20 %	**45.82 %**
AP truck	2.64 %	8.47 %	**8.99 %**
AP van	18.05 %	**36.87 %**	14.20 %
mAP Vehículos	13.67 %	19.28 %	**21.17 %**
mAP Personas + Vehículos	11.97 %	16.91 %	**18.77 %**
Precision	31.95 %	40.75 %	**56.23 %**
Recall	23.15 %	**26.50 %**	23.68 %
F1	26.85 %	32.11 %	**33.33 %**
IoU Detecciones Promedio	21.28 %	27.82 %	**40.23 %**

Tabla 4.8: AP para algunas de las clases más relevantes, mAP para vehículos, mAP de personas y vehículos, precision, recall, F1 e IoU promedio de las detecciones para los modelos tiny inicial, tiny+ y tiny+ (30k). Se utiliza en todos los casos umbral de confianza 0.25 y umbral de IoU de 0.5. Las métricas se obtienen a partir del conjunto de validación.

A partir de los resultados se obtuvo que el pseudo labeling tiene efectos positivos en la detección de objetos, este procedimiento resultó en un fuerte aumento (10 % en mAP, 8 % en F1 y 5 % en IoU) de las métricas de evaluación. Esta mejora es superior a la de las técnicas utilizadas para adaptar los datasets OpenImages y Visdrone al tamaño de los objetos (creación de mosaicos y ampliación mediante clustering respectivamente). Por otra parte el modelo de tres capas YOLO resulta en una mejora de mAP de aproximadamente 5 % (tiny 3l auto vs tiny auto) a costa de un mayor costo computacional.

Clase	tiny m	tiny auto	tiny auto (30k)
AP person	6.73 %	4.63 %	**9.27 %**
AP bicycle	**7.41 %**	0.33 %	5.51 %
AP car	39.05 %	**53.61 %**	50.68 %
AP motorcycle	13.94 %	12.33 %	**14.31 %**
AP bus	23.36 %	**58.11 %**	53.00 %
AP truck	9.07 %	**21.14 %**	12.57 %
AP van	18.46 %	38.06 %	**38.43 %**
mAP Vehículos	18.55 %	**30.60 %**	29.08 %
mAP Personas + Vehículos	16.86 %	**26.89 %**	26.25 %
Precision	**64.45 %**	52.48 %	61.43 %
Recall	17.48 %	**30.59 %**	29.68 %
F1	27.50 %	38.65 %	**40.02 %**
IoU Detecciones Promedio	**47.59 %**	37.14 %	43.90 %

Tabla 4.9: AP para algunas de las clases más relevantes, mAP para vehículos, mAP de personas y vehículos, precision, recall, F1 e IoU promedio de las detecciones para los modelos tiny m, tiny auto y tiny auto (30k). Se utiliza en todos los casos umbral de confianza 0.25 y umbral de IoU de 0.5. Las métricas se obtienen a partir del conjunto de validación.

	Visdrone			
Clase	(Solo)	+COCO	+OI	+OI+COCO
AP person	9.57 %	9.66 %	**10.95 %**	6.71 %
AP bicycle	**5.18 %**	2.34 %	2.47 %	2.18 %
AP car	**60.16 %**	57.92 %	54.29 %	57.77 %
AP motorcycle	11.82 %	11.87 %	**13.35 %**	6.25 %
AP bus	26.50 %	**27.53 %**	22.35 %	20.04 %
AP truck	**12.01 %**	5.78 %	5.98 %	3.54 %
AP van	25.32 %	29.28 %	**36.08 %**	24.43 %
mAP Vehículos	**23.50 %**	22.45 %	22.42 %	19.04 %
mAP Personas + Vehículos	**21.51 %**	20.63 %	20.78 %	17.27 %
Precision	56.11 %	**58.78 %**	58.54 %	54.88 %
Recall	**32.31 %**	29.75 %	29.62 %	28.92 %
F1	**41.00 %**	39.51 %	39.34 %	37.88 %
Average IoU	40.27 %	**42.04 %**	41.89 %	38.93 %

Tabla 4.10: AP para algunas de las clases más relevantes, mAP para vehículos, mAP de personas y vehículos, precision, recall, F1 e IoU promedio de las detecciones para los modelos de arquitectura tiny 3l (Visdrone + COCO), tiny 3l (Visdrone), tiny 3l (Visdrone + Open Images) y tiny 3l (Open Images + Visdrone + COCO). Se utiliza en todos los casos umbral de confianza 0.25 y umbral de IoU de 0.5. Las métricas se obtienen a partir del conjunto de validación.

Clase	tiny			
	3ln	3lm	3l auto	3l auto retrained
AP person	9.99 %	11.80 %	14.82 %	**17.76 %**
AP bicycle	8.85 %	3.65 %	5.31 %	**11.84 %**
AP car	63.29 %	60.14 %	62.12 %	**69.67 %**
AP motorcycle	11.43 %	7.58 %	15.67 %	**15.78 %**
AP bus	30.72 %	25.62 %	**57.82 %**	50.82 %
AP truck	**20.97 %**	19.44 %	12.41 %	19.24 %
AP van	16.27 %	23.01 %	**47.90 %**	46.11 %
mAP Vehículos	25.26 %	23.24 %	33.54 %	**35.58 %**
mAP Personas + Vehículos	23.07 %	21.61 %	30.86 %	**33.03 %**
Precision	64.14 %	57.66 %	67.39 %	**70.09 %**
Recall	29.95 %	28.69 %	32.45 %	**37.30 %**
F1	40.83 %	38.32 %	43.81 %	**48.69 %**
Average IoU	45.71 %	41.13 %	50.46 %	**51.28 %**

Tabla 4.11: AP para algunas de las clases más relevantes, mAP para vehículos, mAP de personas y vehículos, precision, recall, F1 e IoU promedio de las detecciones para los modelos tiny 3ln, tiny 3lm, tiny 3l auto (sin entrenar nuevamente), tiny 3l auto retrained. Se utiliza en todos los casos umbral de confianza 0.25 y umbral de IoU de 0.5. Las métricas se obtienen a partir del conjunto de validación.

Clase	tiny 3l var2	tiny 3l var2 retrained
AP person	11.85 %	**24.31 %**
AP bicycle	0.91 %	**18.61 %**
AP car	**66.77 %**	59.48 %
AP motorcycle	14.27 %	**15.47 %**
AP bus	**61.76 %**	57.88 %
AP truck	**20.28 %**	5.88 %
AP van	**41.90 %**	32.18 %
mAP Vehículos	**34.32 %**	31.58 %
mAP Personas + Vehículos	**31.11 %**	30.54 %
Precision	58.13 %	**65.78 %**
Recall	**37.52 %**	35.51 %
F1	45.60 %	**46.12 %**
Average IoU	42.15 %	**48.51 %**

Tabla 4.12: AP para algunas de las clases más relevantes, mAP para vehículos, mAP de personas y vehículos, precision, recall, F1 e IoU promedio de las detecciones para los modelos tiny 3l var2 y tiny 3l var2 retrained. Se utiliza en todos los casos umbral de confianza 0.25 y umbral de IoU de 0.5. Las métricas se obtienen a partir del conjunto de validación.

A partir de los resultados obtenidos los modelos que presentaron mejores resultados en mAP vehículos en el conjunto de validación son los modelos tiny 3l auto retrained (35.58 %), seguido del modelo tiny 3l auto (33.54 %) y el tiny auto (30.60 %). En relación al mAP de vehículos y personas, el modelo tiny 3l auto retrained (33.03 %) obtiene también los mejores resultados, seguido del modelo tiny 3l var2 (31.11 %) y el tiny 3l auto (30.86 %).

4.6. Resultados de redes neuronales para obtener el estado del semáforo

Para mejorar los resultados de la detección del estado (color) del semáforo se entrenaron redes neuronales como alternativa al modelo basado en OpenCV 3.11.2.1. Los resultados del sistema de OpenCV sobre los primeros cinco minutos de Avenida Brasil y Bulevar Artigas con el sistema basado en OpenCV se presentan en la tabla 4.13.

Los resultados obtenidos inicialmente resultaron inferiores al sistema de OpenCV, resultando en niveles de recall insuficientes a pesar de buenos niveles de precision, potencialmente debido a que la gran mayoría del dataset Bosch está compuesto por imágenes frontales, para subsanar este problema se agregaron imágenes de los primeros minutos del video de 21 de Septiembre y Bulevar Artigas, dividiendo las imágenes entre las categorías existentes, sin agregar ejemplos negativos adicionales. Las nuevas imágenes resultaron en una mejora considerable en todas las categorías para el modelo de seis capas, alcanzando un F1 promedio de 89 % frente a 13 %, comparado con 49 % del sistema de OpenCV.

Realizando el mismo entrenamiento con un modelo de cuatro capas se obtuvieron resultados similares, por lo que se utiliza este modelo al tener menor costo computacional. Adicionalmente se redujo la resolución de 32 píxeles de ancho por 48 píxeles de alto a 24 por 32 sin pérdida importante de exactitud. Se realizaron pruebas con distinto número de neuronas en la capa completamente conexa (utilizando la cantidad de neuronas como hiper-parámetro), ya que esta capa representaba la mayor parte del esfuerzo computacional. La arquitectura inicial especifica 512 neuronas, el número está relacionado con el número de categorías a predecir, las pruebas indican que reducir el número de neuronas de 512 a 128 resulta en una reducción del costo computacional sin

comprometer los resultados, pero se comprobó que utilizar un número inferior de neuronas comienza a tener efectos negativos en los resultados. Reducir el número de neuronas a 32 si bien no afecta los resultados para las clases rojo y verde tiene efecto sobre la clase amarillo, por lo que se mantuvo la cantidad de neuronas en 128. En la tabla 4.14 se presentan los resultados de los modelos para imágenes de semáforos de tamaño 32×48 mientras que en la tabla 4.15 se presentan los resultados para imágenes de semáforos de tamaño 24×32. Adicionalmente se logró un pequeña mejora al combinar ambos modelos. Cuando el clasificador de OpenCV no puede determinar el color del semáforo se utiliza la detección obtenida por la CNN. Este método se describe como *Híbrido* en la tabla 4.15.

	Precision	Recall	F1	Muestras
Verde	1.00	0.55	0.71	900
Rojo	0.94	0.7	0.8	1422
Amarillo	0.41	0.33	0.37	78
micro avg	0.63	0.63	0.63	2400
weighted avg	0.95	0.63	0.75	2400

Tabla 4.13: Medidas Precision, Recall y F1 para el método de OpenCV (tamaño a 24×32) junto con la cantidad de muestras de cada caso. Se detalla también promedio micro y macro. Valores obtenidos a partir de los primeros cinco minutos del video de Avenida Brasil y Bulevar Artigas.

	Clases			Promedio		
Modelo	Verde	Roja	Amarilla	Macro	Ponderado	Parámetros
OpenCV	0.77	0.80	0.21	0.59	0.77	N/A
DeepThin	0.73	0.97	**0.67**	**0.79**	0.87	1942
6l, 512fcl	**0.93**	**0.98**	0.47	**0.79**	**0.94**	2035

Tabla 4.14: Valores F1 de modelos de detección (32×48) del estado del semáforo para los primeros cinco minutos de Avenida Brasil y Bulevar Artigas, así como el número de parámetros para cada CNN. Se detalla F1 por clase y promedio micro y macro. Parámetros en miles.

Durante la evaluación del modelo se detectó que una causa de la menor efectividad del modelo basado en OpenCV es la sobreexposición de luces de semáforo, disminuyendo la saturación de las luces, problema que ocurre fundamentalmente en el horario nocturno. Para disminuir este problema se utilizó ecualización sobre el canal de saturación mediante OpenCV. La ecualización

	Clases			Promedio		
Modelo	Verde	Roja	Amarilla	Macro	Ponderado	Parámetros
DeepThin	0.74	0.97	0.32	0.68	0.86	911
512fcn	0.91	0.97	0.54	0.81	0.93	855
256fcn	0.89	0.96	0.44	0.76	0.92	460
128fcn	0.91	0.96	0.66	**0.84**	0.93	263
64fcn	0.89	0.96	**0.67**	**0.84**	0.92	164
32fcn	0.89	**0.99**	0.60	0.83	**0.94**	115
16fcn	0.76	0.93	0.10	0.60	0.84	90
8fcn	0.80	0.95	0.00	0.58	0.87	78
Híbrido 128fcn	**0.93**	0.96	0.64	0.84	**0.94**	263

Tabla 4.15: Valores F1 de modelos de detección (24×32) del estado del semáforo con cuatro capas para los primeros cinco minutos de Avenida Brasil y Bulevar Artigas, así como el número de parámetros de cada CNN. Se detalla F1 por clase y promedio micro y macro. Parámetros en miles.

se combina con la imagen original para evitar niveles artificiales de saturación, utilizando una proporción de 50 %, se obtuvieron resultados razonables. Los cambios aplicados resultan en un cambio pequeño en condiciones de iluminación favorables, pero un cambio mayor en condiciones desfavorables. Aplicar ecualización antes de la red neuronal no resulta en mejoras, pues el data augmentation ya simula condiciones cambiantes de luz hasta cierto punto.

4.7. Evaluación de los algoritmos de detección de eventos

Esta sección presenta el proceso de generación de tests para la evaluación de los patrones de detección de eventos junto con las métricas de evaluación y los resultados obtenidos.

4.7.1. Generación de tests sintéticos

Dadas las características de la problemática abordada en el proyecto de grado, la cantidad de eventos a utilizar para evaluar los detectores es relativamente baja dados los videos obtenidos (cerca de dos eventos para 3.4 horas de video). A raíz de la baja cantidad de eventos obtenidos, además de utilizar pruebas reales con esos casos, se optó por la realización de casos de prueba

sintéticos. Se evaluaron herramientas como Game Maker y CVAT para generar los escenarios. Si bien Game Maker tiene opciones para generar videos con vehículos ficticios, CVAT mostraba mejores prestaciones para emular la detección de vehículos en los escenarios ficticios. De esta forma, se generan los escenarios con un generador de bounding boxes en video, permitiendo mantener los identificadores de los vehículos. Posteriormente se procede a etiquetar de forma manual en un archivo json, en que rango de frames se detecta el evento y para que id de vehículo fue detectado. En las pruebas reales solo es necesario etiquetar el vehículo que realiza el evento para poder verificar que al emitir una detección de evento se está refiriendo al mismo vehículo. En las pruebas reales se utiliza un detector de objetos mientras que en las sintéticas no.

4.7.2. Métricas de detección de eventos

Se realizó una relevación de trabajos relacionados para obtener que métricas utilizar para la detección de eventos. La problemática surge de que para que se contabilice una detección de evento como certera, se debe poder detectar que se refiere al mismo objeto a nivel de frame (en dos dimensiones saber que se habla del mismo vehículo) y a nivel temporal (saber que los frames detectados y los del ground truth refieren al mismo evento). En la bibliografía se utiliza IoU para realizar el match a nivel de frame. En Bochinski *et al.* (2017) se utilizó IoU = 0.5 para la comparación del valor de la métrica PR-MOTA aplicado al dataset DETRAC de tracking y detección de vehículos. Se decide utilizar 0.5 como valor de IoU para identificar que se refiere al mismo vehículo al evaluar las métricas. A nivel temporal en Ward *et al.* (2011) se mencionaron algunos enfoques para detección de eventos en series temporales, se presentó el caso de utilizar detecciones puntuales por frame y clasificar por frame en TP, FP, FN, TN para luego calcular precision y recall. También se mencionó tomar un evento como true positive si un frame de una detección tiene intersección con el intervalo del ground truth de ese evento.

Para los tres tipos de eventos detectados se utilizó precision, recall y f-score como métricas. Dado que el problema de la detección de eventos en video tiene una nueva dimensión en relación a la detección de eventos que se realiza únicamente a nivel espacial, ya que se debe tener información sobre en que momentos del video se realiza la detección además de en que zona del video se realiza la detección, se generó un método que permite obtener las

métricas anteriormente descriptas teniendo en cuenta ambos factores de forma correcta (posición y tiempo). A continuación se presenta el pseudo-código del algoritmo generado para obtener los true positives y false positives necesarios para calcular precision y recall 1.

Algorithm 1 Algoritmo de detección de elementos requeridos para cálculo de métricas (TP, FP)

```
procedure CALCULAR_TP_FP(ground_truth, detecciones)
    total_vp ← 0
    total_fp ← 0
    for cada patrón p en ground_truth do
        for cada test t del patrón p do
            for cada evento e del test t do
                aux_vp ← 0
                aux_fp ← 0
                detecciones_asociadas_a_evento_e ← [Todas las detecciones en
las que el bbox del vehículo del evento tenga una intersección con el vehículo
asociado al ground_truth con IoU mayor a X==0.25]
                for cada detección d de detecciones_asociadas_a_evento_e do
                    if el frame de la detección d está dentro del intervalo tem-
poral del evento del ground truth e then
                        aux_vp ← aux_vp + 1
                    else
                        aux_fp ← aux_fp + 1
                if aux_vp ≥ aux_fp then
                    total_vp ← total_vp + 1
                else
                    total_fp ← total_fp + 1
    return total_vp, total_fp
```

4.7.3. Consideraciones de los tests generados

La distribución de los test en reales y sintéticos resultó muy ligado a la disponibilidad del evento en los datos provistos por la intendencia. De esta forma, para el caso de cruce en roja y estacionamiento doble fila, fue posible agregar casos de prueba reales, lo que no fue posible para el caso de movimiento brusco; para el cual se agregaron algunos casos de prueba para evaluar la cantidad de falsos positivos además de las pruebas sintéticas.

Se realizó una evaluación inicial para estimar el tiempo requerido para realizar cada prueba sintética, en una prueba sintética se debe contar con los

carriles etiquetados, la información de mapeo geográfica, el video en el cual se trabajó, las etiquetas de CVAT para el vehículo que realiza el evento y un archivo json con metadata temporal de los eventos ocurridos para el vehículo. Para las pruebas sintéticas, se estimó que el tiempo necesario para realizar una prueba una vez se tienen los elementos necesarios era de siete minutos. Si bien ese tiempo no era relativamente grande, conseguir los elementos necesarios para poder etiquetar un escenario requería la mayor cantidad del tiempo (cerca de una hora). La cantidad de escenarios que fue posible generar por cámara de forma sintética estaba limitado dependiendo del patrón por la variedad de los eventos generados.

4.7.4. Tests generados

A continuación se presenta para cada patrón las pruebas realizadas y el proceso de evaluación junto con los resultados obtenidos y una interpretación de los mismos.

4.7.4.1. Tests de movimiento brusco

En el caso de detección de movimientos bruscos, no se cuenta con casos del evento en los videos obtenidos. A raíz de la falta de eventos a evaluar se optó por generar casos de prueba a partir de videos de distintas cámaras de forma de generar una cantidad de pruebas más diversa. Se utilizaron segmentos de video de las siguientes cámaras:

- 21 de Setiembre y Bulevar Artigas (tres pruebas sintéticas, dos de falsos positivos, de un minuto de duración cada una)
- Bulevar España y Libertad (tres pruebas sintéticas)
- Bulevar Artigas y Rodó (una prueba sintética)
- Sarmiento y 21 de Setiembre (cuatro pruebas sintéticas)

En total se generan once pruebas sintéticas y dos pruebas de falsos positivos. Una prueba de falsos positivos es una prueba en la que se utiliza un segmento de video en el que se conoce que hay ausencia de eventos de interés y se ejecuta el detector para evaluar la cantidad de falsos eventos detectados. De esta forma es posible aproximar la cantidad de falsos positivos en una ejecución real. Para las pruebas de falsos positivos se utilizaron el primer y tercer minuto

del video de 21 de Setiembre y Bulevar Artigas (18:00hs). La configuración de los parámetros para la ejecución de las pruebas son los siguientes:

- speed_turning = 40
- min_turning_angle = 15
- max_turning_angle = 70
- timeframe_turning = 30
- turning_count_thresh = 5
- cant_points_to_average_direction = 10
- minimum_ammout_of_data_points_for_quick_turn = 30
- cant_points_to_not_use_from_tail = 10
- min_speed_for_detection_filter = 5

Se realizaron 2 experimentos, uno con la finalidad de obtener el mejor modelo para la detección (experimento 1) y uno con la finalidad de obtener los parámetros que obtienen mejores resultados a partir del modelo que obtuvo mejores resultados (experimento 2). Para ambos experimentos se evalúan las métricas de TP, FP, FN, Precision, Recall, F-Score en todos los casos de prueba generados.

Los modelos utilizados para el experimento 1 fueron los 2 modelos que obtuvieron mejores valores de mAP en el conjunto de validación, tiny 3l auto retrained (35.58 %) y tiny 3l auto (33.54 %). Se utiliza también con el objetivo de comparar con otro modelo adicional, el modelo tiny+ (30k) (21.17 %). Con los valores de los parámetros ya definidos, para el experimento 1 se obtienen los valores presentados en la tabla 4.16.

	tiny 3l auto	tiny+ (30k)	tiny 3l auto retrained
TP	**9**	**9**	**9**
FP	**2**	4	**2**
FN	**2**	**2**	**2**
Precision	0.82	0.692	**0.82**
Recall	**0.82**	**0.82**	**0.82**
F-score	**0.82**	0.75	**0.82**

Tabla 4.16: Resultados de los test sintéticos y de falsos positivos realizados para el patrón de movimiento brusco. Se indica para cada caso TP, FP, FN, Precision, Recall, F-Score.

Las métricas indican que tanto el modelo tiny 3l auto como el modelo tiny 3l auto retrained obtuvieron los mismos resultados en la detección de eventos. Se

utiliza el modelo tiny 3l auto retrained ya que obtiene los mismos valores en sus métricas que el tiny 3l auto pero con mayor mAP de vehículos en el conjunto de validación. En todos los casos 9/11 test sintéticos fueron detectados y se generaron al menos dos falsos positivos en las pruebas de falsos positivos. Los falsos negativos se corresponden a los ejemplos no detectados de las pruebas sintéticas.

Se generan para el experimento 2 resultados sobre los test de prueba con el mejor modelo de la etapa anterior modificando los valores de speed_turning y turning_count_thresh. Parámetros que controlan la velocidad mínima y la cantidad de frames en la que debe tener el vehículo el ángulo de giro correcto para que se detecte el patrón respectivamente. En la tabla 4.17 se muestran los resultados obtenidos para cada uno de los pares utilizados.

	turning_count_thresh, speed_turning		
	(5,40)	(4,35)	(1,30)
TP	9	**10**	9
FP	**2**	7	30
FN	2	**1**	2
Precision	**0.82**	0.588	0.231
Recall	0.82	**0.91**	0.82
F-score	**0.82**	0.714	0.36

Tabla 4.17: Resultados de los test sintéticos y de falsos positivos utilizando el modelo tiny3l auto retrained para distintos valores de speed_turning y turning_count_thresh. Se indica para cada caso TP, FP, FN, Precision, Recall, F-Score. En este caso se tiene para cada columna, la combinación de parámetros turning_count_thresh y speed_turning utilizados en la prueba realizada.

Los resultados indican que la primera configuración resulta óptima en la métrica F-score dado que se detectan de forma satisfactoria 9/11 tests generando una cantidad baja de falsos positivos en las pruebas destinadas a medir los falsos positivos. Si bien con la segunda configuración se detecta un evento más, subiendo el recall, igualmente aumenta la cantidad de falsos positivos. Para la tercera configuración, la cantidad de falsos positivos aumenta. El aumento en los falsos positivos baja la precision a niveles que no permiten que sea utilizable en la práctica. De esta forma dependiendo de que métrica se busque optimizar (precision o recall) se tiene una configuración que brinda mejores resultados para cada caso. De igual forma la primera configuración ofrece un

buen equilibrio, dejando baja la cantidad de falsos positivos pero detectando la mayor parte de los eventos.

4.7.4.2. Tests de cruce en luz roja

Se realizaron siete tests, sobre Bulevar Artigas y 21 de Septiembre, sobre Bulevar Artigas y Rodó, sobre Bulevar Artigas y Avenida Brasil y tres sobre Bulevar Artigas y Canelones. Se consideró que existe un cruce en luz roja si la persona o vehículo cruza con el semáforo en rojo, o cuando al inicio del video se encuentre cruzando con el semáforo en rojo. Se presentan a continuación los resultados para los videos de las cámaras de Bulevar Artigas y 21 de Septiembre en los periodos 18:00-18:05 (tabla 4.18) y 19:30-19:35 (tabla 4.19), Bulevar Artigas y Rodó de 18:00 a 18:05 (tabla 4.20), Bulevar Artigas y Avenida Brasil de 18:00 a 18:05 (tabla 4.21), Bulevar Artigas y Canelones en los periodos 18:00-18:10, 18:50-19:00 y 19:40-19:50 (tablas 4.22 a 4.24) y los resultados generales de las siete pruebas (tabla 4.25).

	Anotaciones	tiny+ (30k)	tiny 3l auto
TP	**5**	2	4
FP	14	**11**	16
FN	**1**	4	2
Precision	**0.26**	0.15	0.20
Recall	**0.83**	0.33	0.67
F1	**0.40**	0.21	0.31

Tabla 4.18: Resultados para Bulevar Artigas y 21 de Septiembre (18:00-18:05). *Anotaciones* utiliza anotaciones manuales para detección de objetos en lugar de un modelo. TP+FN=6.

	tiny+ (30k)	tiny 3l auto
TP	**2**	**2**
FP	39	**23**
FN	**3**	**3**
Precision	0.05	**0.08**
Recall	**0.40**	**0.40**
F1	0.09	**0.13**

Tabla 4.19: Resultados para Bulevar Artigas y 21 de Septiembre (19:30-19:35). TP+FN=5.

	tiny+ (30k)	tiny 3l auto
TP	6	**8**
FP	20	**6**
FN	5	**3**
Precision	0.23	**0.57**
Recall	0.55	**0.73**
F1	0.32	**0.64**

Tabla 4.20: Resultados para Bulevar Artigas y Rodó (18:00-18:05). TP+FN=11.

	tiny+ (30k)	tiny 3l auto
TP	6	**8**
FP	49	**48**
FN	10	**8**
Precision	0.11	**0.14**
Recall	0.38	**0.50**
F1	0.17	**0.22**

Tabla 4.21: Resultados para Bulevar Artigas y Avenida Brasil (18:00-18:05). TP+FN=16.

	tiny+ (30k)	tiny 3l auto
TP	**8**	**8**
FP	42	**23**
FN	**12**	**12**
Precision	0.16	**0.26**
Recall	**0.40**	**0.40**
F1	0.23	**0.31**

Tabla 4.22: Resultados para Bulevar Artigas y Canelones (18:00-18:10). TP+FN=20.

	tiny+ (30k)	tiny 3l auto
TP	**12**	**12**
FP	44	**8**
FN	**2**	**2**
Precision	0.21	**0.60**
Recall	**0.85**	**0.85**
F1	0.34	**0.71**

Tabla 4.23: Resultados para Bulevar Artigas y Canelones (18:50-19:00). TP+FN=14.

	tiny+ (30k)	tiny 3l auto
TP	**10**	8
FP	95	**15**
FN	**2**	4
Precision	0.10	**0.35**
Recall	**0.83**	0.67
F1	0.17	**0.46**

Tabla 4.24: Resultados para Bulevar Artigas y Canelones (19:40-19:50). TP+FN=12.

	tiny+ (30k)	tiny 3l auto
TP	46	**50**
FP	**30**	139
FN	38	**34**
Precision	0.13	**0.26**
Recall	0.55	**0.60**
F1	0.21	**0.37**

Tabla 4.25: Resultados generales para las siete pruebas. TP+FN=84.

Si bien la tasa de falsos positivos fue elevada lo que redunda en una precision baja, el recall en general es aceptable. A partir de los resultados se obtuvo que la cantidad de falsos positivos del patrón en relación al cruce de personas era elevado. Se identificaron algunas posibles correcciones: para la elección del lane en personas se considera el borde inferior de la detección en lugar de un punto intermedio entre el borde inferior y el centro como en el caso de los vehículos, asimismo se aumentó la velocidad mínima para determinar un cruce para reducir el efecto del ruido en las posición de las detecciones. Asimismo se exploró utilizar una CNN para detectar el color del semáforo cuando no fuera posible mediante el método de OpenCV, en el proceso se observó que es posible mejorar los resultados del método de OpenCV sobre semáforos pequeños simplemente escalando los mismos cuando la visibilidad es buena. Se presentan los resultados de estas correcciones por test en las tablas 4.26 a 4.32 y los resultados generales en la tabla 4.33.

Las modificaciones resultaron principalmente en una disminución del número de falsos positivos, lo que mejora la precision. El modelo tiny 3l resulta en métricas considerablemente superiores al tiny en este patrón al reducir el

	Sin CNN		Con CNN	
	tiny+ (30k)	tiny 3l auto	tiny+ (30k)	tiny 3l auto
TP	2	**4**	2	**4**
FP	**2**	6	3	7
FN	4	**2**	4	**2**
Precision	**0.50**	0.40	0.40	0.36
Recall	0.33	**0.67**	0.33	**0.67**
F1	0.40	**0.50**	0.36	0.47

Tabla 4.26: Resultados para Bulevar Artigas y 21 de Septiembre (18:00-18:05), TP+FN=6.

	Sin CNN		Con CNN	
	tiny+ (30k)	tiny 3l auto	tiny+ (30k)	tiny 3l auto
TP	**2**	**2**	**2**	**2**
FP	18	**8**	35	21
FN	**3**	**3**	**3**	**3**
Precision	0.05	**0.10**	0.03	0.04
Recall	**0.25**	**0.25**	**0.25**	**0.25**
F1	0.08	**0.14**	0.05	0.07

Tabla 4.27: Resultados para Bulevar Artigas y 21 de Septiembre (19:30-19:35). TP+FN=5.

	Sin CNN		Con CNN	
	tiny+ (30k)	tiny 3l auto	tiny+ (30k)	tiny 3l auto
TP	6	**8**	6	**8**
FP	22	**7**	22	**7**
FN	5	**3**	5	**3**
Precision	0.21	**0.53**	0.21	**0.53**
Recall	0.55	**0.73**	0.55	**0.73**
F1	0.31	**0.62**	0.31	**0.62**

Tabla 4.28: Resultados para Bulevar Artigas y Rodó (18:00-18:05). TP+FN=11.

número de falsos positivos y obtener menor ruido en la posición de los objetos detectados. Por otro lado se puede observar que sin utilizar la CNN se lograron mejores métricas, pues con la CNN solo se logró agregar un verdadero positivo a costa de un aumento considerable de los falsos positivos. Una posible explicación es que la mayoría de los casos del patrón ocurren en áreas donde

	Sin CNN		Con CNN	
	tiny+ (30k)	tiny 3l auto	tiny+ (30k)	tiny 3l auto
TP	8	10	9	**11**
FP	**28**	**28**	31	33
FN	8	6	7	**5**
Precision	0.22	**0.26**	0.23	0.25
Recall	0.50	0.63	0.56	**0.69**
F1	0.31	**0.37**	0.32	0.36

Tabla 4.29: Resultados para Bulevar Artigas y Avenida Brasil (18:00-18:05). TP+FN=16.

	Sin CNN		Con CNN	
	tiny+ (30k)	tiny 3l auto	tiny+ (30k)	tiny 3l auto
TP	**8**	**8**	**8**	**8**
FP	24	**11**	32	**11**
FN	**12**	**12**	**12**	**12**
Precision	0.25	**0.42**	0.20	**0.42**
Recall	**0.40**	**0.40**	**0.40**	**0.40**
F1	0.31	**0.41**	0.27	**0.41**

Tabla 4.30: Resultados para Bulevar Artigas y Canelones (18:00-18:10). TP+FN=20.

	Sin CNN		Con CNN	
	tiny+ (30k)	tiny 3l auto	tiny+ (30k)	tiny 3l auto
TP	12	**13**	12	**13**
FP	35	**2**	42	4
FN	2	**1**	2	**1**
Precision	0.26	**0.87**	0.22	0.76
Recall	0.86	**0.93**	0.86	**0.93**
F1	0.39	**0.90**	0.35	0.84

Tabla 4.31: Resultados para Bulevar Artigas y Canelones (18:50-19:00). TP+FN=14.

los semáforos son altamente visibles. Si bien con la CNN se logra un mayor recall en la detección del color del semáforo se obtiene una menor precision lo que redunda en mayor riesgo de detección incorrecta en semáforos de menor visibilidad (que antes se emitían como color desconocido y no eran tomadas en cuenta por el patrón), en el caso particular de la tabla 4.28 se observa que

	Sin CNN		Con CNN	
	tiny+ (30k)	tiny 3l auto	tiny+ (30k)	tiny 3l auto
TP	**10**	8	**10**	8
FP	85	**10**	85	11
FN	**2**	4	**2**	4
Precision	0.11	**0.44**	0.11	0.42
Recall	**0.83**	0.67	**0.83**	0.67
F1	0.19	**0.53**	0.19	0.52

Tabla 4.32: Resultados para Bulevar Artigas y Canelones (19:40-19:50). TP+FN=12.

	Sin CNN		Con CNN	
	tiny+ (30k)	tiny 3l auto	tiny+ (30k)	tiny 3l auto
TP	48	53	49	**54**
FP	214	**72**	250	94
FN	36	31	35	**30**
Precision	0.18	**0.42**	0.16	0.36
Recall	0.57	0.63	0.58	**0.64**
F1	0.28	**0.51**	0.26	0.47

Tabla 4.33: Resultados generales para las siete pruebas. TP+FN=84.

durante un tiempo considerable se detecta un semáforo como en rojo cuando se encontraba en verde, cuando el método de OpenCV lo clasificaba como desconocido. De todas formas algunos falsos positivos continúan ocurriendo debido a ruido en la detección de la velocidad.

Una dificultad que se observa en algunas pruebas, por ejemplo en el video de Bulevar Artigas y 21 de Septiembre es que el color de algunos semáforos no es visible en la cámara cuando el sol está presente en la imagen, ejemplificado en la figura 4.1, la causa puede deberse al ajuste automático de la cámara con respecto al brillo del sol. En los casos afectados se observa que la saturación del color del semáforo obtenida por la cámara aumenta al reducir la iluminación ambiental, sin embargo la saturación de la estructura del semáforo y de otras partes de la imagen no cambia significativamente, por lo que esta dificultad no puede solucionarse simplemente mediante métodos de mejora de contraste.

Figura 4.1: Serie de imágenes de un semáforo que muestra la dificultad de saturación en diferentes instantes de tiempo. Imágenes tomadas del video de 21 de Setiembre y Bulevar Artigas.

4.7.4.3. Tests de estacionamiento en doble fila

Para las pruebas se utilizaron los parámetros detallados en la sección 3.11.3.2, con los siguientes valores:

- PERIODO_PATRON = 7
- INTERVALO_VEL_INSTANTANEA = 3
- VEL_MINIMA = 1
- PUNTOS_REPRESENTATIVOS = {0.1, 0.25}
- UMBRAL_DETECCIONES_ESTACIONADO = 0.8
- UMBRAL_SEMAFORO_VERDE = 0.95
- MINIMO_DETECCIONES = 30

Los valores de los parámetros fueron determinados experimentalmente en los videos de Bulevar Artigas y 21 de Setiembre y Bulevar Artigas y Rodó, en concreto en los siguientes segmentos:

- Bulevar Artigas y 21 de Setiembre de 18:00 a 18:05
- Bulevar Artigas y Rodó de 18:00 a 18:05

Para la evaluación se utilizaron segmentos de 3 de las cámaras que se obtuvieron, ya que no se encontraron incidencias del patrón en las demás cámaras. Se utilizaron tres modelos de los mencionados en la sección 4.5, los cuales son

tiny inicial, tiny+ (30k) y tiny 3l auto retrained. En la tabla 4.34 se detallan los resultados generales y los resultados por cada segmento junto con la tabla correspondiente se detalla a continuación:

- Bulevar Artigas y 21 de Setiembre de 19:34 a 19:36 - tabla 4.35
- Bulevar Artigas y Rodó al Sur de 18:24:49 a 18:25:49 - tabla 4.36
- Bulevar Artigas y Rodó al Sur de 18:22:55 a 18:24:32 - tabla 4.37
- Bulevar Artigas y Rodó al Sur de 18:52:30 a 18:53:08 - tabla 4.38
- Bulevar Artigas y Rodó al Sur de 19:40:07 a 19:40:58 - tabla 4.39
- Bulevar Artigas y Rodó de 18:10:57 a 18:16:02 - tabla 4.40
- Bulevar Artigas y Rodó de 18:34:03 a 18:40:03 - tabla 4.41

Métrica	tiny inicial	tiny+ (30k)	tiny 3l auto retrained
TP	8	9	**10**
FP	13	13	**7**
FN	3	2	**1**
Precision	0.38	0.41	**0.59**
Recall	0.73	0.82	**0.91**
F1	0.50	0.55	**0.71**

Tabla 4.34: Resultados de los test realizados para el patrón estacionamiento en doble fila. Se indican los True Positives, False Positives y las métricas Precision, Recall y F1.

Métrica	tiny inicial	tiny+ (30k)	tiny 3l auto retrained
TP	**1**	**1**	**1**
FP	**2**	3	3
FN	**0**	**0**	**0**
Precision	**0.33**	0.25	0.25
Recall	**1**	**1**	**1**
F1	**0.50**	0.40	0.40

Tabla 4.35: Resultados de los test realizados en el segmento Bulevar Artigas y 21 de Setiembre de 19:34 a 19:36.

Métrica	tiny inicial	tiny+ (30k)	tiny 3l auto retrained
TP	**1**	**1**	**1**
FP	2	**0**	**0**
FN	**0**	**0**	**0**
Precision	0.33	**1**	**1**
Recall	**1**	**1**	**1**
F1	0.50	**1**	**1**

Tabla 4.36: Resultados de los test realizados en el segmento Bulevar Artigas y Rodó al Sur de 18:24:49 a 18:25:49.

Métrica	tiny inicial	tiny+ (30k)	tiny 3l auto retrained
TP	0	**1**	**1**
FP	3	**1**	**1**
FN	1	**0**	**0**
Precision	0	**0.5**	**0.5**
Recall	0	**1**	**1**
F1	0	**0.67**	**0.67**

Tabla 4.37: Resultados de los test realizados en el segmento Bulevar Artigas y Rodó al Sur de 18:22:55 a 18:24:32.

Métrica	tiny inicial	tiny+ (30k)	tiny 3l auto retrained
TP	**1**	**1**	**1**
FP	3	1	**0**
FN	**0**	**0**	**0**
Precision	0.25	0.5	**1**
Recall	**1**	**1**	**1**
F1	0.4	0.67	**1**

Tabla 4.38: Resultados de los test realizados en el segmento Bulevar Artigas y Rodó al Sur de 18:52:30 a 18:53:08.

Métrica	tiny inicial	tiny+ (30k)	tiny 3l auto retrained
TP	**1**	0	**1**
FP	**0**	**0**	**0**
FN	**0**	1	**0**
Precision	**1**	**1**	**1**
Recall	**1**	0	**1**
F1	**1**	0	**1**

Tabla 4.39: Resultados de los test realizados en el segmento Bulevar Artigas y Rodó al Sur de 19:40:07 a 19:40:58.

Métrica	tiny inicial	tiny+ (30k)	tiny 3l auto retrained
TP	**2**	**2**	**2**
FP	**1**	5	2
FN	**0**	**0**	**0**
Precision	**0.67**	0.29	0.5
Recall	**1**	**1**	**1**
F1	**0.8**	0.44	0.67

Tabla 4.40: Resultados de los test realizados en el segmento Bulevar Artigas y Rodó de 18:10:57 a 18:16:02.

Métrica	tiny inicial	tiny+ (30k)	tiny 3l auto retrained
TP	2	**3**	**3**
FP	2	3	**1**
FN	2	**1**	**1**
Precision	0.5	0.5	**0.75**
Recall	0.5	**0.75**	**0.75**
F1	0.5	0.6	**0.75**

Tabla 4.41: Resultados de los test realizados en el segmento Bulevar Artigas y Rodó de 18:34:03 a 18:40:03.

En los tests se priorizó optimizar la performance en cuanto a la métrica de Recall, debido a que si el sistema es usado por un operador es mejor que un evento del patrón se reporte y sea un falso positivo a que no sea reportado. Por lo tanto, el mejor modelo fue el tiny 3l auto retrained, que posee un Recall de 0.91 y una Precision de 0.59. Una observación sobre los tests es que uno de los falsos positivos del segmento Bulevar Artigas y Rodó de 18:34:03 a 18:40:03 se da por la asignación de los semáforos, el semáforo que habilita el avance de los vehículos en un grupo carril no es detectable con precisión por la cámara por lo que se decidió asignar otro semáforo coincide mayoritariamente con el avance de los vehículos del grupo carril, pero en el falso positivo mencionado, el semáforo asignado está en verde cuando el vehículo en realidad no está habilitado para avanzar. La figura 4.2 ilustra el caso mencionado.

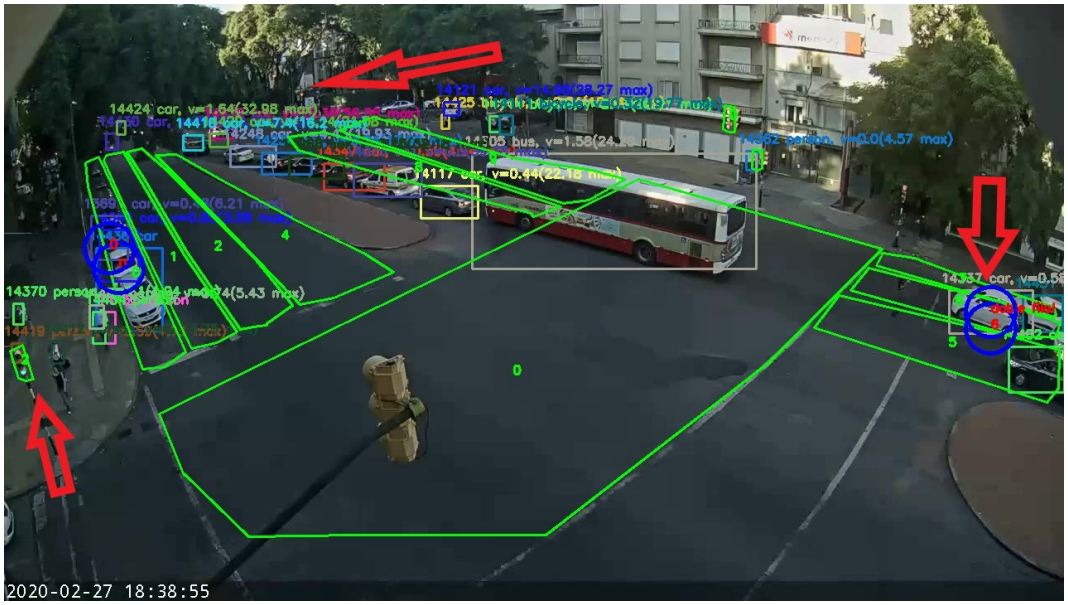

Figura 4.2: Falso positivo del patrón estacionamiento en doble fila. El vehículo señalado en el borde derecho es detectado como estacionado en doble fila debido a que tiene asignado el semáforo señalado en el borde izquierdo, este semáforo se encuentra en verde, mientras que el semáforo que en realidad le habilita el paso, el cual está señalado al fondo de la imagen, se encuentra en rojo.

Capítulo 5

Conclusiones y trabajo futuro

En este capítulo se sintetizan las conclusiones que se desprenden del estudio realizado junto con los resultados obtenidos en la evaluación experimental. Se agrega también posibles mejoras o extensiones posibles en forma de trabajo futuro que permitan a otros investigadores extender el trabajo presentado.

5.1. Conclusiones

Durante el desarrollo del proyecto se investigó sobre temas como detección de objetos (mayoritariamente los algoritmos YOLO), análisis de tránsito, rastreo de objetos (tracking) y recolección de datos. Se adquirieron conocimientos sobre como funcionan las redes neuronales convolucionales, más específicamente los detectores de objetos, junto con que métricas se utilizan para su evaluación y que parámetros interactúan en su funcionamiento (anchors, NMS). También fue necesario aprender sobre sistemas de seguimiento de objetos, como realizar mejora de datos para obtener mejores resultados al entrenar modelos y sobre como las redes generativas adversarias pueden ser útiles para la tarea de mejorar los datos. Los conocimientos adquiridos durante el proyecto no forman parte de los cursos de la carrera, si bien hay cursos relacionados a inteligencia artificial, los cursos no tienen tanto énfasis en elementos referentes a procesamiento de imágenes.

A lo largo del proyecto de grado se entrenaron varios modelos de detección de vehículos a partir de videos provistos por la Intendencia de Montevideo. Durante su evaluación el mayor mAP de vehículos obtenido fue de 35.58 %, mientras que el mejor mAP obtenido para vehículos y personas fue de 33.03 %

(en ambos casos referentes al mAP del conjunto de validación). Los detectores que obtuvieron mejores resultados fueron utilizados para la detección de eventos. Se encontró que los modelos tiny con dos capas no logran un nivel de detección suficientemente bueno, por lo que es necesario utilizar un modelo de tres capas. El uso de pseudo-labelling para incrementar el tamaño del conjunto de entrenamiento a partir de utilizar un modelo más complejo fue efectivo para mejorar los resultados. Durante la evaluación de los patrones, se observó que para el patrón de cruce en luz roja, se obtuvieron valores buenos de recall (0.63), si bien la precision no fue tan buena (0.42). Para el patrón de movimiento brusco, se obtuvieron buenos valores de precision y recall (0.82 en ambos casos), aunque fue necesario agregar pruebas sintéticas a las pruebas realizadas dada la dificultad para encontrar instancias del evento en los videos provistos. Para el patrón de estacionamiento en doble fila, se obtuvo un buen valor de recall (0.91) con un nivel aceptable de precision (0.59). A partir de los resultados obtenidos en la evaluación experimental, se puede ver que es posible la creación de un sistema de detección de conducción temeraria a partir de video en tiempo real aplicado a la zona metropolitana de Montevideo. Fue posible generar una solución que no solo obtenga resultados en tiempo real sino que también sea de buena calidad, permitiendo asistir en el análisis de la seguridad vial con asistencia computacional. El sistema generado podría integrarse al sistema de gestión inteligente del sistema de movilidad o utilizarse en el análisis particular de videos de tránsito con cámaras.

Si bien el sistema puede ser integrado sin necesidad de una gran cantidad de cambios, existen algunas limitaciones que deben ser trabajadas para poder escalar la solución presentada a grandes cantidades de cámaras en simultaneo. Algunos elementos que limitan parcialmente la escala de la solución es la necesidad de generar datos etiquetados para los carriles de cada cámara. Es necesario también para cada cámara generar datos de mapeo de información geográfica que permitan calibrar el detector de velocidad para cada caso. A nivel de cómputo, si bien es posible procesar el video en tiempo real, en caso de escalar el trabajo a más cámaras en simultáneo es necesario contar con mayor cantidad de recursos de GPU.

5.2. Trabajo futuro

Una posible mejora sería la utilización de modelos más nuevos de la familia YOLO (v4, v5). Las versiones de YOLO v4 y v5 logran mejores resultados tanto en mAP como en velocidad de cómputo que YOLOv3, permitiendo ejecutar todos los patrones en tiempo real en un única computadora o con diferentes cámaras a la vez.

El procesamiento hilado permitiría alcanzar mejoras en la performance en FPS de los detectores, permitiendo tener un hilo de ejecución por cada etapa del proceso, evitando terminar una etapa de tracking o detección de eventos para obtener las detecciones del frame siguiente. El uso de hilos de ejecución podría ser acompañado de inferencia por batch de imágenes a la hora de detectar los vehículos de interés. El procesamiento en batch sería de interés no solo en el caso particular de una sola cámara sino que también en el caso en que se busque aplicar los algoritmos a varias cámaras en simultáneo, mejorando la eficiencia de recursos del sistema.

Para tener una detección más fidedigna de los carriles de tráfico fue necesario etiquetar a mano los carriles de cada cámara. El proceso de etiquetado de carriles podría automatizarse, disminuyendo el tiempo de calibración por cámara, esto presenta ciertas dificultades pues las líneas delimitadoras frecuentemente son poco visibles por el desgaste de la pintura. El tiempo de calibración está constituido por el tiempo de definición de carriles, semáforos, cruces y mapeo de puntos de la pantalla a coordenadas geográficas.

Referencias bibliográficas

AlexeyAB (2017). Yolo-v3 and yolo-v2 for windows and linux. https://github.com/AlexeyAB/darknet. Último acceso en 19/11/2020.

AlexeyAB (2019). Improve object detection - tiny version. https://github.com/AlexeyAB/darknet/issues/3497. Último acceso en 10/07/2020.

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

Auto-Data.net (2010). Datos de tamaño de autos. https://www.auto-data.net/es/. Último acceso en 17/04/2020.

Behrendt, K. & Novak, L. (2017). A deep learning approach to traffic lights: Detection, tracking, and classification. En *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE.

Blake, S. (2019). How to track objects in the real world with tensorflow, sort and opencv. https://medium.com/hal24k-techblog/how-to-track-objects-in-the-real-world-with-tensorflow-sort-and-opencv-a64d9564ccb1. https://medium.com/hal24k-techblog/how-to-track-objects-in-the-real-world-with-tensorflow-sort-and-opencv-a64d9564ccb1. Último acceso en 6/7/2020.

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

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection.

Bolme, D. S., Beveridge, J. R., Draper, B. A., & Lui, Y. M. (2010). Visual object tracking using adaptive correlation filters. En *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2544–2550.

Brownlee, J. (2019). How to Perform Object Detection With YOLOv3 in Keras. https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/. https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/. Último acceso en 1/10/2019.

Chen, K., Shou, T. D., Li, J. K., & Tsai, C. (2018). Vehicles detection on expressway via deep learning: Single shot multibox object detector. En *2018 International Conference on Machine Learning and Cybernetics (ICMLC)*, volumen 2, pp. 467–473.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338.

Fan, Z., Jing, B., Xiaoyu, L., Changxing, P., & Havyarimana, V. (2019). An ensemble cascading extremely randomized trees framework for short-term traffic flow prediction. *KSII Transactions on Internet & Information Systems*, 13(4):1975 – 1988.

Fregin, A., Muller, J., Krebel, U., & Dietmayer, K. (2018). The driveu traffic light dataset: Introduction and comparison with existing datasets. En *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3376–3383.

Gauen, K., Dailey, R., Laiman, J., Zi, Y., Asokan, N., Lu, Y., Thiruvathukal, G. K., Shyu, M., & Chen, S. (2017). Comparison of visual datasets for machine learning. En *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, pp. 346–355.

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

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. En *2014*

IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587.

Google (2020). Overview of open images v5. https://storage.googleapis.com/openimages/web/factsfigures_v5.html. https://storage.googleapis.com/openimages/web/factsfigures_v5.html. Último acceso en 5/2/2020.

Haque, W. A., Arefin, S., Shihavuddin, A., & Hasan, M. A. (2021). Deepthin: A novel lightweight cnn architecture for traffic sign recognition without gpu requirements. *Expert Systems with Applications*, 168:114481.

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

Henderson, P. & Ferrari, V. (2016). End-to-end training of object class detectors for mean average precision.

Henriques, J. F., Caseiro, R., Martins, P., & Batista, J. (2015). High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596.

Huang, R., Pedoeem, J., & Chen, C. (2018). Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers. En *2018 IEEE International Conference on Big Data (Big Data)*, pp. 2503–2510.

hyun Lee, D. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks.

Jocher, G., guigarfr, perry0418, Ttayu, Veitch-Michaelis, J., Bianconi, G., Baltacı, F., Suess, D., & WannaSeaU (2019). ultralytics/yolov3: Video Inference, Transfer Learning Improvements. https://github.com/ultralytics/yolov3.

Kim, C., Lee, J., Han, T., & Kim, Y.-M. (2018). A hybrid framework combining background subtraction and deep neural networks for rapid person detection. *Journal of Big Data*, 5(1):22.

Kristan, M., Matas, J., Leonardis, A., Felsberg, M., Cehovin, L., Fernandez, G., Vojir, T., Hager, G., Nebehay, G., Pflugfelder, R., Gupta, A., Bibi, A.,

Lukezic, A., Garcia-Martin, A., Saffari, A., Petrosino, A., & Solis Montero, A. (2015). The visual object tracking vot2015 challenge results. En *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pp. 564–586.

Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Duerig, T., & Ferrari, V. (2018). The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv:1811.00982*.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. En Fleet, D., Pajdla, T., Schiele, B., & Tuytelaars, T., editores, *Computer Vision – ECCV 2014*, pp. 740–755, Cham. Springer International Publishing.

Lin, Y., Dai, X., Li, L., & Wang, F. (2019). Pattern sensitive prediction of traffic flow based on generative adversarial framework. *IEEE Transactions on Intelligent Transportation Systems*, 20(6):2395–2400.

Liu, C., Tao, Y., Liang, J., Li, K., & Chen, Y. (2018). Object detection based on yolo network. En *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pp. 799–803.

Liu, M.-Y., Breuel, T., & Kautz, J. (2018). Unsupervised image-to-image translation networks.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. En Leibe, B., Matas, J., Sebe, N., & Welling, M., editores, *Computer Vision – ECCV 2016*, pp. 21–37, Cham. Springer International Publishing.

Lucas, B. D. & Kanade, T. (1981). Iterative image registration technique with an application to stereo vision. volumen 2, pp. 674–679. Cited By :5935.

Lukežič, A., Vojíř, T., Čehovin Zajc, L., Matas, J., & Kristan, M. (2018). Discriminative correlation filter tracker with channel and spatial reliability. *International Journal of Computer Vision*, 126(7):671 – 688.

Mou, L., Zhao, P., Xie, H., & Chen, Y. (2019). T-lstm: A long short-term memory neural network enhanced by temporal information for traffic flow prediction. *IEEE Access*, 7:98053–98060.

Nesmachnow, S. e Iturriaga, S. (2019). Cluster-uy: Collaborative scientific high performance computing in uruguay. En Torres, M. & Klapp, J., editores, *Supercomputing*, pp. 188–202, Cham. Springer International Publishing.

Ooi, H.-L., Bilodeau, G.-A., Saunier, N., & Beaupré, D.-A. (2018). Multiple object tracking in urban traffic scenes with a multiclass object detector. En Bebis, G., Boyle, R., Parvin, B., Koracin, D., Turek, M., Ramalingam, S., Xu, K., Lin, S., Alsallakh, B., Yang, J., Cuervo, E., & Ventura, J., editores, *Advances in Visual Computing*, pp. 727–736, Cham. Springer International Publishing.

Padilla, R., Netto, S. L., & da Silva, E. A. B. (2020). A survey on performance metrics for object-detection algorithms. En *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 237–242.

Pengyi Zhang, Yunxin Zhong, X. L. (2019). Slimyolov3: Narrower, faster and better for real-time uav applications. *CoRR*, abs/1907.11093.

Plotka, S. (2018). Cifar-10 classification using keras tutorial. https://ermlab.com/en/blog/nlp/cifar-10-classification-using-keras-tutorial/. Último acceso en 10/12/2020.

Ponnusamy, A. (2018). YOLO Object Detection with OpenCV and Python. https://towardsdatascience.com/yolo-object-detection-with-opencv-and-python-21e50ac599e9. Último acceso en 12/12/2019.

Priambodo, B. & Ahmad, A. (2018). Traffic flow prediction model based on neighbouring roads using neural network and multiple regression. *Journal of Information & Communication Technology*, 17(4):513 – 535.

Rabbouch, H., Saâdaoui, F., & Mraihi, R. (2018). A vision-based statistical methodology for automatically modeling continuous urban traffic flows. *Advanced Engineering Informatics*, 38:392 – 403.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. En *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Redmon, J. & Farhadi, A. (2017). Yolo9000: Better, faster, stronger. En *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525.

Redmon, J. & Farhadi, A. (2018). Yolov3: An incremental improvement. *CoRR*, abs/1804.02767.

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149.

Reyes, E., Gómez, C., Norambuena, E., & Ruiz-del Solar, J. (2018). Near real-time object recognition for pepper based on deep neural networks running on a backpack.

Subrayado (2019). Suman ocho nuevos puntos de control con cámaras para fiscalizar exceso de velocidad. *Subrayado*. https://www.subrayado.com.uy/suman-ocho-nuevos-puntos-control-camaras-fiscalizar-exceso-velocidad-n538448. Último acceso en 15/4/2021.

Temel, D., Alshawi, T., Chen, M., & AlRegib, G. (2019). Challenging environments for traffic sign detection: Reliability assessment under inclement conditions. *CoRR*, abs/1902.06857.

Vittorio, A. (2018). Toolkit to download and visualize single or multiple classes from the huge open images v4 dataset. https://github.com/EscVM/OIDv4_ToolKit. Último acceso en 19/12/2019.

Ward, J., Lukowicz, P., & Gellersen, H. (2011). Performance metrics for activity recognition. *ACM TIST*, 2:6.

Xiaochus, J Redmon, A. F. (2018). Yolov3: An incremental improvement. Último acceso en 10/12/2019.

Yu, L., Zhao, J., Gao, Y., & Lin, W. (2019). Short-term traffic flow prediction based on deep learning network. En *2019 International Conference on Robots Intelligent System (ICRIS)*, pp. 466–469.

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. En *Computer Vision (ICCV), 2017 IEEE International Conference on*.

Zhu, P., Wen, L., Bian, X., Ling, H., & Hu, Q. (2018). Vision meets drones: A challenge.

Glosario

En esta sección se incluya una lista de términos y siglas, junto con una explicación sucinta de cada uno.

AP La precisión promedio (AP) es una forma de resumir la curva de precisión-recall en un solo valor que representa el promedio de todas las precisiones. Es una métrica altamente utilizada para evaluar detectores de objetos en la actualidad.

ARMA En estadística, los modelos auto-regresivos de media móvil (ARMA), se aplican a series temporales de datos. Dada una serie temporal de datos Xt, el modelo ARMA es una herramienta para entender y, aún más, para predecir futuros valores de la serie.

ASEF Average of Synthetic Exact Filters. El algoritmo ASEF reduce la amplitud de los picos de cualquier tipo de señal.

AUC Término matemático referente a Area Under the Curve o área bajo la curva.

Anchors Los anchors son un conjunto de bounding boxes predefinidos de cierta altura y ancho. Estos cuadros se definen para capturar la escala y la relación de aspecto de clases de objetos específicas que desea detectar y por lo general, se eligen en función del tamaño de los objetos en sus conjuntos de datos de entrenamiento.

Autoencoder Autoencoder es un modelo de red neuronal que aprende a codificar una representación de la entrada para luego reproducirla en su salida lo más similar posible. El autoencoder consta de dos partes, por un lado el encoder que toma la entrada y la codifica en lo que se denomina un espacio latente y el decoder que toma un elemento del espacio latente y lo convierte en una reconstruccción de la entrada.

BFLOPS Un FLOP es un acrónimo de operaciones de coma flotante por segundo. Las operaciones de punto flotante incluyen números irracionales

y puntos decimales, y son mucho más complejas que las operaciones de punto fijo, que se limitan a enteros binarios. Debido a la dificultad inherente de las operaciones de punto flotante, se utilizan para medir la potencia de cálculo de un sistema en particular. Un BFLOP representa un billón de FLOPs.

Background sustraction Background sustraction es un método popular para aislar las partes móviles de una escena segmentando el fondo y el primer plano. OpenCV posee una implementación disponible del método.

Batch Normalization La normalización por lotes o Batch Normalization (también conocida como norma por lotes) es un método que se utiliza para hacer que las redes neuronales artificiales sean más rápidas y estables mediante la normalización de la capa de entrada al centrar y re-escalar sus valores.

Blob En la visión por computadora, los métodos de detección de blobs tienen como objetivo detectar regiones en una imagen digital que difieren en propiedades, como brillo o color, en comparación con las regiones circundantes. Un blob refiere a una región de una imagen en la que algunas propiedades son constantes o aproximadamente constantes. Todos los puntos de un blob se pueden considerar en cierto sentido similares entre sí.

Bounding Box Los cuadros delimitadores o bounding boxes son representaciones geométricas muy simplificadas de objetos 3D que se suelen utilizar como formato intermedio para intercambiar información.

CCD El término CCD es conocido popularmente como la designación de uno de los elementos principales de las cámaras fotográficas y de video digitales. En estas, el CCD es el sensor con diminutas células fotoeléctricas que registran la imagen. Desde allí la imagen es procesada por la cámara y registrada en la tarjeta de memoria.

CCTV El circuito cerrado de televisión o CCTV es una tecnología de videovigilancia diseñada para supervisar una diversidad de ambientes y actividades. Se le denomina circuito cerrado ya que, al contrario de lo que pasa con la difusión, todos sus componentes están enlazados. Además, a diferencia de la televisión convencional, este es un sistema pensado para un número limitado de espectadores.

CLAHE La ecualización adaptativa de histograma (AHE) es una técnica de procesamiento de imágenes por computadora que se utiliza para mejorar el contraste en las imágenes. Se diferencia de la ecualización de histogramas ordinaria en que el método adaptativo calcula varios histogramas, cada uno correspondiente a una sección distinta de la imagen, y los utiliza para redistribuir los valores de luminosidad de la imagen. Por lo tanto, es adecuado para mejorar el contraste local y mejorar las definiciones de los bordes en cada región de una imagen.

CLEAR MOT Referente a las métricas MOTA y MOTP en el dataset CLEAR.

CNN Una red neuronal convolucional (CNN) es una clase de redes neuronales profundas, que se aplica más comúnmente al análisis de imágenes visuales. También se conocen como redes neuronales artificiales invariantes en el desplazamiento o invariantes en el espacio (SIANN), basadas en la arquitectura de peso compartido de los núcleos de convolución que escanean las capas ocultas y las características de invariancia de traducción. Tienen aplicaciones en reconocimiento de imagen y video, sistemas de recomendación, clasificación de imágenes, segmentación de imágenes, análisis de imágenes médicas.

COCO COCO es un conjunto de datos de detección, segmentación y subtítulos de objetos a gran escala. COCO tiene varios elementos: segmentación de objetos, reconocimiento en contexto, entre otras.

CPU La unidad central de procesamiento (Central Processing Unit), es el hardware dentro de un ordenador u otros dispositivos programables, su trabajo es interpretar las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y externas (provenientes de la unidad de entrada/salida).

CSRT El tracker CSRT es una implementación en C del algoritmo de tracking CSR-DCF (Channel and Spatial Reliability of Discriminative Correlation Filter) en la biblioteca OpenCV.

CVAT CVAT es una herramienta gratuita, en línea e interactiva de anotación de imágenes y videos para visión por computadora.

DPM El modelo de piezas deformables (DPM) ha surgido recientemente como una herramienta muy útil y popular para abordar el problema de diversidad dentro de la categoría en la detección de objetos.

EA Función mediante la cual la cámara selecciona de forma automática la apertura y velocidad de obturación, para obtener una correcta iluminación.

EET Algoritmo utilizado para predecir flujo de tráfico de vehículos en condiciones no estacionarias.

EM En el modo de exposición manual (EM), un fotógrafo debe ajustar la apertura y/o la velocidad para lograr la exposición deseada.

Espacio latente Un espacio latente es una representación de datos comprimidos en los que puntos de datos similares están más juntos en el espacio. El espacio latente es útil para aprender las características de los datos y para encontrar representaciones de datos más simples para su análisis.

FN A false negative error (FN), or false negative, is a test result which wrongly indicates that a condition does not hold.

FP A false positive error, or false positive, is a result that indicates a given condition exists when it does not.

FPN Feature Pyramid Network (FPN) es un extractor de características diseñado con el objetivo de mantener buen precisión y velocidad. Genera múltiples capas de mapas de características (mapas de características de múltiples escalas) con información de mejor calidad que la pirámide de características normal para la detección de objetos.

FPS FPS representa "cuadros por segundo". FPS se utiliza para medir la velocidad de fotogramas: el número de imágenes consecutivas a pantalla completa que se muestran cada segundo. Es una especificación común utilizada en la captura y reproducción de video.

GAN Una red generativa adversarial (GAN) es un modelo de aprendizaje automático en el que dos redes neuronales compiten entre sí para ser más precisas en sus predicciones. Las GAN generalmente se ejecutan sin supervisión y utilizan un marco de juego cooperativo para aprender.

GHZ Giga HertZ representa mil millones de ciclos por segundo. Las computadoras de alta velocidad tienen relojes internos clasificados en GHz y las aplicaciones de radiofrecuencia transmiten en este rango.

GMM Algoritmo de clustering que no solo utiliza la media sino que también la varianza de los datos para asignar los centroides.

GPU Una unidad de procesamiento gráfico o GPU (graphics processing unit) es un coprocesador dedicado al procesamiento de gráficos u operaciones

de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos o aplicaciones 3D interactivas.

GRU Las unidades recurrentes cerradas (GRU) son un mecanismo de compuerta en redes neuronales recurrentes. GRU es como una memoria a corto plazo (LSTM) con una puerta de olvido, pero tiene menos parámetros que LSTM, ya que carece de una puerta de salida.

GTX GTX significa Giga Texel Shader eXtreme y es una variante de la marca GeForce propiedad de Nvidia. Se introdujeron por primera vez en 2008 con la serie 200, con nombre en código Tesla.

Ground truth Ground truth es un término utilizado en varios campos para referirse a la información proporcionada por observación directa (es decir, evidencia empírica) en contraposición a la información proporcionada por inferencia.

HOG El histograma de gradientes orientados (HOG) es un descriptor de características que se utiliza en la visión por computadora y el procesamiento de imágenes con el propósito de detectar objetos. La técnica cuenta las ocurrencias de orientación de los gradientes en partes localizadas de una imagen.

HSV Modelo que representa un color como una tupla de tres componentes, tono (H), saturación (S), y valor (V), con el fin de obtener una representación más perceptual que el modelo RGB.

IOT Internet de las cosas (IoT) describe la red de objetos físicos que están integrados con sensores, software y otras tecnologías con el fin de conectar e intercambiar datos con otros dispositivos y sistemas a través de Internet.

IOU La intersección sobre unión (IoU) es una buena métrica para medir la superposición entre dos cuadros delimitadores o máscaras. Se usa comúnmente en tareas de visión por computadora para evaluar si una predicción es correcta o no. Se selecciona un valor mínimo de superposición para tomar la predicción como verdadera o falsa.

IP Significa "Protocolo de Internet". IP proporciona un conjunto estándar de reglas para enviar y recibir datos a través de Internet. Permite que los dispositivos que se ejecutan en diferentes plataformas se comuniquen entre sí siempre que estén conectados a Internet.

K-Means K-means es un método de clustering, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano.

KNN Es un enfoque para la clasificación de datos que estima la probabilidad de que un punto de datos sea miembro de un grupo u otro, dependiendo del grupo en el que se encuentren los puntos de datos más cercanos. Es un ejemplo de un algoritmo de "aprendiz perezoso", lo que significa que no construye un modelo utilizando el conjunto de entrenamiento hasta que se realiza una consulta del conjunto de datos.

Keras Keras es una librería de inteligencia artificial creada orientada a seres humanos, no para máquinas. Keras sigue las mejores prácticas para reducir la carga cognitiva: ofrece API consistentes y simples, minimiza la cantidad de acciones del usuario requeridas para casos de uso comunes y proporciona mensajes de error claros y procesables. También cuenta con una extensa documentación y guías para desarrolladores.

Kernel Kernel es una función que toma dos vectores en un espacio original y devuelve el producto punto de estos vectores luego de aplicar una transformación sobre los mismos, esta transformación puede ser no lineal. El llamado *kernel trick* consiste en escribir un problema de optimización en función de estos productos punto sin aplicar la transformación directamente, estos productos punto frecuentemente son más fáciles de encontrar que la transformación sobre los vectores.

LSTM LSTM (Long Short Term Memory) es una arquitectura de red neuronal recurrente artificial (RNN) utilizada en el campo del aprendizaje profundo. A diferencia de las redes neuronales de retroalimentación estándar, LSTM tiene conexiones de retroalimentación. No solo puede procesar puntos de datos individuales (como imágenes), sino también secuencias completas de datos (como voz o video). Por ejemplo, LSTM es aplicable a tareas como el reconocimiento de escritura a mano conectada y no segmentada, reconocimiento de voz y detección de anomalías en el tráfico de red o IDS (sistemas de detección de intrusos).

LabelBox Labelbox es una plataforma de etiquetado de datos que permite colaboración entre varios colaboradores. Permite etiquetado de distintos tipos de datos, desde detección de objetos a entidades con nombre en texto.

Leaky Relu Leaky ReLU es un tipo de función de activación. Se sugiere que es una mejora del ReLU tradicional.

Loss Loss es el castigo o costo por una mala predicción. Es decir, el loss es un número que indica qué tan mala fue la predicción del modelo en un solo ejemplo.

MAE En estadística, el error absoluto medio (mean average error) es una medida de la diferencia entre dos variables continuas. Considerando dos series de datos (unos calculados y otros observados) relativos a un mismo fenómeno, el error absoluto medio sirve para cuantificar la precisión de una técnica de predicción comparando por ejemplo los valores predichos frente a los observados, el tiempo real frente al tiempo previsto, o una técnica de medición frente a otra técnica alternativa de medición.

MAPE El error porcentual absoluto medio o mean absolute percentage error (MAPE), también conocido como desviación porcentual absoluta media (MAPD), es una medida de la precisión de la predicción de un método de predicción en estadística, por ejemplo en la estimación de tendencias, que también se utiliza como función de pérdida para problemas de regresión en máquinas.

MIO-CTD MIOvision Traffic Camera Dataset (MIO-CTD), es un dataset generado para un desafío de localización y identificación de vehículos en cámaras de tránsito.

MOG Es un algoritmo de que permite distinguir el fondo de un video (background substraction) utilizando gaussian mixture. Fue introducido en el documento "Un modelo de mezcla de fondo adaptativo mejorado para seguimiento en tiempo real con detección de sombras" por P. KadewTra-KuPong y R. Bowden en 2001. Es un algoritmo que existe como librería en Opencv.

MOSSE MOSSE o Minimum Output Sum of Squared Error es un algoritmo de tracking que permite obtener velocidades de tracking muy altas (700fps). El algoritmo es robusto a cambios de luz, escala, pose y algunas deformaciones.

MOT Sigla referente a la tarea de multiple object tracking o tracking de múltiples objetos.

MOTA MOTA (multiple object tracking accuracy) es una métrica utilizada en el tracking de objetos.

MOTP MOTP (multiple object tracking precision) es una métrica utilizada en el tracking de objetos. MOTP es una medida de la superposición del bounding boxes.

NMS Non Max Supression (NMS) es una técnica utilizada en muchos algoritmos de visión por computadora. Es una clase de algoritmos para seleccionar una entidad (por ejemplo, cuadros delimitadores) entre muchas entidades superpuestas.

OBB Oriented Bounding Box o cuadrado mínimo delimitador orientado refiere a un cuadrado mínimo delimitador que se encuentre rotado, fuera de su posición habitual, donde se encuentra alineado con los ejes x,y.

OPENCV OpenCV (Open Source Computer Vision Library) es una biblioteca de funciones de programación dirigida principalmente a la visión por computadora en tiempo real. Desarrollado originalmente por Intel.

Objectness La objetividad o objectness es esencialmente una medida de la probabilidad de que un objeto exista en una región de interés propuesta. Si tenemos una objetividad alta, esto significa que la ventana de la imagen probablemente contiene un objeto.

Open Images Open Images es un conjunto de datos de Google con cerca de 9 millones de imágenes variadas con anotaciones. Las imágenes son muy diversas y suelen contener escenas complejas con varios objetos (8,4 por imagen de media). Contiene anotaciones de etiquetas a nivel de imagen, cuadros delimitadores de objetos, segmentaciones de objetos, relaciones visuales, narrativas localizadas y más.

PR-MOTA Se define la curva PR-MOTA como la curva en 3D que caracteriza la relación entre el rendimiento de detección de objetos (precisión y recuperación) y el rendimiento de tracking the multiples objetos (MOTA).

PSN Tipo de red neuronal que utiliza redes generativas adversarias para generar su predicción.

Pooling La operación de pooling representa disminuir la resolución de una imagen. El filtro de pooling más común es de tamaño 2x2, que descarta tres cuartas partes de las activaciones. Se toma un tamaño de ventana nxn y para cada uno, se toma el valor máximo dentro de esa ventana.

Pytorch PyTorch es una biblioteca de aprendizaje automático de código abierto basada en la biblioteca de Torch, utilizado para aplicaciones que

implementan cosas como visión artificial y procesamiento de lenguajes natural.

RCNN Region Based Convolutional Neural Networks (RCNN) o redes convolucionales basadas en regiones son redes neuronales donde el output de la red son bounding boxes delimitando objetos junto con su clase.

RGB Modelo que representa un color descomponiéndolo en componentes rojo (R), verde(G) y azul (G) que al combinarse de forma aditiva, como en una pantalla de ordenador, reproducen el color.

RMSE El error cuadrático medio o root mean square error (RMSE) es una medida de uso frecuente de las diferencias entre los valores (valores de muestra o población) predichos por un modelo o estimador y los valores observados.

RNN Una red neuronal recurrente (RNN) es una clase de redes neuronales artificiales donde las conexiones entre nodos forman un gráfico dirigido a lo largo de una secuencia temporal. Esto le permite exhibir un comportamiento dinámico temporal. Derivado de redes neuronales de alimentación directa, los RNN pueden usar su estado interno (memoria) para procesar secuencias de entradas de longitud variable.

Relu ReLu es una función de activación no lineal que se utiliza en redes neuronales multi capa o redes neuronales profundas.

SORT Es una implementación de un algoritmo de seguimiento visual de múltiples objetos basado en técnicas rudimentarias de asociación de datos y estimación de estado. Está diseñado para aplicaciones de rastreo online donde solo están disponibles los frames pasados y actuales y el método produce identidades de objetos sobre la marcha. Si bien este rastreador no maneja la oclusión o el reingreso de objetos, su propósito es servir como línea de base para el desarrollo de futuros trackers.

SPP Spatial Pyramid Pooling (SPP) es una capa de agrupación que elimina la restricción de tamaño fijo de la red, es decir, para que una CNN no requiera una imagen de entrada de tamaño fijo. Se agrega una capa SPP encima de la última capa convolucional. La capa SPP agrupa las características y genera salidas de longitud fija, que luego se alimentan a las capas completamente conectadas (u otros clasificadores). En otras palabras, realizamos cierta agregación de información en una etapa más profunda de la jerarquía de la red (entre capas convolucionales y ca-

pas completamente conectadas) para evitar la necesidad de recortar o deformar al principio.

SSD SSD es un detector de disparo único o single shot detector. No tiene una red de propuesta de regiones previa y predice los bounding boxes y las clases directamente desde los mapas de características en una sola pasada. Para mejorar la precisión, SSD introduce: pequeños filtros convolucionales para predecir clases de objetos.

SVM Las máquinas de vectores de soporte o Support Vector Machine (SVM) son modelos de aprendizaje supervisado con algoritmos de aprendizaje asociados que analizan datos para clasificación y análisis de regresión. Dado un conjunto de ejemplos de entrenamiento, cada uno marcado como perteneciente a una de dos categorías, un algoritmo de entrenamiento de SVM construye un modelo que asigna nuevos ejemplos a una categoría u otra, convirtiéndolo en un clasificador lineal binario no probabilístico.

SVR Support Vector Regression (SVR) utiliza el mismo principio que SVM pero para problemas de regresión. El problema que el mismo resuelve es encontrar una función que aproxima un mapeo de un dominio de entrada a números reales.

Sigmoide Muchos procesos naturales y curvas de aprendizaje de sistemas incluyen una progresión temporal desde unos niveles bajos al inicio, hasta acercarse a un punto de transición transcurrido un cierto tiempo. La transición se produce en una región caracterizada por una fuerte aceleración intermedia. La función sigmoide permite describir esta evolución.

Softmax La función softmax es una función que convierte un vector de K valores reales en un vector de K valores reales que suman 1 (los transforma en una probabilidad).

Stride Stride es un componente de las redes neuronales convolucionales. Stride es un parámetro del filtro de la red neuronal que modifica la cantidad de movimiento sobre la imagen. Por ejemplo, si el stride de una red neuronal se establece en 1, el filtro se moverá un píxel o una unidad a la vez.

TN Verdadero negativo o true negative (TN) es un resultado en el que el modelo predice correctamente la clase negativa.

TP Verdadero positivo o true positive (TP) es un resultado en el que el modelo predice correctamente la clase positiva.

Tensorflow TensorFlow es una biblioteca de código abierto para aprendizaje automático desarrollado por Google para construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.

Theano Theano es una biblioteca de Python y un compilador de optimización para manipular y evaluar expresiones matemáticas, especialmente aquellas con valores matriciales.

Upsampling El muestreo superior es el aumento de la resolución espacial mientras se mantiene la representación 2D de una imagen.

VAE Variational Autoencoder (VAE) es un modelo de red neuronal del tipo Autoencoder propuesto por Diederik P. Kingma y Max Welling que aprende a codificar una representación de la entrada para luego reproducirla en su salida mediante el uso de un espacio latente probabilístico, el cual se puede considerar representado por una distribución Gaussiana Multivariada.

VGG VGG es un modelo de red neuronal convolucional propuesto por K. Simonyan y A. Zisserman de la Universidad de Oxford en el artículo "Redes convolucionales muy profundas para el reconocimiento de imágenes a gran escala".

VOC El desafío Pascal VOC es un conjunto de datos muy popular para construir y evaluar algoritmos para clasificación de imágenes, detección de objetos y segmentación.

VOT VOT es un desafío de tracking de objetos de manera visual. En el mismo se libera un dataset donde diversos participantes intentan superarse entre ellos en dicho dataset.

VisDrone El conjunto de datos VisDrone 2020 es recopilado por el equipo de AISKYEYE en el Laboratorio de Aprendizaje Automático y Minería de Datos de la Universidad de Tianjin, China. El conjunto de datos de referencia consta de 400 videoclips formados por 265,228 cuadros y 10,209 imágenes estáticas, capturadas por varias cámaras montadas en drones, que cubren una amplia gama de aspectos, incluida la ubicación (tomada de 14 ciudades diferentes separadas por miles de kilómetros en China), el medio ambiente (urbano y campestre), objetos (peatones, vehículos, bicicletas, etc.) y densidad (escenas dispersas y concurridas).

YOLO YOLO (You only look once) es un sistema de detección de objetos en tiempo real de última generación. Conocido por tener diversas versiones tanto para dispositivos móviles, como para sistemas edge.

mAP Mean Average Precision es una métrica utilizada para evaluar la tarea de detección de objetos. La métrica se basa en tomar el promedio de la average precision (AP) para cada clase considerada.

pseudo-labeling Refiere a obtener un dataset etiquetado utilizando un algoritmo más apto que el que se quiere entrenar para cierta tarea particular. Esto permite entrenar modelos más simples con etiquetas generadas con modelos más complejos de forma semi-supervisada.

ANEXOS

Anexo 1

Histogramas de tamaños de vehículos

En este anexo se presentan los histogramas de los tamaños de los vehículos de las clases 'van', 'bicycle', 'motorcycle' y 'truck'.

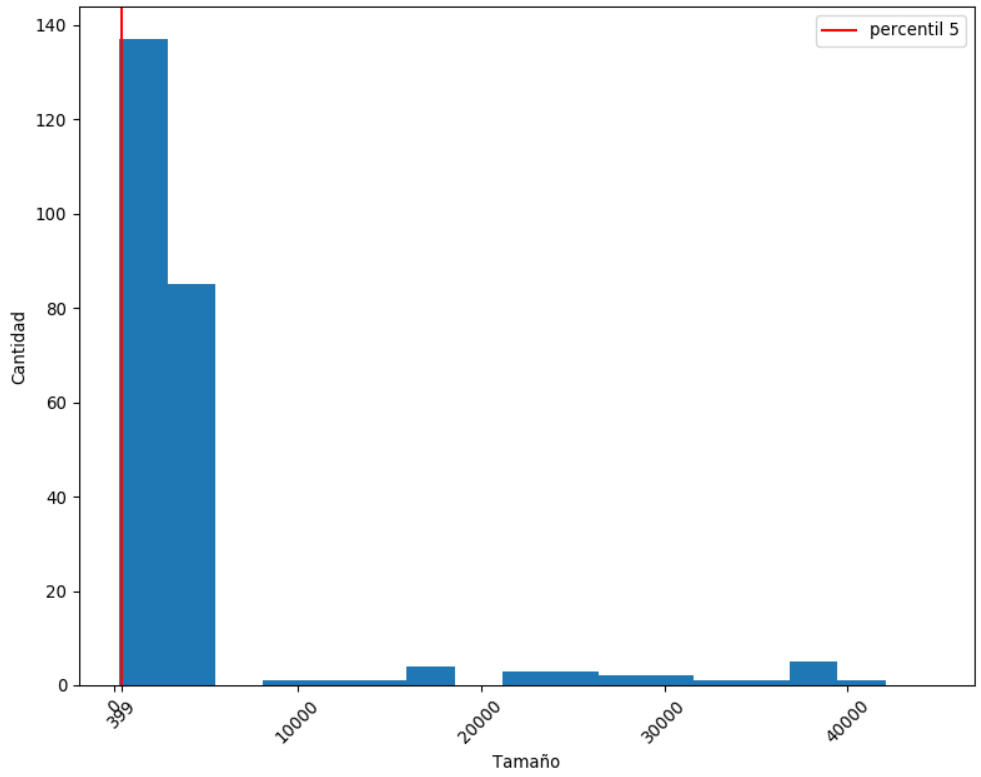

Figura 1.1: Histograma de tamaños de vehículos para la clase 'van'. Se indica el tamaño del percentil 5 de los tamaños, el cual se usa para determinar tamaños anómalamente pequeños.

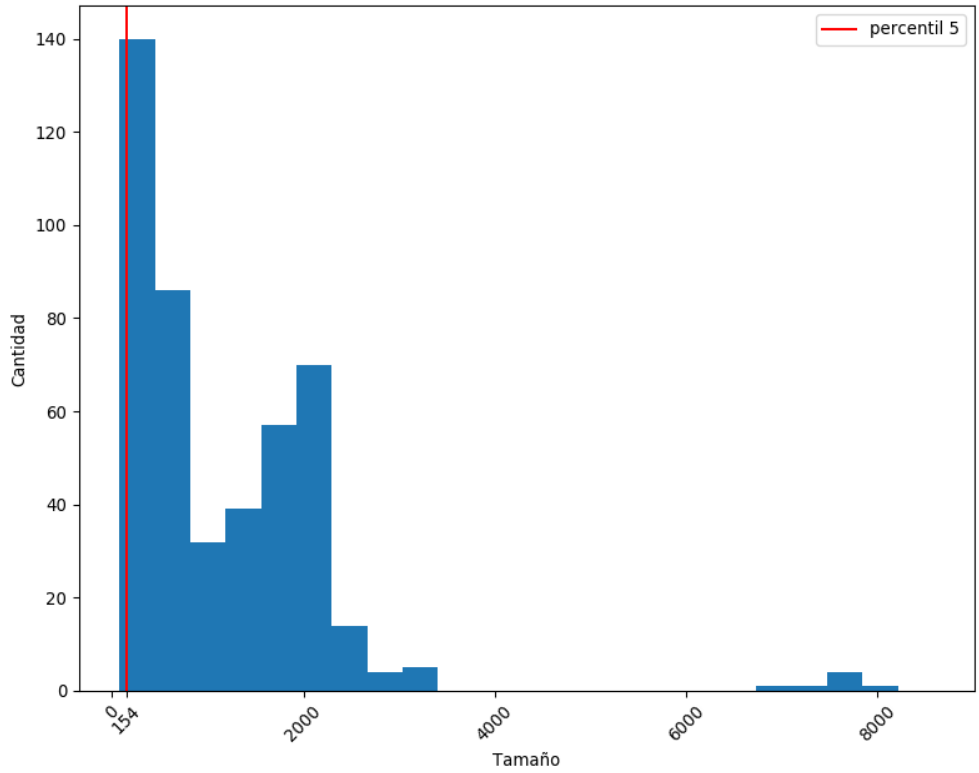

Figura 1.2: Histograma de tamaños de vehículos para la clase 'bicycle'. Se indica el tamaño del percentil 5 de los tamaños, el cual se usa para determinar tamaños anómalamente pequeños.

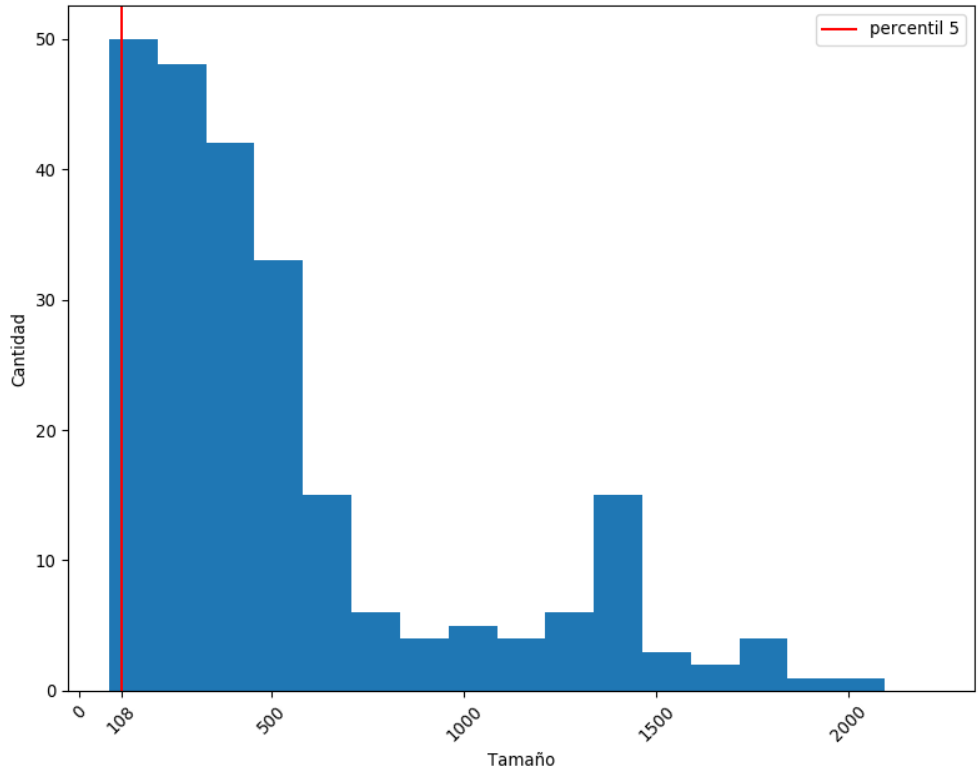

Figura 1.3: Histograma de tamaños de vehículos para la clase 'motorcycle'. Se indica el tamaño del percentil 5 de los tamaños, el cual se usa para determinar tamaños anómalamente pequeños.

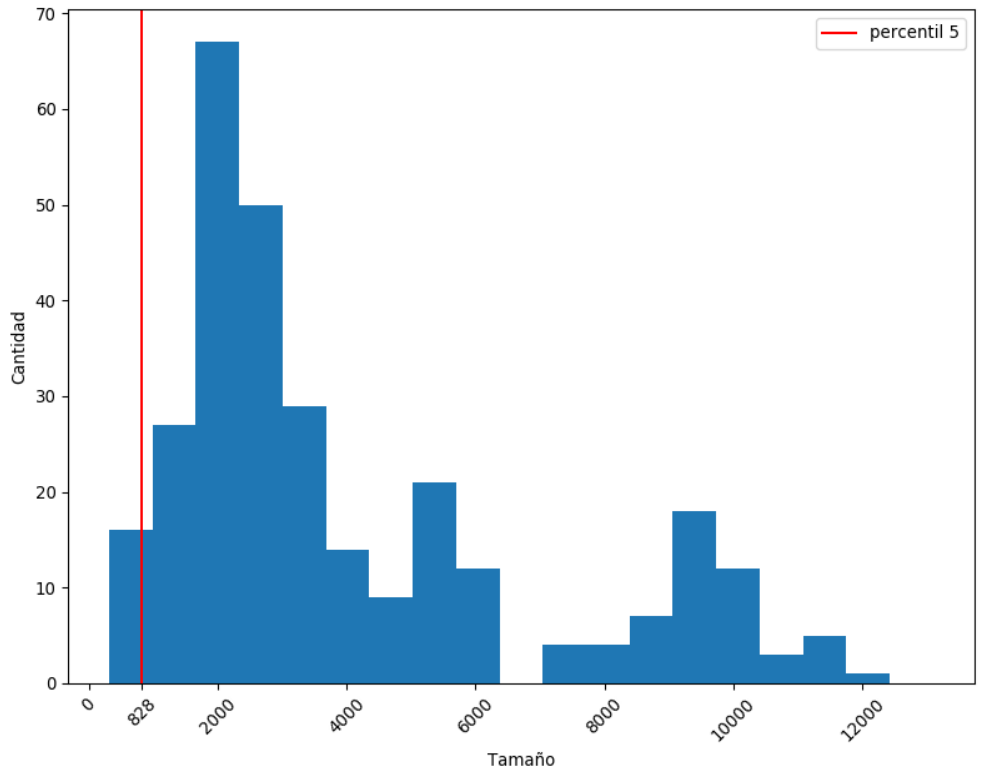

Figura 1.4: Histograma de tamaños de vehículos para la clase 'truck'. Se indica el tamaño del percentil 5 de los tamaños, el cual se usa para determinar tamaños anómalamente pequeños.