UNIVERSIDAD DE ANTIOQUIA

Proyecto de automatización de aforamiento vehicular.

Autor
Orion Montoya Correa

Universidad de Antioquia

Facultad de Ingeniería, Departamento de Ingeniería Electrónica y Telecomunicaciones

Medellin, Colombia

2020

Proyecto de automatización de aforamiento vehicular.

Orion Montoya Correa

Informe de práctica o monografía o investigación o tesis o trabajo de grado
como requisito para optar al título de:
Ingeniero Electrónico.

Asesores (a) o Director(a) o Co- Directores(a).

Asesor Interno: David Stephen Fernandez Mc Cann. Ingeniero Electrónico
Asesor Externo: Santiago Salgado Duque. Ingeniero Electrónico

Universidad de Antioquia
Facultad de Ingeniería, Departamento de Ingeniería Electrónica y Telecomunicaciones.
Medellín, Colombia
2020.

Resumen

En este informe se presenta la manera en al que se desarrolló, en mi tiempo de práctica, mi proyecto realizado en la empresa INTER-TELCO S.A.S, el cual nace de la necesidad de llevar a cabo aforos vehiculares de manera más automática. Se mostrará el desarrollo del proyecto, la problemática inicial de la empresa, los fundamentos teóricos de las herramientas utilizadas para el desarrollo del proyecto, los objetivos definidos, los resultados y la evaluación de estos y cuales han sido las ventajas de esta nueva manera de llevar a cabo los aforamientos. También se describe la siguiente etapa a la que la empresa llevará el proyecto llamado AforosDron.

Introducción

La alcaldía de Rionegro desde su secretaría de movilidad, tránsito y transporte busca mejorar el tráfico del municipio, apoyándose en su aliado tecnológico, la empresa Inter-Telco [1]. Ambas instituciones han desarrollado conjuntamente varios proyectos, tales como: Pasos seguros, Pantallas informativas, Totems peatonales, Sistema de Big Data Kratos, Sistemas de semaforización y Aforos vehiculares. Es en este último proyecto en el que se encuentra enmarcado mi semestre de industria. Actualmente para conseguir un conteo de vehículos en una intersección, existen tres maneras de afrontar el problema. La primera consiste en realizar tomas aéreas con un dron de las intersecciones de interés, para que con estas tomas, los empleados puedan realizar **visualmente** un conteo de vehículos, clasificándolos por tipo y por ruta tomada. La principal desventaja de esta estrategia de conteo radica en el tiempo que le lleva al personal la realización del aforo. El tiempo aproximado oscila entre dos y tres horas por toma aérea de entre 8 y 10 minutos. Adicionalmente, es evidente que no se pueden analizar múltiples videos en paralelo, ya que el número de videos que se puedan aforar en el mismo transcurso de tiempo - en paralelo - dependerá del número de personas disponibles en la empresa. Otro desafío es mejorar la fiabilidad, ya que cuando el aforo se realiza de esta manera solo se logra una fiabilidad entre el 80% y 85%.

Una segunda estrategia para realizar el aforo vehicular consiste en llevar el personal directamente a las intersecciones, normalmente mínimo 4 personas, cada una encargada de una ruta diferente. El conteo se hace de manera manual, llevando el conteo en libretas. Esta estrategia presenta las mismas dificultades expuesta en el primer caso, pero adicionalmente hace uso de un número mayor de personas para realizar el aforo, el número de personas dependerá del tipo de intersección, entre 4 y 8 personas. En este caso es muy complicado llegar a realizar múltiples a foros paralelamente, debido al alto número de personas necesarias.

La tercera estrategia, utilizada para realizar el aforo vehícular, utiliza una red de cámaras - una cámara para cada sentido vial - desplegadas en las ménsulas semafóricas que se encuentran en la intersección, realizando conteos únicamente unidireccionales, lo cual hace que solo se pueda saber la entrada o salida de un vehículo, pero no la ruta seguida por este. Por lo tanto, la información es sólo parcial y menor con relación a los dos casos anteriores. También es una dificultad para este sistema unos recursos de computación muy altos, ya que por cada cámara es necesario un PC compacto, para el análisis del video.

Así pues, se genera la necesidad de crear un sistema de aforamiento vehicular más autónomo, con el cual utilizando técnicas de procesamiento digital imagenes y tecnologías de deep learning, se analizen tomas aeres realizadas por un dron en la intersección de interés. Se busca que el sistema esté en la capacidad de poder obtener la información del tipo vehículo, ruta tomada por este y llevar un conteo de las diferentes rutas en la escena, todo esto de la manera más autónoma posible. Con este nuevo sistema se busca mejorar los tiempos de aforamiento, reducir costos y recursos, para así darle solución a los problemas anteriormente mencionados.

El sistema con el cual se tomará registro del comportamiento de los vehículos en las intersecciones puede ser implementado de las dos siguientes maneras. La primera es desarrollar una aplicación que apoye el conteo de vehículos que se realiza directamente en las intersecciones, con la cual personal posicionado en un punto estratégico de la intersección, pueda contar vehículos por clase y ruta tomada, haciendo mucho más fácil el conteo, ya que actualmente el personal hace uso de libretas o planillas. El segundo enfoque que se le puede dar, es el de utilizar tomas aéreas realizadas por un dron, para que estas tomas sean analizadas por una herramienta de software y mediante esta, realizar el conteo de los vehículos, llevando el registro de la ruta tomada por estos y su tipo, en estas cuatro categorías: liviano, moto, bus o camión.
Se evalúan escenarios de solución con diferentes técnicas basadas en la segunda estrategia recientemente mencionada, dándole al proyecto el nombre de AforosDron, ya que esta solución ofrece un mayor nivel de automatización en el proceso de aforamiento, analizando tomas aéreas realizadas por un dron con técnicas como, optical flow o detección de movimiento en videos, ya sea la técnica Lucas-Kanade Optical Flow[23] o Dense Optical Flow[23]. Otra alternativa es la detección de objetos en imagen mediante deep learning, ya se con el framework de Google para aprendizaje automático Tensor flow [24], la biblioteca de Redes Neuronales Keras [25], otra opción es Caffe [26]enfocada en aprendizaje profundo y por último el algoritmo de detección en imágenes llamado YOLOv3 [2]: You only look once por sus siglas en inglés.

1. Realidad Problemática.

Con la ayuda de tecnologías basadas en deep learning y visión artificial, es posible crear un sistema que realice los aforos de manera automática, analizando tomas aéreas realizadas por un dron.
La forma en la que actualmente se están llevando a cabo los aforos vehiculares demanda tiempo y personal. En el nuevo sistema que se desarrollara se busca realizar varios aforos simultáneamente de forma automática, liberando de tiempo y obligaciones a varios empleados para que puedan realizar otras actividades. También es clave resaltar el factor del desarrollo, innovación e inversión de tiempo en nuevas tecnologías como lo es el aprendizaje automático y visión artificial, ya que en un futuro se tendrá un conocimiento más amplio adquirido gracias a este proyecto, resultando en abrir un nuevo mercado para la empresa para dar solución a un sin fin de problemas a los que se enfrentan las secretarías de movilidad de los municipios, otras entidades y empresas.

La detección de automóviles a partir de imágenes aéreas, que es uno de los componentes más importantes del proyecto, ha sido estudiada durante años por diferentes equipos de desarrollo a nivel mundial. Sin embargo, dada una imagen aérea con variaciones típicas del aspecto del fondo y del automóvil, la detección robusta y eficiente del automóvil sigue siendo un problema difícil de solventar. Estos son algunos de los referentes que han abordado el problema.

2. Definición del problema.

El propósito de este proyecto es darle respuesta a la siguiente pregunta.

¿Es posible crear un sistema que mediante el análisis de tomas aéreas, utilizando tecnologías de deep learning y visión artificial, sea capaz de realizar aforos vehiculares, de manera que se optimicen los recursos de tiempo, costo y personal necesarios para la realización de estos?

3. **Antecedentes**.

Estos son algunos de los referentes que han abordado el problema.

Referentes internacionales.

- Localización y conteo de automóviles con imágenes aéreas [6]: Utilizan detección de objetos con la red neuronal convolucional YOLT2 para la tarea de detección de automóviles en imágenes aéreas. Para un conjunto muy amplio y diverso de imágenes de prueba con una resolución de 30 cm, localizamos correctamente los automóviles el 90% del tiempo y alcanzamos una tasa de error en el recuento de automóviles del 5%.

Estudios previos con YOLT2 demostraron resultados razonables con pequeños conjuntos de entrenamiento. En consecuencia, reservamos la región geográfica más grande (Utah, con 19,807 automóviles) para realizar pruebas. Esto deja 13,303 autos de entrenamiento entre Potsdam, Selwyn y Toronto.

- Vehicle detection and tracking in video [7]: Se presenta un esquema para la detección y seguimiento de vehículos en video. El método propuesto combina efectivamente el conocimiento estadístico sobre la clase de vehículos con información de movimiento. La distribución desconocida de los patrones de imagen de los vehículos se modela aproximadamente utilizando información estadística de orden superior derivada de imágenes de muestra. La información estadística sobre el fondo se aprende "sobre la marcha". Un detector de movimiento identifica regiones de actividad en la escena. Esto se logra mediante el aprendizaje del fondo, la búsqueda de imágenes y la clasificación. Para el aprendizaje del fondo y detección de vehículos se utiliza la técnica detección HOS que clasifica los patrones de prueba como vehículos o no. Cuando se prueba en imágenes reales de vistas aéreas de actividad vehicular, el método da buenos resultados incluso en escenas complicadas. No requiere ninguna información a priori sobre el sitio. Sin embargo, es susceptible de aumento con información contextual.
- Detección rápida de vehículos con agrupación de características probabilística y su aplicación al seguimiento de vehículos [8]: Un nuevo enfoque distinto al del proyecto para la detección de vehículos, realiza la fusión de sensores de un escáner láser y un sensor de video. Esta combinación proporciona suficiente información para manejar el problema de las múltiples vistas de un automóvil. El

escáner láser calcula la distancia y la información de contorno de los objetos observados. La información de contorno se puede utilizar para identificar los lados discretos de objetos rectangulares en el sistema de coordenadas del escáner láser. Toda esta información dada por el láser es utilizada para hacer un seguimiento al vehículo.

Referentes nacionales.

- La empresa Sistemas Inteligentes en Red S.A.S. [9] es una empresa basada en el conocimiento, orientada a la gestión inteligente de los sistemas de tiempo real aplicados al sector Transporte y Movilidad. Utilizan tecnologías de machine learning y procesamiento digital de imágenes, en conjunto con herramientas de ingeniería y estadística, tales como, aforos vehiculares mediante videoanalítica, hacen tomas aéreas realizadas con drones sobre las intersecciones de interés.
- La empresa Kineo Ingeniería del tráfico, S.L. [10] realiza aforos y clasificación vehicular: con sensores a nivel de calzada o introducidos en el pavimento, dependiendo del campo de aplicación en función de las características de la medición (temporal o permanente), del tipo de tránsito vehicular (flujo libre, a vuelta de rueda, arranque-parada), y de la precisión requerida. A nivel de sensores, las técnicas más comúnmente utilizados son: Manguera neumática, lazos inductivos, sensores piezoeléctricos, sensores en base a cables de fibra óptica.

Al analizar los referentes y sus sistemas implementados anteriormente mencionados, se determina por parte de los directivos de la empresa e ingenieros, realizar el sistema basado en procesamiento digital de imágenes e inteligencia artificial.

Objetivos

1. General.

Desarrollar un sistema que automatice el proceso de aforamiento vehicular en intersecciones, utilizando técnicas de aprendizaje profundo y procesamiento digital de imágenes, con el fin de optimizar tiempos, costos, recursos y personal en la realización del mismo.

2. Objetivos específicos.

- Desarrollar y/o modificar una GUI en python para la recolección de datos, necesarios para el entrenamiento del algoritmo de detección.
- Entrenar algoritmo de detección YOLOv3 para ubicar e identificar 4 tipos de vehículos: livianos, motos, buses y camiones.
- Implementar un sistema de tracking utilizando las detecciones realizadas por YOLOv3, para determinar la ruta seguida por el vehículo.
- Implementar algoritmo de conteo de vehículos en una intersección utilizando los insumos de la herramientas de detección y tracking.
- Evaluar resultados entregados por el sistema frente a afloramientos realizados y validados manualmente, mediante la métrica de medición el error cuadrático medio.

Marco Teórico

El sistema AforosDron con el cual se tomará registro del comportamiento de los vehículos en las intersecciones puede ser implementado de las dos siguientes maneras. La primera consiste en desarrollar una aplicación que apoye el conteo de vehículos directamente en las intersecciones que realiza el personal, la cual puede contar vehículos por clase y ruta tomada, haciendo mucho más fácil el conteo, que hasta hoy se realiza en libretas. El segundo enfoque que se le puede dar, es el de utilizar las tomas aéreas realizadas por el dron, para que sean analizadas por un sistema desarrollar y mediante este, realizar el conteo directamente del video, y así registrar los vehículos por tipo y por ruta.

Se evalúan escenarios de solución con diferentes técnicas basadas en las dos tecnologías recientemente mencionadas, tales como, optical flow o detección de movimiento en videos, ya sea la técnica Lucas-Kanade Optical Flow o Dense Optical Flow, la otra alternativa es la detección de objetos en imagen mediante deep learning, ya se con el framework de Google para aprendizaje automático Tensor flow, la biblioteca de Redes Neuronales Keras, otra opción es Caffe enfocada en aprendizaje profundo y por último el framework para entrenamientos de redes neuronales Darknet [3] y/o el sistema de reconocimiento de objetos en tiempo real YOLOv3 [2]. Haciendo un estimado en cuanto tiempo de desarrollo, costos, adquisición de conocimientos de la tecnología a usar y desempeño en la detección de objetos, se decide entonces tomar desarrollar la solución número dos, enfocada en el aprendizaje automático para la detección de objetos con la tecnología YOLOv3 [2] apoyándonos en el framework Darknet [3] para los entrenamientos de las red neuronal.

YOLOv3 [2]: You only look once por sus siglas en inglés, es el estado de arte de los sistemas en tiempo real de detección de objetos en imágenes, para su implementación se hace necesario el desarrollo de otras herramientas que facilitaran el trabajo, principalmente para el entrenamiento de la red neuronal, tales como la herramienta de etiquetado, la herramienta de preprocesado del video y herramienta de aumentación de datos, entre otras. Para entrenar la red se utilizará el framework Darknet [3] por su facilidad de uso, además cuenta con una muy buena comunidad online en los foros fundamentales para recibir soporte de su funcionamiento en caso de ser necesario. Con estas ventajas mencionadas se toman estas dos tecnologías como la base del proyecto.

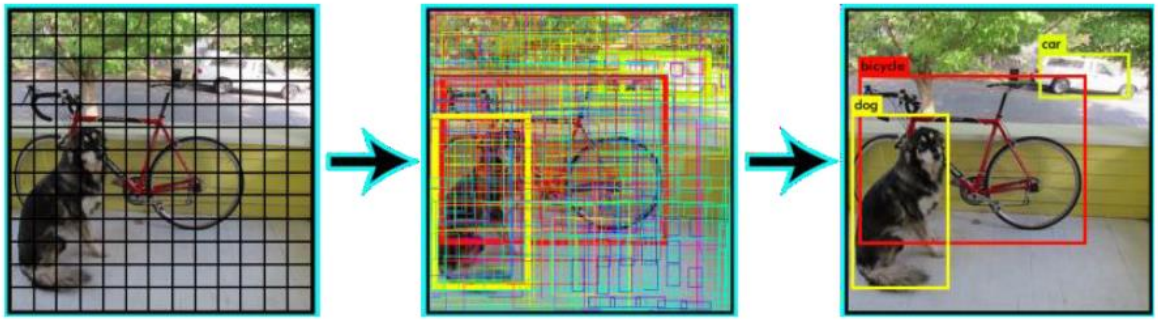
Funcionamiento de YOLOv3:

Podemos pensar en un detector de objetos como una combinación de un localizador de objetos y un clasificador de objetos como uno solo. En los enfoques tradicionales de visión artificial, se usaba una ventana deslizante para buscar objetos en diferentes ubicaciones y escalas. Como se trataba de una operación tan costosa (computacionalmente), se suponía que la relación de aspecto del objeto era fija los algoritmos de detección de objetos basados en Early Deep Learning como R-CNN y Fast R-CNN utilizaron un método llamado Búsqueda selectiva para reducir el número

de cuadros delimitadores que el algoritmo tuvo que probar, otro enfoque llamado Overfeat consistió en escanear la imagen a múltiples escalas utilizando mecanismos deslizantes tipo ventanas hechos de manera convolucional, esto fue seguido por F-R-CNN más rápido, que usó una técnica conocida como Region Proposal Network (RPN) para identificar los cuadros delimitadores que debían probarse. Mediante un diseño inteligente, las características extraídas para reconocer objetos también fueron utilizadas por el RPN para proponer posibles cuadros de límite, ahorrando así una gran cantidad de esfuerzo computacional.
YOLO, por otro lado, aborda el problema de detección de objetos de una manera completamente diferente. Envía la imagen completa solo una vez a través de la red. SSD es otro algoritmo de detección de objetos que reenvía la imagen una vez a través de una red de aprendizaje profundo, pero YOLOv3 es mucho más rápido que SSD y logra una precisión muy comparable. YOLOv3 ofrece resultados más rápidos que en tiempo real en una GPU M40, TitanX o 1080 Ti.

Veamos cómo YOLO detecta los objetos en una imagen dada.

Primero, divide la imagen en una cuadrícula de celdas de 13 × 13. El tamaño de estas 169 celdas varía según el tamaño de la entrada. Para un tamaño de entrada de 416 × 416 que usamos en nuestros experimentos, el tamaño de la celda era 32 × 32. Cada celda es responsable de predecir una serie de cuadros en la imagen.



Función de pérdida.[21]
YOLO predice múltiples cuadros delimitadores por una celda de coordenadas centrales *x, y* altura *h* y ancho *w* . Para calcular la pérdida de un verdadero positivo, solo queremos que uno de ellos sea responsable del objeto. Para este propósito, seleccionamos el que tiene la IoU (Intersection over union) más alta con la certeza mínima. Esta estrategia lleva a la especialización entre las predicciones de la celda delimitadora. [22]

YOLO usa el error de suma cuadrada entre las predicciones y la verdad básica para calcular la pérdida. La función de pérdida se compone de:

Perdida de clasificación [21]
Si se detecta un objeto, la pérdida de clasificación en cada celda es el error al cuadrado de las probabilidades condicionales de clase para cada clase:

$$\sum_{i=0}^{s^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \quad (1)$$

Donde:
1iobj= 1 si un objeto aparece en la celda i, de lo contrario es 0.
c = clase (Carro, moto, bus, camión, numerados de 0 a 3)
pi(c) Indica la probabilidad condicional para la clase c en la celda i.

Pérdida de localización [21]
La pérdida de localización mide los errores en las ubicaciones y tamaños de las cajas de predicción. Solo contamos la casilla responsable de detectar el objeto.

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] + \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right]$$

(2)

Donde:
xi = Coordenada x del centro de la celda.
yi= Coordenada y del centro de la celda.
wi =Ancho de la celda.
hi=Altura de la celda.
1ijobj= 1 si j th caja en la celda i es responsable por la detección del objeto, de otra manera es 0.
coord Incrementa el peso por las pérdidas de las coordenadas de las cajas.

No queremos ponderar los errores absolutos en cajas grandes y cajas pequeñas por igual. es decir, un error de 2 píxeles en un cuadro grande es el mismo para un cuadro pequeño. Para abordar esto parcialmente, YOLO predice la raíz cuadrada del ancho y alto del cuadro delimitador en lugar del ancho y alto. Además, para poner más énfasis en la precisión del cuadro de límite, multiplicamos la pérdida por λcoord (por defecto:

Pérdida de confianza [21]
Si se detecta un objeto en la caja, la pérdida de confianza (que mide la certeza de la caja) es:

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} (c_i - \hat{c}_i)^2 \quad (3)$$

Donde:
ci es la certeza de la j en la celda i.
1ijobj =1 si la caja j en la celda i es responsable de la detección del objeto, de otra manera 0.

Si el objeto no es detectado la pérdida de confianza es la siguiente:

$$\lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (c_i - \hat{c}_i)^2 \quad (4)$$

1ijnoobj: es el complemento de 1ijobj.
Ci: es la certeza de la j en la celda i.
Noobj: baja el peso de la pérdida cuando se detecta un fondo.

La mayoría de las cajas no contienen ningún objeto. Esto causa un problema de desequilibrio de clases, es decir, entrenamos al modelo para detectar el fondo u objetos innecesarios con más frecuencia que detectar los objetos para los cuales lo estamos entrenando. Para remediar esto, atenuamos esta pérdida en un factor λnoobj (predeterminado: 0.5).

Pérdida [21]
La pérdida final agrega pérdidas de localización, confianza y clasificación juntas, así:

$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] + \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right]$$

$$+ \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{obj} (c_i - \hat{c}_i)^2 + \lambda_{noobj} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (c_i - \hat{c}_i)^2$$

$$\sum_{i=0}^{s^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \quad (5)$$

Estas ecuaciones reflejan que tan seguro es el modelo de que la caja contenga un objeto y también cuán preciso cree que es la caja que predice. Si no existe ningún objeto en esa celda, los puntajes de confianza deben ser cero. De lo contrario, queremos que el puntaje de confianza sea igual a la intersección sobre la unión (IOU) entre el cuadro predicho y la verdad básica.

Inferencia: *Non-maximal suppression*

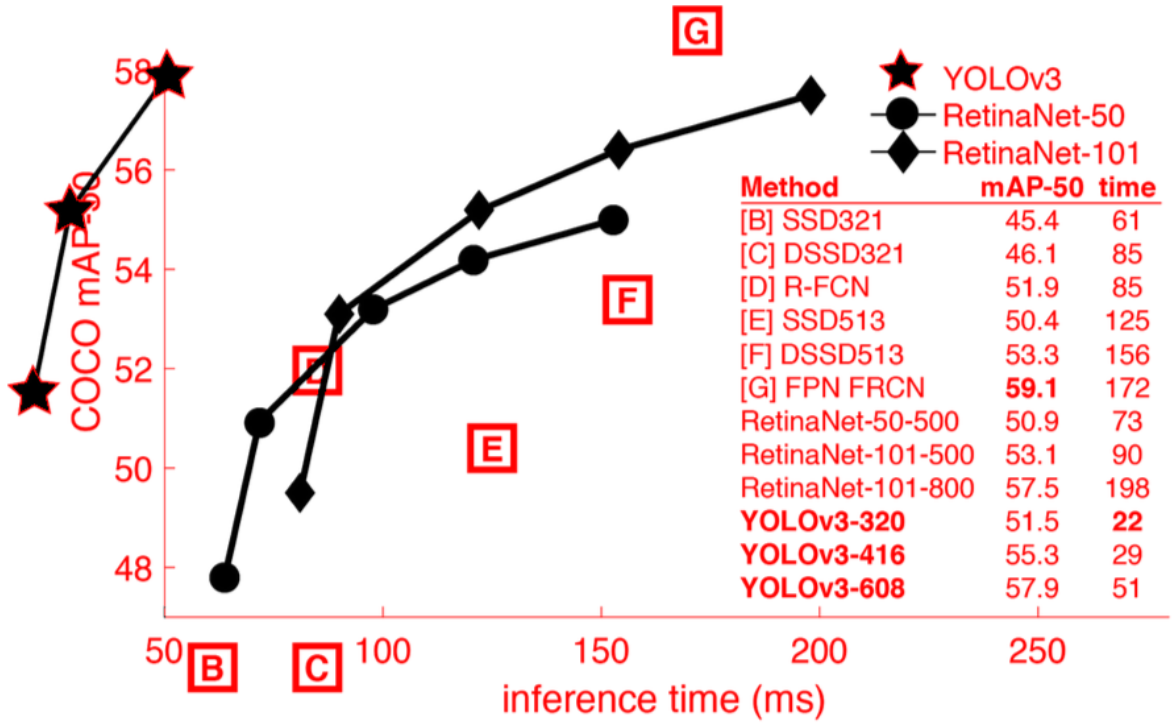
YOLO puede hacer detecciones duplicadas para el mismo objeto. Para solucionar esto, YOLO aplica supresión no máxima para eliminar duplicaciones con menor confianza:

1. Ordenar las predicciones por los puntajes de confianza.
2. Comience desde los puntajes más altos, ignore cualquier predicción actual si encontramos predicciones previas que tengan la misma clase e IoU> 0.5 con la predicción actual.
3. Repetir el paso 2 hasta que se verifiquen todas las predicciones.

Beneficios de YOLO.

1. Rápido. Bueno para procesamiento en tiempo real.
2. Las predicciones (ubicaciones de objetos y clases) se realizan desde una sola red. Se puede entrenar de punta a punta para mejorar la precisión.
3. YOLO es más generalizado. Supera a otros métodos al generalizar desde imágenes naturales a otros dominios como obras de arte.
4. Los métodos de propuesta de región limitan el clasificador a la región específica. YOLO accede a toda la imagen en la predicción de límites. Con el contexto adicional, YOLO demuestra menos falsos positivos en las áreas de fondo.

5. YOLO detecta un objeto por celda de cuadrícula. Aplica la diversidad espacial al hacer predicciones.
6. En mAP(mean Average Precision) medido a .5 IOU(Intersection over Union) YOLOv3 [2] está a la par con la Pérdida focal pero aproximadamente 4 veces más rápido. Además, puede intercambiar fácilmente entre velocidad y precisión simplemente cambiando el tamaño del modelo, sin necesidad de reentrenar el modelo.



Comparación en FPS de YOLOv3 [2] con otros modelos de detección.

Model	Train	Test	mAP	FLOPS	time (ms)
SSD300	COCO trainval	test-dev	41.2	-	46
Retinanet-50-500	COCO trainval	test-dev	50.9	-	73

Retinanet-101-800	COCO trainval	test-dev	57.5	-	198
YOLOv3-320	COCO trainval	test-dev	57.9	140.69 Bn	22

Tabla 1: Desempeño de diferentes modelos con el COCO Dataset.

Los autores de YOLOv3, Joseph Redmon y Ali Farhadi, han hecho que YOLOv3 sea más rápido y preciso que su trabajo anterior YOLOv2. YOLOv3 maneja mejor las escalas múltiples. También han mejorado la red al hacerla más grande y llevarla hacia redes residuales al agregar conexiones de acceso directo.

Metodología

Recolección de datos y desarrollo de GUI para etiquetado.

La recolección de datos se llevó a cabo capturando tomas aéreas desde dron en diferentes intersecciones del municipio de Rionegro, con un total de 13 intersecciones importantes en materia de movilidad, en diferentes horas del día. Los videos tomados en cada intersección tienen una duración promedio de 8 minutos.

Intersección	**Fecha**	**Videos**
Cruce la Pola	5 de marzo	5
Zeus	14 de marzo	15
El chato	15 de marzo	8
Alto de la capilla	19 de marzo	9
Ibiza	20 de marzo	5
Drogueria cordoba	1 de marzo	8
Exito San Nicolas	21 de mayo	8
Cruce terminal Rionegro	22 de mayo	8
Somer	22 febrero	14

Cruce Piamonte	24 de mayo	14
Cruce terpel	27 de febrero	10
Cruce auteco	28 de febrero	10
Cruce mimos	5 de abril	7

Tabla 2: Tomas aéreas realizadas por intersección.

Se realizaron tomas aéreas en horas de la mañana entre las 6 am y las 8 am y en la tarde entre las 3 pm y 5 pm. Esto con el fin de tener una mayor cobertura a diferentes niveles de luminosidad y sombra en los vehículos. También se grabó las alturas diferentes con el dron, alturas de entre 15 y 30 metros. Todas las tomas se realizaron con el dron totalmente inmóvil para poder utilizar una herramienta de tracking en la *GUI* de etiquetado que más adelante se explicará.

Desarrollo de GUI en python para el etiquetado de imágenes.

Tipo	**# Clase**
Carro	0
Moto	1
Bus	2
Camión	3

Tabla 3: Tipo de vehículo con su número de clase correspondiente.

El banco de datos recolectado necesita pasar por un proceso de etiquetado antes del entrenamiento, imágenes de los objetos de interés - carro, moto, bus y camión - etiquetados de manera que cumpla el formato requerido por framework darknet para empezar el entrenamiento. El formato está definido de la siguiente manera:

object-id center_x center_y width height

Donde:

object-id: Es el número clase correspondiente a cada vehículo, que se puede ver en la tabla 2.

center_x: Corresponde a la coordenada x del centro de la caja con la que se encierra el objeto de interés. Ver figura 1.

center_y: Corresponde a la coordenada y del centro de la caja con la que se encierra el objeto de interés. Ver figura 1.

width: Corresponde al ancho de la caja que con la que se encierra el objeto de interés. Ver figura 1.

height: Corresponde al alto de la caja que con la que se encierra el objeto de interés. Ver figura 1.

Consideraciones:

- Una fila por objeto.

- Las coordenadas de la caja deben estar en formato normalizado (de 0 a 1).

- Los números de clase están indexados a cero (comienzan desde 0).

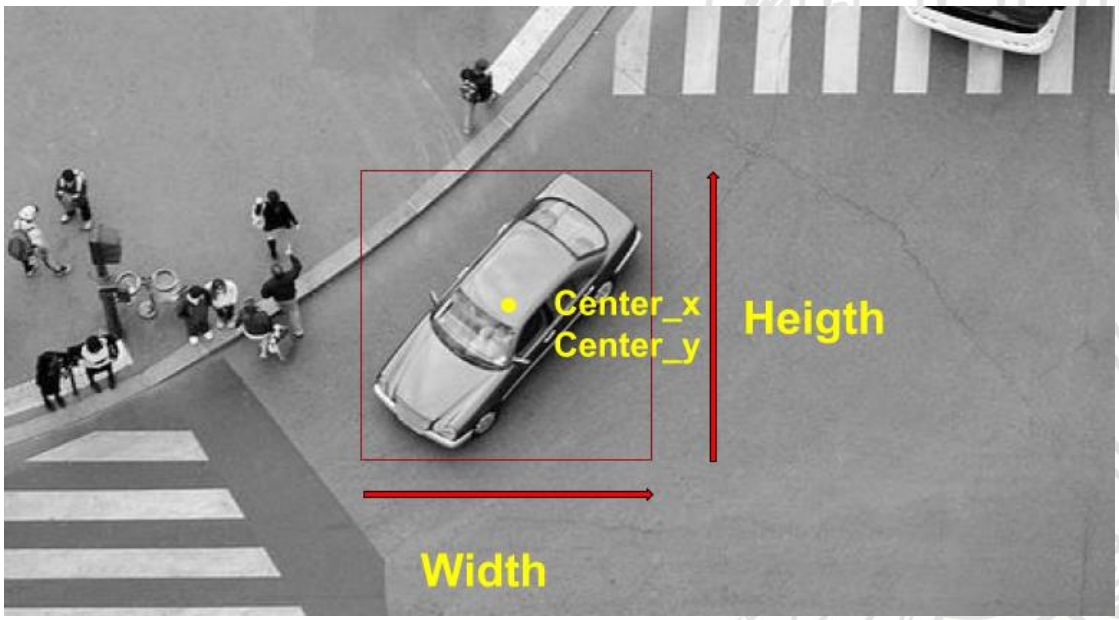


Figura 1. Formato de etiquetado de objetos.

Por cada imagen se debe crear un archivo .txt que contenga toda la información con el formato requerido de los objetos etiquetados en esta.

Para el desarrollo de la herramienta de etiquetado se toma como base *OpenLabeling Tool* de licencia libre, escrita en python. Se le añaden las

siguientes funcionalidades:

1. Extracción de frames de videos en formato .MP4.
2. *DASIAMRPN Tracker.*
3. Separación de formato Voc con formato YOLO en archivos .txt de salida.
4. El archivo de etiqueta de cada imagen debe ser localizable simplemente reemplazando /images/*.jpg con /labels/*.txt en su nombre de ruta. Un ejemplo de imagen y par de etiquetas sería:

   ../coco/images/train2017/000000109622.jpg // image

   ../coco/labels/train2017/000000109622.txt // label

Código fuente herramienta de etiquetado.

Un total se etiquetaron 1556 imágenes provenientes del banco de datos recolectado, asegurando una gran diversidad de vehículos por cada una de las cuatro clases.

Tipo vehículo	# Clase	Etiquetas
Carro	0	3972
Moto	1	1956
Bus	2	493
Camión	3	336

Tabla 4: Etiquetas por tipo de vehículo.

6.2 Entrenamiento de YOLOv3.

Requerimientos:

1. El entrenamiento necesita que las etiquetas realizadas en el paso anterior, sean organizadas en dos grupos, el primer grupo es de entrenamiento y el otro grupo es de *test.* Se dividen las etiquetas en una proporción 80:20, 80% de las etiquetas para entrenamiento y 20% de etiquetas para validación. Las direcciones de la imagen y su archivo .txt con las etiquetas de esta, serán almacenadas en un uno de dos

archivos text.txt o train.txt.

2. Archivo con extensión. names, en el cual están los nombres de las clases de los objetos etiquetados y en este orden: cars, moto, bus, camion.
3. Archivo de extensión .data, en este archivo se llenarán los siguientes campos:

   classes: Tendrá el numero total de clases, o sea 4.

   train = Ruta del archivo train.txt

   valid = Ruta del archivo test.txt

   names = Ruta del archivo de extensión. names

   backup = Ruta de *backup* o *checkpoint* de entrenamientos pausados.

4. Actualizar archivo de extensión .cfg. Según el rendimiento requerido, se puede seleccionar el archivo de configuración YOLOv3. Para este proyecto usaremos yolov3.cfg. Mientras se entrenan las imágenes, los pesos de las redes neuronales se actualizan de forma iterativa. Con un set de conjunto de datos de entrenamiento enormes, lo que consume muchos recursos para actualizar los pesos de todo el conjunto de entrenamiento en una sola iteración. Para usar un pequeño conjunto de imágenes - como en este caso - para actualizar iterativamente los pesos, se configura el parámetro por lotes. Por defecto está configurado en 64. El número máximo de iteraciones para las cuales nuestra red debe ser entrenada se establece con el parámetro *max_batches* = 4000. También actualice los pasos = 3200,3600, que es 80%, 90% de *max_batches* respectivamente. Con esto se actualizan las clases y los parámetros de filtro de las capas [*yolo*] y [*convolucional*] que están justo antes de las capas [*yolo*]. En este ejemplo, dado que tenemos cuatro clases actualizaremos las clases en las capas [*yolo*] a 1 en los números de línea: 610, 696, 783. Del mismo modo, se actualiza el parámetro de filtros en función del recuento de clases filtros = (clases(4) + 5) * 3. Para una sola clase debemos establecer filtros = 27 en los números de línea: 603, 689, 776. Todos los cambios de configuración se realizan en custom_data/cfg/*yolov3-custom.cfg*.
5. Para entrenar el detector de objetos, se usa los pesos pre entrenados existentes que ya están entrenados en grandes conjuntos de datos. En este proyecto se utilizan los pesos ***darknet53.conv.74*** **[3]** recomendados en la documentación de YOLOv3.

Teniendo ya todos los requisitos y las configuraciones realizadas, se empieza con el entrenamiento. El entrenamiento se realiza en un PC de sistema operativo Linux con las siguientes características:

Especificaciones de hardware

Dron	**Modelo**	DJI Spark, más control y 4 baterías de repuesto. [13]
Computador	**Processor**	4x Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz
	Memory	8108 MB (6361 MB used)
	Operating System	Ubuntu 16.04.6 LTS
	Date/Timevie	02 ago 2019 09:38:46 -05
	GPU	VGA compatible controller: NVIDIA Corporation GM206 [GeForce GTX 950]

Tabla 5: Especificaciones de hardware.

Especificaciones de software

Componente	**Versión**
Sistema operativo	Ubuntu 16.04.6 LTS
Python3 [16]	Python 3.5.2
OpenCV [4]	4.1.0
Dlib [17]	19.17.0
Numpy [18]	1.16.4
Imutils [19]	0.5.2
CUDA [20]	9.1.85
Entorno de desarrollo	Microsoft Visual Studio: versión 16.1.6

Tabla 6: Especificaciones de software.

El entrenamiento se puede continuar hasta alcanzar un cierto umbral. De configura para que, de manera automática, los pesos para el detector se guardaran por cada 100 iteraciones hasta alcanzar las 1000 iteraciones y luego continúa guardando por cada 10000 iteraciones. Este comportamiento se modifica actualizando la condición en la línea 138 del archivo *detector.c*.

El resultado de las detecciones realizadas con el algoritmo se muestra a continuación:

Figura 2: Detección de vehículos en un solo frame.

Para medir el algoritmo de detección se pasan 100 frames o imágenes por este, y con esos datos se evalúa el rendimiento del algoritmo con la matriz de confusión y precisión y recuperación.

Tipo vehículo	**#**
Liviano	565

Moto	345
Bus	65
camión	53
Total	1028

Tabla 7: Conteo de vehículos en los 100 frames.

	Liviano - Predicho	**Liviano - No Predicho**
Liviano	544	21
No Liviano	8	455

Tabla 8: Matriz de confusión para vehículos tipo livianos.

Precisión (Precision)	98.55%
Recuperación (Recall)	96.28%

Tabla 9: Precisión y recuperación de livianos.

	Moto - Predicho	**Moto - No Predicho**
Moto	286	59
No Moto	0	683

Tabla 10: Matriz de confusión para vehículos tipo motos.

Precisión (Precision)	100%
Recuperación (Recall)	82.89%

Tabla 11: Precisión y recuperación de livianos.

	Bus - Predicho	**Bus - No Predicho**
Bus	55	10
No Bus	7	956

Tabla 12: Matriz de confusión para vehículos tipo buses.

Precisión (Precision)	88.70%
Recuperación (Recall)	84.61%

Tabla 13: Precisión y recuperación de livianos.

	Camión - Predicho	**Camión - No Predicho**
Camion	46	7
No Camion	7	968

Tabla 14: Matriz de confusión para vehículos tipo camiones.

Precisión (Precision)	86.79%
Recuperación (Recall)	86.79%

Tabla 15: Precisión y recuperación de livianos.

6.3 El algoritmo de detección devuelve las coordenadas y clase del vehículo, mas que suficiente para rastrear el vehículo por unos cuantos frames. El seguimiento del vehículo es parte fundamental del sistema, de modo que se diseña un algoritmo de *tracking* con varias pasos validación, con el fin de preservar el vehículo durante todo su trayecto.

Para el seguimiento del vehículo en la escena se crean las dos siguientes clases en python llamadas *TrackableObject y CentroidTracker,* en las cuales se almacenará la información de cada vehículo una vez detectado y se evaluará el estado del mismo. Los atributos de clase del primer objeto son:

self.objectID: Id único por cada vehículo que entre en escena.

self.centroids: Centro de la caja que contiene el vehículo detectado.

self.color: Color de la caja de detección según sea la clase del vehículo.

self.vehicle: Tipo de vehículo detectado.

self.counted: Indica si el vehículo ya está siendo seguido o no.

La segunda clase contiene los siguientes atributos:

self.nextObjectID: Contiene el id del próximo *TrackableObject*. Comienza en 0.

self.objects: Contiene los centroides de los vehículos que estaban siendo seguidos en el momento.

self.carTypeObjects: Contiene el tipo de vehículo que se están siguiendo en el momento.

self.disappeared: Contiene el número de veces en el que el vehículo se marca como desaparecido.

self.carRouteIn: Contiene todos los centroides de la primera detección de cada vehículo.

self.carRouteOut: Contiene todos los centroides de la última detección de cada vehículo.

self.maxDisappeared: Tope máximo del contador de desapariciones de los vehículos.

self.maxDistance: Distancia máxima{ de aparición con respecto a la anterior de los vehículos antes de marcarlos como desaparecidos.

El flujo de programa que siguen objetos de estas dos clases es el siguiente es el mostrado en la figura 3:

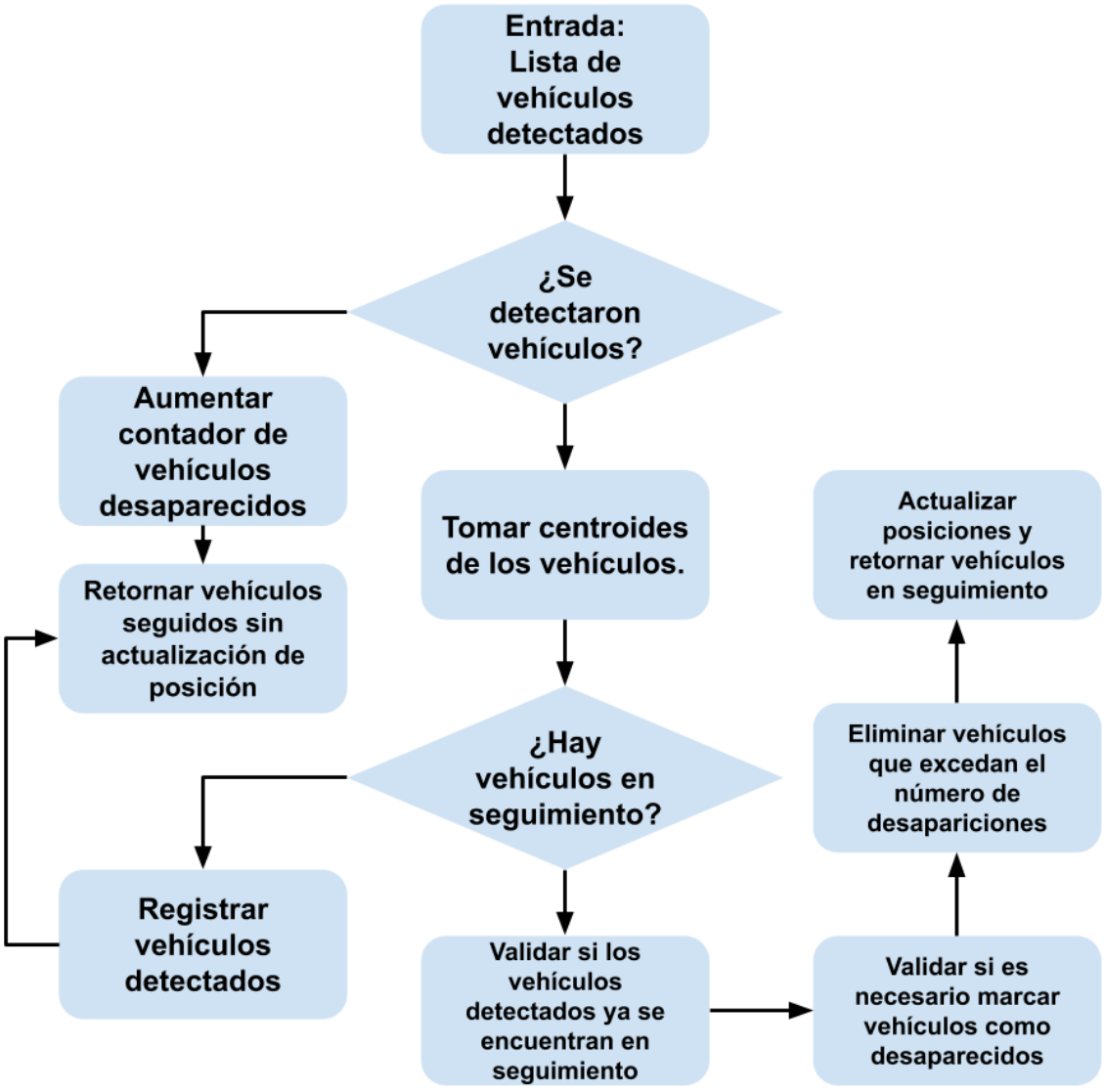


Figura 3: Flujo de programa para el tracking de los vehículos.

Las dos clases anteriormente mencionadas son creadas con el fin de evitar la expansión de las cajas de detección, ya que os *tracking* tienden a expandir la caja con la que hacen seguimiento al vehículo, lo cual representaba un problema, principalmente cuando los objetos de interés, se acercaban mucho entre sí. Lo que permite la solución implementada, es dar más rigor en el seguimiento y poder asegurar una menor pérdida en los vehículos seguidos. Otro de los propósitos de esta

implementación es bajar tiempos en el procesamiento del video, ya que realizar las detecciones con YOLOv3 tiene un alto esfuerzo computacional. Entonces solo se realizan detecciones cada 5 frames, en los 4 frames restantes se hace el seguimiento del vehículo con el algoritmo de tracking implementado para este proyecto. De esta manera evitamos hacer detecciones en un 80% de los frames.

6.4 El conteo de vehículos se basa en los 2 pasos anteriores, para utilizar su salida como insumo y así determinar la ruta que tomó el vehículo en la escena. Para realizar esto, el sistema necesita saber las entradas y salidas de las intersecciones, las cuales son ingresadas al inicio del programa. Para esto fue necesario crear una GUI, con la que se dibuja en la imagen, polígonos que marcan entradas en verde y salidas en rojo, sobre la intersección de interés, de la siguiente manera.

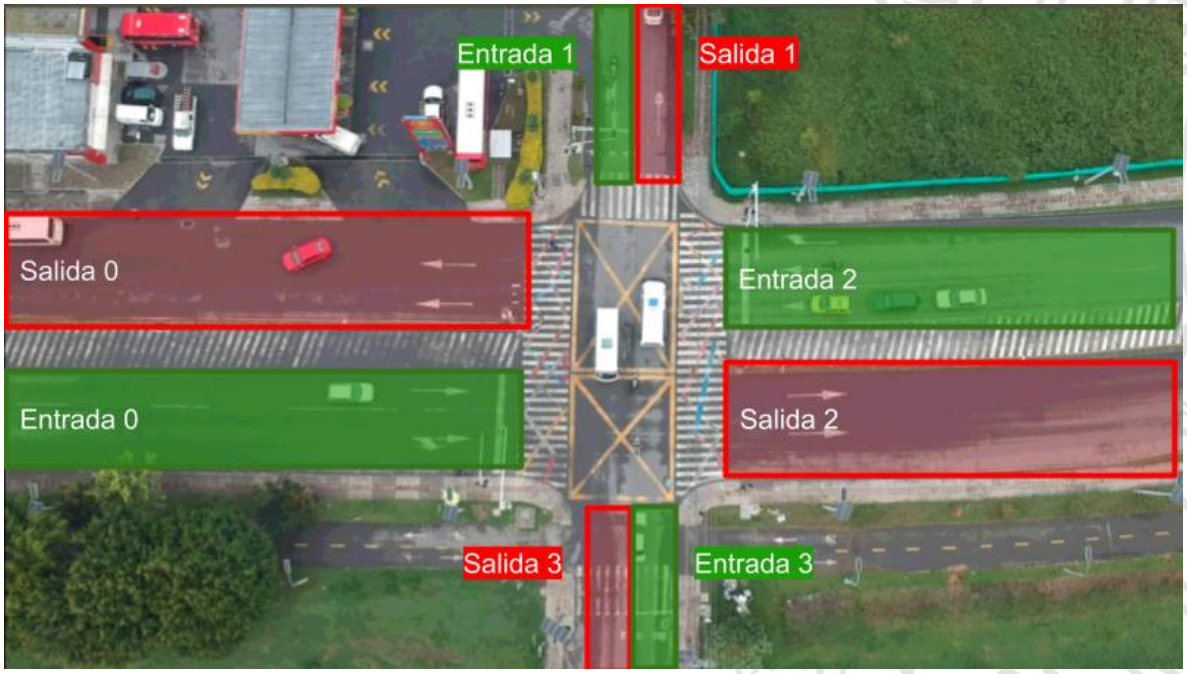


Figura 4: Selección de entradas y salidas en intersección, entradas en verde y salidas en rojo.

Cuando el vehículo es detectado por primera vez, a este se le asocia un id, una de las 4 clases de vehículo y las coordenadas en píxeles del lugar donde se detectó por primera vez, llamadas *carRouteIn.* Cuando el vehículo es detectado por última vez, se le agrega una variable más llamada *carRouteOut*, la cual guarda las coordenadas en pixeles de donde el vehículo fue detectado por última vez.

Al finalizar todas las detecciones en el video, ya se tienen todas las

coordenadas *carRouteIn y carRouteOut,* además también se cuenta con la información de las salidas y las entradas ver figura 4. El sistema procede a realizar validaciones, para saber si un vehículo tiene el *carRouteIn* dentro de alguno de los polígonos de entrada y luego verificar que el *carRouteOut* esté dentro de alguno de los polígonos de salida. Ver figura 5.

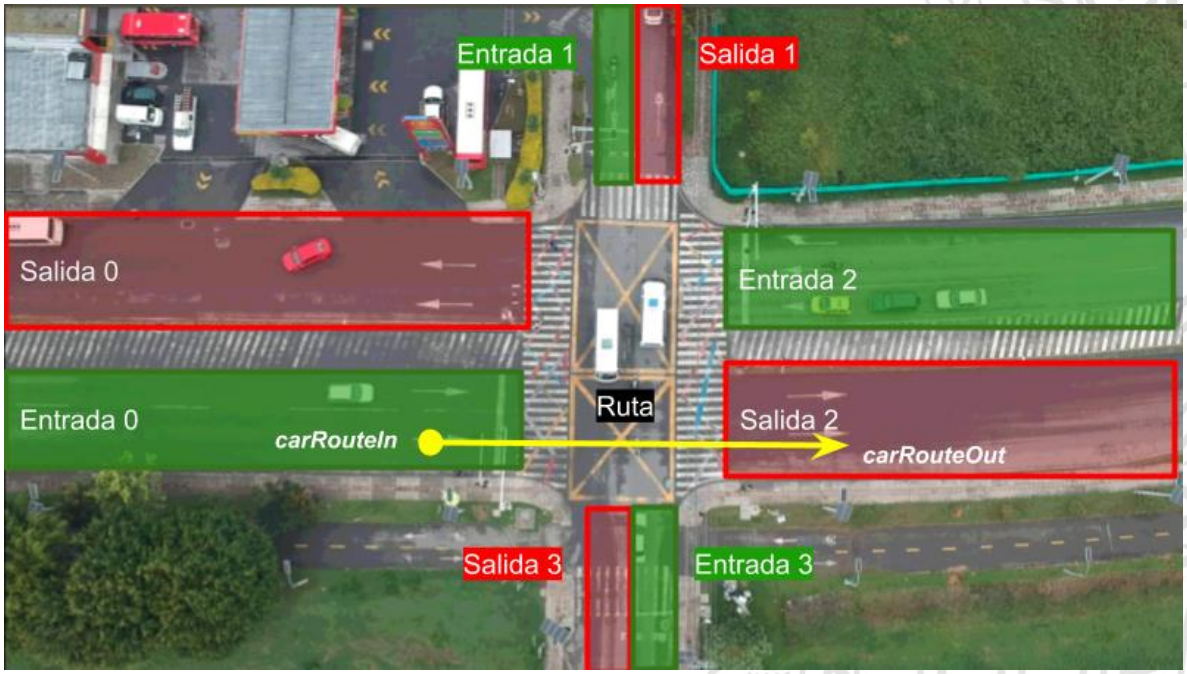


Figura 5: Detección inicial y detección final, para determinar ruta tomada por vehículo.

El código utilizado para realizar el conteo se encuentra en el archivo tracking_cars.py del repositorio en GitHub del proyecto debidamente comentado.

7.5 Para hacer el test no se deben utilizar los videos con los que se alimentó el algoritmo de detección, por esto, se realizan tres nuevas tomas aéreas dentro de las 13 intersecciones iniciales, escogidas de forma aleatoria.

Resultado de aforamientos:

Los resultados se muestran en una tabla con tres tipos de vehículos diferentes: liviano, moto, pesado. Liviano corresponde a la clase 0 de vehículo: cars, Moto corresponde a la clase 1 de vehículo: moto, Pesados corresponde a la clase de vehículo 2 y 3, que corresponden a la clase 2: bus y la clase 3: camión.

Intersección 1: Droguería Córdoba.

Figura 6: Vista superior de intersección.

				Livianos		Motos		Pesados	
Resultados	**Ruta**	**Entrada**	**Salida**	**A**	**M**	**A**	**M**	**A**	**M**
00 [32, 13, 3, 6]	0	0	0	32	34	13	19	9	8
01 [57, 34, 22, 6]	1	0	1	57	58	34	44	28	29
10 [6, 13, 1, 0]	2	1	0	6	7	13	22	1	1
11 [2, 0, 0, 0]	3	1	1	2	3	0	9	0	1

Tabla 16: Resultados de aforamiento automático (A) y aforamiento manual (M).

Gráficas comparativas entre aforamiento automático vs aforamiento manual:

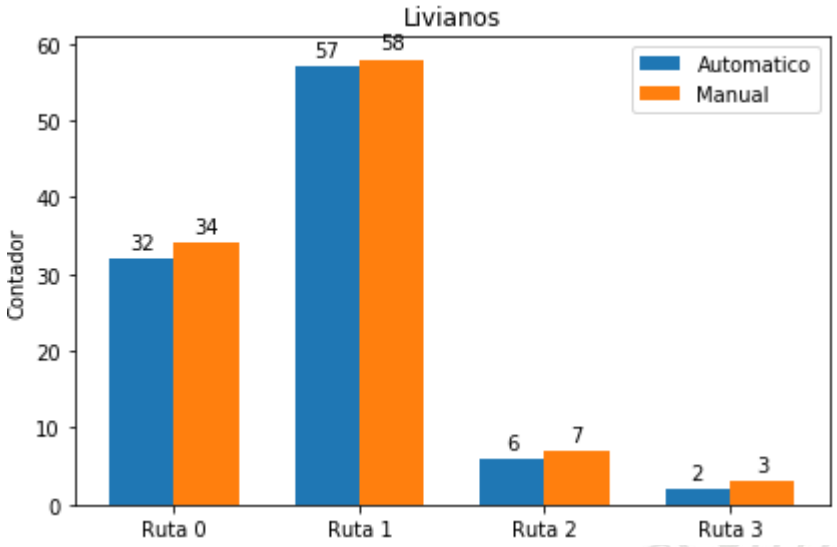


Figura 7: Comparativa por ruta, con tipo de vehículo liviano.

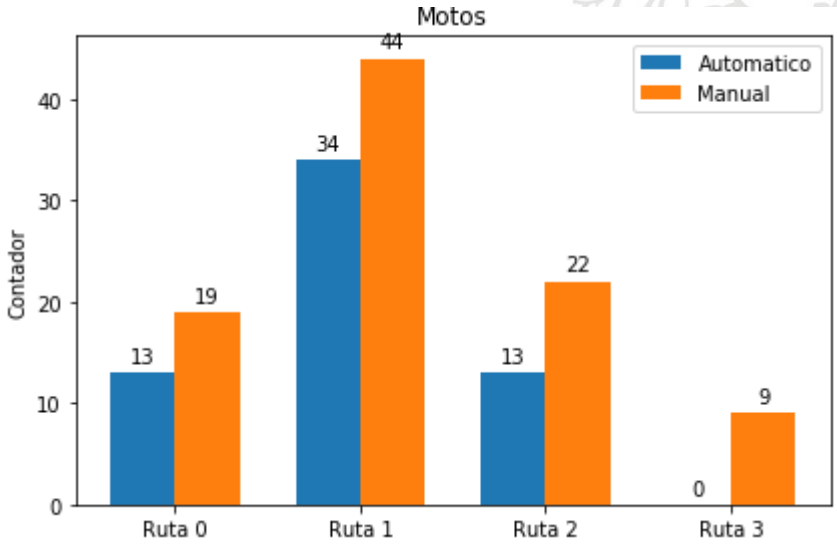


Figura 8: Comparativa por ruta, con tipo de vehículo moto.

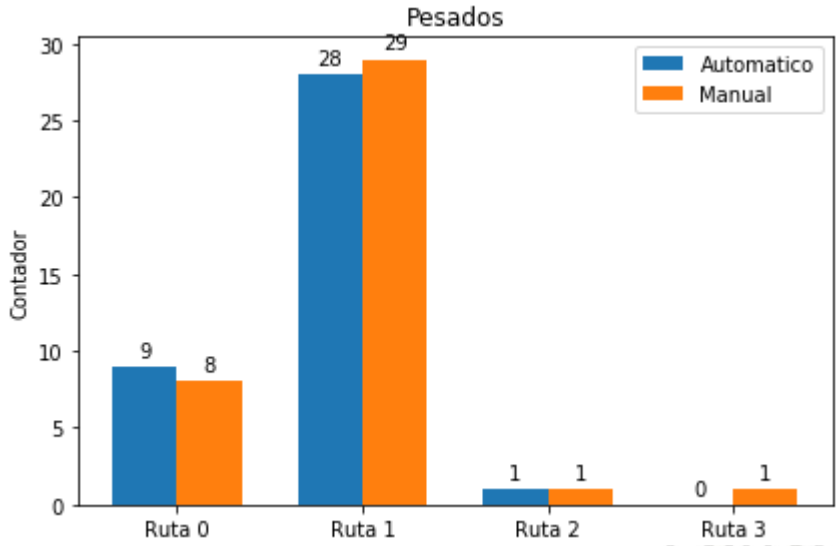


Figura 9: Comparativa por ruta, con tipo de vehículo pesados.

Intersección 2: Balcones.

Figura 10: Vista superior de intersección.

				Livianos		Motos		Pesados	
Resultados	**Ruta**	**Entrada**	**Salida**	**A**	**M**	**A**	**M**	**A**	**M**
00 [91, 42, 13, 7]	0	0	0	91	103	42	72	20	25
01 [25, 9, 5, 1]	1	0	1	25	28	9	10	6	7
02 [0, 0, 0, 0]	2	0	2	0	0	0	0	0	0
10 [0, 0, 0, 0]	3	1	0	0	0	0	0	0	0
11 [0, 0, 0, 0]	4	1	1	0	0	0	0	0	0
12 [115, 27, 13, 9]	5	1	2	115	110	27	45	22	23

Tabla 17: Resultados de aforamiento automático (A) y aforamiento manual (M).

Gráficas comparativas entre aforamiento automático vs aforamiento manual:

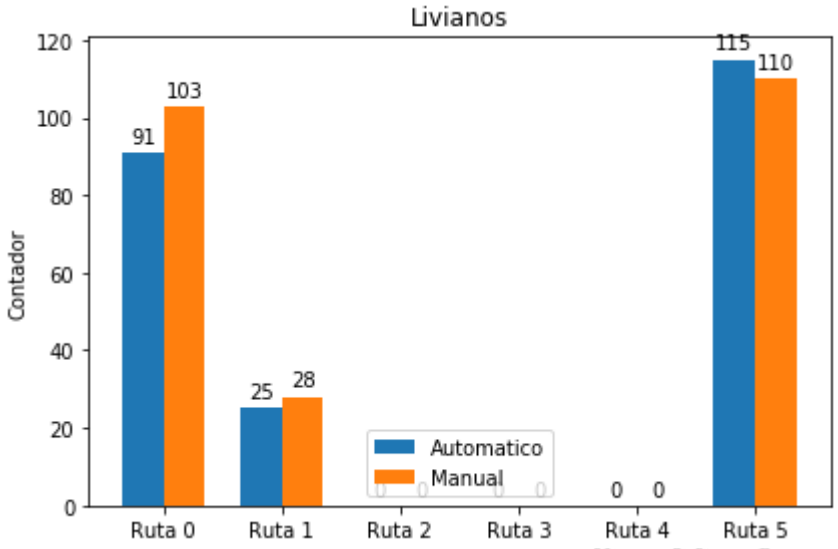


Figura 11: Comparativa por ruta, con tipo de vehículo liviano.

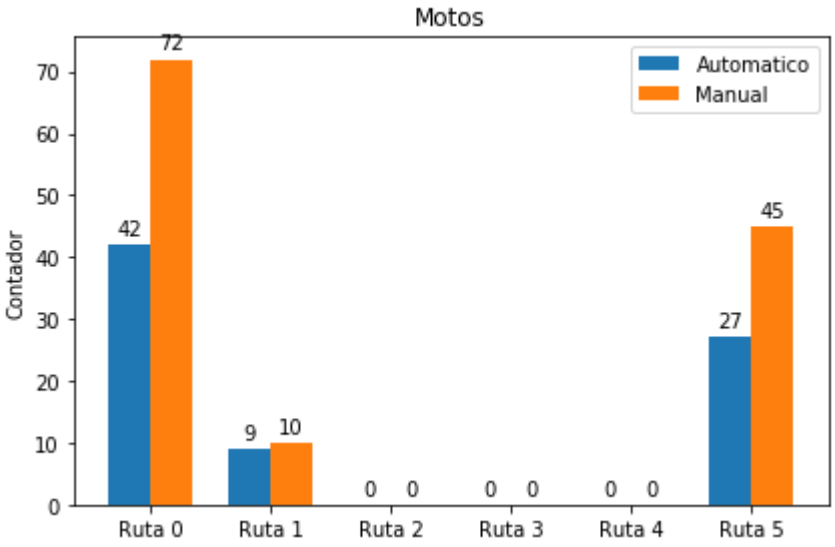


Figura 12: Comparativa por ruta, con tipo de vehículo Motos.

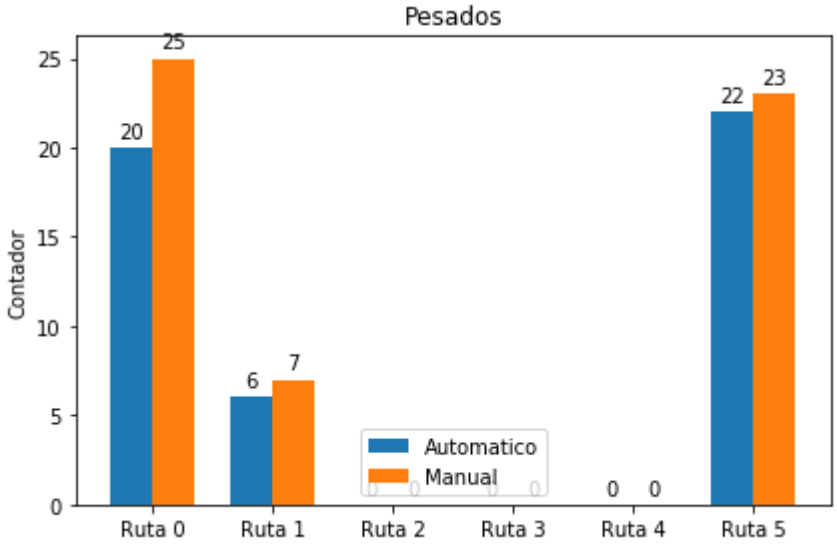


Figura 13: Comparativa por ruta, con tipo de vehículo Pesados.

Intersección 2: Éxito San Nicolás.

Figura 14: Vista superior de intersección.

				Livianos		Motos		Pesados	
Resultados	**Ruta**	**Entrada**	**Salida**	**A**	**M**	**A**	**M**	**A**	**M**
00 [1, 0, 0, 0]	0	0	0	1	0	0	0	0	0
22 [0, 2, 0, 1]	1	2	2	0	0	2	0	1	0
01 [56, 11, 1, 1]	2	0	1	56	61	11	29	2	4
20 [0, 0, 0, 0]	3	2	0	0	0	0	0	0	0
32 [0, 0, 1, 0]	4	3	2	0	0	0	0	1	0
11 [0, 0, 0, 0]	5	1	1	0	0	0	0	0	0
10 [97, 25, 5, 12]	6	1	0	97	106	25	34	17	18
33 [0, 1, 0, 0]	7	3	3	0	0	1	0	0	0
31 [0, 1, 0, 0]	8	3	1	0	0	1	0	0	0
03 [4, 2, 0, 0]	9	0	3	4	6	2	4	0	0
21 [6, 1, 1, 2]	10	2	1	6	4	1	2	3	4

12 [0, 0, 0, 0]	11	1	2	0	0	0	0	0	0
02 [25, 11, 1, 0]	12	0	2	25	26	11	17	1	1
30 [0, 0, 0, 0]	13	3	0	0	0	0	0	0	0
23 [0, 0, 0, 0]	14	2	3	0	0	0	0	0	0
13 [0, 0, 0, 0]	15	1	3	0	0	0	0	0	0

Tabla 18: Resultados de aforamiento automático (A) y aforamiento manual (M).

Gráficas comparativas entre aforamiento automático vs aforamiento manual:

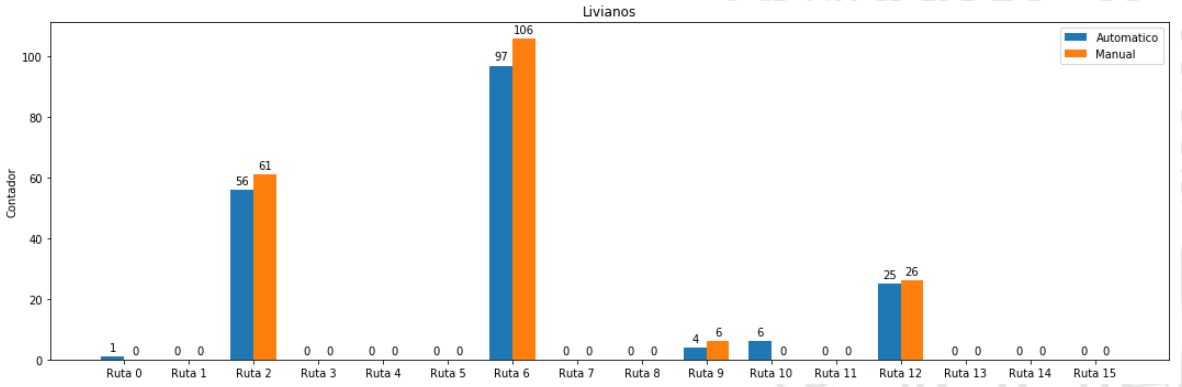


Figura 15: Comparativa por ruta, con tipo de vehículo Livianos.

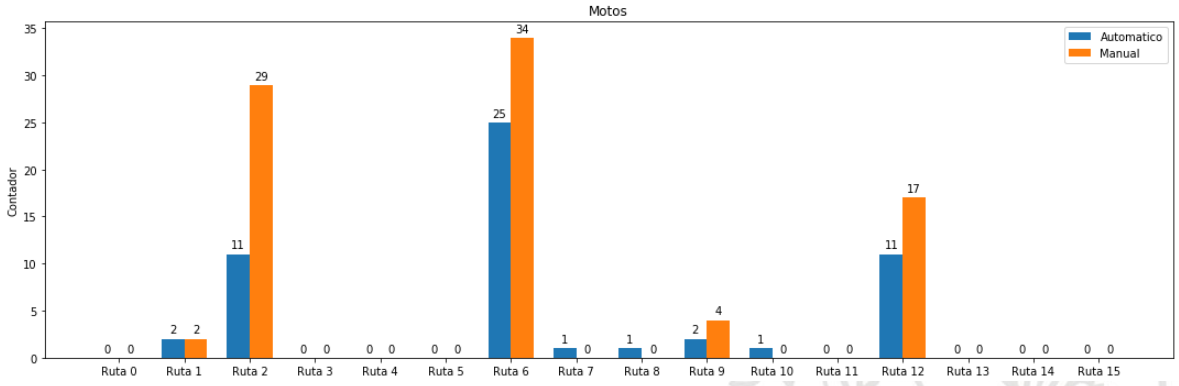


Figura 16: Comparativa por ruta, con tipo de vehículo Motos.

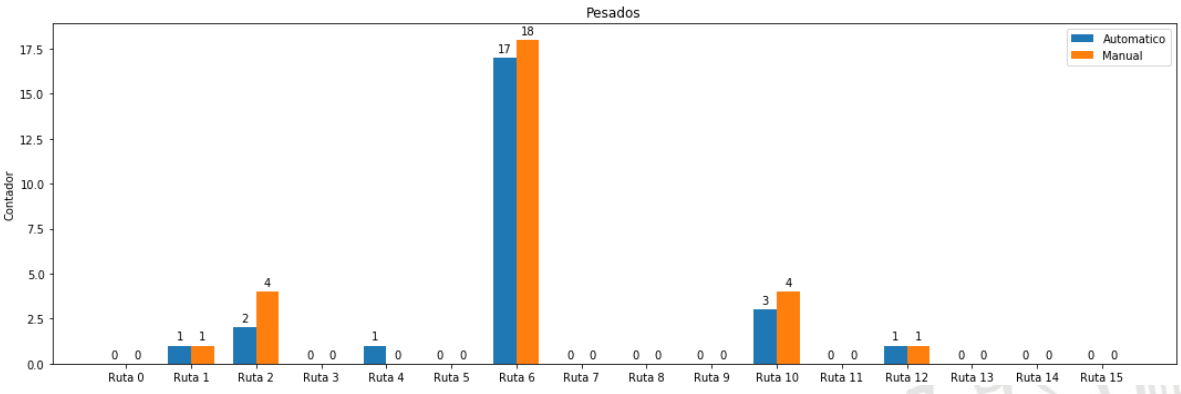


Figura 17: Comparativa por ruta, con tipo de vehículo Pesados.

Resultados y análisis

GUI para etiquetado de imágenes.

- La GUI facilito de sobre manera el etiquetado de las 1556 imágenes, ya que cuenta con seguimiento - *tracking* -, lo que permite que se realicen 30 etiquetas de manera rápida y automática.
- Los atajos de teclado permiten navegar alrededor de los frames extraídos de los videos y de las clases de vehículos.
- Cuenta con un algoritmo de fragmentación de video, que funciona perfectamente, con el cual se fraccionan en el número de frames deseados los videos de las intersecciones,
- Guarda de manera ordenada y precisa el formato requerido por YOLOv3 para el entrenamiento.
- Crea de manera exitosa los archivos train.txt y test.txt necesarios para el entrenamiento y validación de las predicciones durante el entrenamiento del algoritmo de detección.

Algoritmo de detección.

- En la tabla 8 se muestra la matriz de confusión de los vehículos tipo livianos, que nos muestra que, de 565 livianos, se lograron detectar 544, lo cual nos da una *recall* de 96.28%, diciéndonos que, del total de autos livianos en el set de imágenes de test, más del 96% lograron ser clasificados correctamente. A su vez la tabla 9 nos muestra la precisión de la detecciones con un porcentaje de 98.55% lo cual muestra que de todas las detecciones que se tomaron como livianos solo el 1.45% fueron incorrectas.
- En la tabla 10 se muestra la matriz de confusión de la clase número 1 correspondiente al tipo de vehículo motos, que nos muestra que de 345 vehículos tipo moto, se lograron detectar 286, lo cual nos da un *recall* de 82.89%, diciéndonos que del total de motos en el set de imágenes de test, más del 82% lograron ser clasificados correctamente, es de resaltar que este recall es el más bajo en las cuatro clases, debido a que muchas menos motos pudieron ser encontradas en el set de imágenes de prueba. A su vez la tabla 11 nos muestra

la precisión de la detección con un porcentaje de 100%, el más alto entre todas las 4 clases lo cual nos dice que de todas las detecciones en las que el vehículo fue tomado como moto, son correctas, lo que esto muestra es que los vehículos tipo moto son difícilmente confundidos con alguna de las otras 3 clases.

- En la tabla 12 se muestra la matriz de confusión de los vehículos tipo bus, que nos muestra que, de un total de 65 buses, se lograron detectar 53, lo cual nos da un *recall* de 84.61%, diciéndonos que, del total de buses en el set de imágenes de test, más del 84% lograron ser clasificados correctamente. A su vez la tabla 13 también nos muestra la precisión de la detecciones, con un porcentaje de 88.70% lo cual muestra que de todas las detecciones que se tomaron como buses más del 88% son correctas.
- En la tabla 14 se muestra la matriz de confusión de los vehículos tipo camión, que nos muestra que, de un total de 53 camiones, se lograron detectar 46, lo cual nos da un *recall* de 86.79%, diciéndonos que, del total de camiones en el set de imágenes de test, más del 86% lograron ser clasificados correctamente. A su vez la tabla 15 nos muestra la precisión de la detecciones, con un porcentaje de 86.79% lo cual muestra que de todas las detecciones que se tomaron como buses más del 88% son correctas.

Algoritmo de tracking.

- El algoritmo de tracking implementado cumple con su propósito principal el cual es seguir el vehículo durante el 80% de los frames del video. Resultado de esto son los números arrojados por las gráficas 7 a al 17, que nos muestran un buen resultado no solo en la detección de las 4 diferentes tipos de vehículos, si no que también muestra que el algoritmo de tracking está funcionando correctamente, ya que de no hacerlo, los resultados no serían ni cercanos a los aforamientos realizados manualmente.

Algoritmo conteo de vehículos.

- En la tabla 16 se observa el aforamiento vehicular manual y el realizado por el sistema, clasificando tipos de vehículo y ruta tomada por los vehículos. Donde se puede extraer un MSE *(Mean Squared Error)* que muestra qué tan lejos está el estimador - el sistema - de los resultados reales dando mayor relevancia a los errores grandes que a los pequeños, *para* los autos livianos el MSE es del 1.75, para las motos el del 74.5 y para el de los pesados es del 0.75. En este tipo de media el que mejor se ajusta a los datos reales es el tipo de auto pesados.

  De la tabla 16 también se extrae la métrica de RMSE (*Root-Mean-Square Error*) el cual amplifica y penaliza con menor fuerza que el MSE aquellos errores de mayor magnitud. Para los datos de la tabla 16 se encontró que el RMSE de los livianos es 1.32, para las motos es de un 8.63 y finalmente para los pesados es de 0.87.

  Finalmente se extrae de la tabla 16 el MAE (*Mean Absolute Error*), para los vehículos tipo livianos es 1.25, para las motos es de un 8.5 y por último para los camiones es de un 0.75.

- En la tabla 17 se observa el aforamiento vehicular manual y el realizado por el sistema, clasificando tipos de vehículo y ruta tomada por los vehículos. Donde se

puede extraer un MSE *(Mean Squared Error)* que muestra qué tan lejos está el estimador - el sistema - de los resultados reales dando mayor relevancia a los errores grandes que a los pequeños, *para* los autos livianos el MSE es del 29.67, para las motos el del 204.17 y para el de los pesados es del 0.75. En este tipo de media el que mejor se ajusta a los datos reales es el tipo de auto pesados.

De la tabla 17 también se extrae la métrica de RMSE (*Root-Mean-Square Error*) el cual amplifica y penaliza con menor fuerza que el MSE aquellos errores de mayor magnitud. Para los datos de la tabla 16 se encontró que el RMSE de los livianos es 5.45, para las motos es de un 14.29 y finalmente para los pesados es de 0.87.

Finalmente se extrae de la tabla 17 el MAE (*Mean Absolute Error*), para los vehículos tipo livianos es 3.33, para las motos es de un 8.0 y por último para los camiones es de un 0.75.

- En la tabla 18 se observa el aforamiento vehicular manual y el realizado por el sistema, clasificando tipos de vehículo y ruta tomada por los vehículos. Donde se puede extraer un MSE *(Mean Squared Error)* que muestra qué tan lejos está el estimador - el sistema - de los resultados reales dando mayor relevancia a los errores grandes que a los pequeños, *para* los autos livianos el MSE es del 9.25, para las motos el del 28.0 y para el de los pesados es del 4.37. En este tipo de media el que mejor se ajusta a los datos reales es el tipo de auto pesados.

  De la tabla 18 también se extrae la métrica de RMSE (*Root-Mean-Square Error*) el cual amplifica y penaliza con menor fuerza que el MSE aquellos errores de mayor magnitud. Para los datos de la tabla 16 se encontró que el RMSE de los livianos es 30.41, para las motos es de un 5.29 y finalmente para los pesados es de 6.614.

  Finalmente se extrae de la tabla 18 el MAE (*Mean Absolute Error*), para los vehículos tipo livianos es 1.5, para las motos es de un 2.37 y por último para los camiones es de un 3.125.

Conclusiones

- Se logra poder integrar un *GUI* al sistema, la cual permite etiquetar en los videos y/o imágenes las clases necesarias para el entrenamiento, logrando un acople perfecto con el algoritmo de detección, respetando en el set de datos etiquetado los formatos necesarios, archivos y ubicaciones requeridas para llevar a cabo el entrenamiento.
- Se logran niveles de detección muy aceptables para las clases de livianos, buses y camiones, estos tres con recalls y precisiones bastante altas, esto es dado que el tamaño en escena de estos 3 tipos de vehículos es la suficiente mente grande com para que el algoritmo de detección pueda extraerlos y clasificarlos de un frame con bastante ruido e información no necesaria.

- El tipo de vehículo motos, presenta el *recall* más pequeño de las 4 clases de interés, esto es debido principalmente al tamaño en escena, lo que hace que sea aun mas dificil de encontrar y clasificar. También es de resaltar que este tipo de vehículos se ve muy afectado principalmente en este tipo de situaciones, como lo son: transitar por una zebra vial, transitar debajo semáforos, bicicletas en escena y sombras pronunciadas.
- Los camiones y buses en ocasiones presentan confusiones entre ellos y/o vehículos livianos, esto debido a que hay autos "Livianos" que llegan a ser muy parecidos a un microbus o camiones tipo jaula pequeños.
- En el entrenamiento de YOLOv3 se logra percibir un punto de saturación, en donde las detecciones no logran mejorar, a pesar de que se realicen esfuerzos en aumentación de datos en diferentes escenas o vídeos.
- Se logra acoplar perfectamente la salida del detector, para esta ser utilizada como entrada en el algoritmo de tracking, además de esto se logra confiar al trackin el 80% de los frames del video, para lograr un mejor desempeño y bajar el esfuerzo computacional requerido.
- El algoritmo de tracking logra reconocer puntos clave como los son el de entrada a escena de un vehículo y el punto de salida de este, insumo necesario para el algoritmo de conteo de vehículos o aforamiento.
- El comportamiento general del sistema es satisfactorio, dados las restricciones de hardware dadas por la empresa por tema monetario, pero aun así se logra de manera más automática realizar aforos vehiculares, lo que logra mejorar en tiempos la productividad de la empresa en esta área. También es bueno resaltar que dado los resultados generados por este proyecto, la empresa ha emprendido una migración de sus anteriores tecnologías de detección en imágenes a la utilizada e implementada en este proyecto YOLOv3.

Referencias Bibliográficas

1. Inter - telco S.A.S. (n.d.). Retrieved April 22, 2019, from https://www.inter-telco.com/.
2. YOLO: Real Time Object Detection. (n.d.). Retrieved April 22, 2019, from https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection.
3. Darknet: Open Source Neural Networks in C. (n.d.). Retrieved April 23, 2019, from https://pjreddie.com/darknet/.
4. Open Source Computer Vision Library. (n.d.). Retrieved April 23, 2019, from https://github.com/opencv/opencv.
5. Machine Learning is Fun! (n.d.). Retrieved April 23, 2019, from https://medium.com/@ageitgey/machine-learning-is-fun-80ea3ec3c471.
6. Car Localization and Counting with Overhead Imagery, an Interactive Exploration. (2017, Marzo 15). Retrieved April 23, 2019, from https://medium.com/the-downlinq/car-localization-and-counting-with-overhead-imagery-an-interactive-exploration-9d5a029a596b.
7. Vehicle detection and tracking in video. (2002, August 6). Retrieved April 23, 2019, from https://ieeexplore.ieee.org/abstract/document/900967.

8. 3D vehicle detection using a laser scanner and a video camera. (2018, June 2). Retrieved April 23, 2019, from https://digital-library.theiet.org/content/journals/10.1049/iet-its_20070031.
9. Sistemas de inteligencia en red. (n.d.). Retrieved April 23, 2019, from https://sistemasinteligentesenred.com.co/.
10. Kineo S.A.S. (n.d.). Retrieved April 23, 2019, from https://www.kineo.es/prow/.
11. Vehicle Detection in Aerial Images Based on Region Convolutional Neural Networks and Hard Negative Example Mining. (2017, February 10). Retrieved April 23, 2019, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5335960/.
12. Vehicle Detection and Tracking in Car Video Based on Motion Model. (2011, June 2). Retrieved April 23, 2019, from https://ieeexplore.ieee.org/abstract/document/5723749.
13. Mavic. (n.d.). Retrieved April 23, 2019, from https://www.dji.com/mavic.
14. YOLOv3: An Incremental Improvement. (n.d.). Retrieved April 23, 2019, from https://pjreddie.com/media/files/papers/YOLOv3.pdf.
15. Installing Darknet. (n.d.). Retrieved April 23, 2019, from https://pjreddie.com/darknet/install/#cuda.

Anexos

a. Ipython notebook creado para el análisis de los datos entregados por el sistema. https://colab.research.google.com/drive/1Aky__YekhpLZNdhZdDpap_FUUfteo1k_
b. Resultados de los aforamientos realizados por el sistema y expuestos en este informe. https://docs.google.com/spreadsheets/d/1Bfg1C_wMFhCNauB_CDe309SK9oEZ3PAVOWd8SQW8AFE/edit?usp=sharing
c. Repositorio con el código fuente de los archivos python encargados de realizar el aforamiento. https://github.com/inter-telco/aforosDron