UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA CIVIL

TESIS

“SISTEMA DE DETECCIÓN Y CLASIFICACIÓN VEHICULAR BASADO EN REDES NEURONALES DE APRENDIZAJE PROFUNDO”

PARA OBTENER EL TÍTULO PROFESIONAL DE INGENIERO CIVIL

ELABORADO POR:

DAVID ALEXANDER NIZAMA YAMUNAQUE

ASESOR:

MSc. LEONARDO FLORES GONZALEZ

LIMA - PERÚ

2022

Nizama Yamunaque, David Alexander
dnizamay@uni.pe
990 273 929

DEDICATORIA

A Dios, por su infinita misericordia y fortaleza.
A mi amada esposa Kelly, por su apoyo, paciencia y amor incondicional, a mi hija Melodía, quien es mi razón y motivo de crecer día a día.
A mis padres Jose y Mercedes, por la confianza depositada y por siempre creer en mi.
A mis familiares, maestros y amigos, que siempre me han ofrecido su apoyo, amistad y cariño.

AGRADECIMIENTOS

Un agradecimiento profundo a mi alma mater, la Universidad Nacional de Ingeniería (UNI), por enseñarme que la ingeniería es más que resolver números, es una profesión al servicio del país. Es mi compromiso y promesa como persona dar mi máximo esfuerzo en las labores de ingeniería que desarrolle, dejando siempre en alto el nombre de la UNI y el de todos sus profesionales.

Mi más gran y sincero agradecimiento al Msc. Ing. Leonardo Flores González por la disposición, amistad y confianza brindada, por su paciencia y enseñanzas en el desarrollo de la presente tesis, sin su orientación este trabajo no hubiera sido posible. A mis profesores del pre grado de la Facultad de Ingeniería Civil, quienes con sus conocimientos y dedicación formaron como persona y profesional a quien suscribe estas palabras, mi total agradecimiento a ellos.

A mis padres Jose y Mercedes, por acompañarme en cada etapa de mi vida, por sus consejos de vida, por ser mi soporte, por inculcarme valores y permitirme ser un profesional, a ellos les debo mi vida y gratitud eterna. A mi hermanos, Jose y Yenifer, quienes me cuidaron desde que era un bebé, me aconsejaron y me vieron crecer, les agradezco por estar conmigo cuando los necesité. A mi familia y amigos que han sido parte de mi crecimiento personal y profesional a lo largo de mi vida, reciban mi más sincero agradecimiento.

Finalmente, un agradecimiento muy especial a la familia con la que Dios me bendijo, mi esposa Kelly y mi hija Melodía, quienes son la razón de mi vida y existir, gracias por acompañarme en todo este proceso y darme las fuerzas para seguir adelante, a ustedes les debo mi felicidad y el motivo por el que me levantó todos los días para ser mejor persona.

ÍNDICE

RESUMEN

La presente tesis presenta un sistema de detección, clasificación y conteo vehicular mediante el uso de redes neuronales de aprendizaje profundo y el de la inteligencia artificial, específicamente del área de visión artificial, con el objetivo de realizar conteos y aforos vehiculares. Para ello, se utilizarán diversos algoritmos y redes neuronales previamente entrenadas que permiten dotar al ordenador la capacidad de ver y comprender el contenido de imágenes y videos a través del reconocimiento de patrones y características.

El aforo vehicular es uno de los aspectos más importantes y primarios en un estudio de tráfico vehicular, debido a que a partir de estos datos se determina el grado de ocupación y condiciones en las que una vía funciona, así como, las futuras tendencias de crecimiento, lo que permite una correcta planificación y diseño de una construcción, rehabilitación o mejora de una obra vial. Por ello, la finalidad del presente trabajo de investigación es brindar una alternativa accesible, rentable y económica que permita realizar aforos vehiculares en una vía mediante las virtudes de la inteligencia artificial, las cuales en los últimos años han tenido un desarrollo y progreso destacable.

Para la detección y clasificación de vehículos se utilizaron las redes neuronales convolucionales, las cuales están diseñadas para imitar la corteza visual del cerebro y reconocer objetos en imágenes y videos. Estás redes contienen una serie de capas jerarquizadas y especializadas que permiten identificar y diferenciar un objeto de otro, por lo que fue factible clasificar los vehículos de acuerdo a su tipología, esto aportó a obtener una data completa y confiable.

Palabras claves: *Inteligencia artificial, visión artificial, redes neuronales, aprendizaje profundo, aforo vehicular, estudio de tráfico vehicular, algoritmo de detección, reconocimiento de patrones, convoluciones.*

ABSTRACT

This thesis presents a vehicle detection, classification and counting system through the use of deep learning neural networks and artificial intelligence, specifically in the area of artificial vision, with the objective of performing vehicle counting and gauging. For this purpose, several algorithms and previously trained neural networks will be used to provide the computer with the ability to see and understand the content of images and videos through the recognition of patterns and features.

The vehicle capacity is one of the most important and primary aspects in a study of vehicular traffic, because these data determine the degree of occupation and conditions in which a road works, as well as future growth trends, which allows a correct planning and design of a construction, rehabilitation or improvement of a road work. Therefore, the purpose of this research work is to provide an accessible, profitable and economical alternative that allows to perform vehicular gauging on a road through the virtues of artificial intelligence, which in recent years have had a remarkable development and progress.

For the detection and classification of vehicles, convolutional neural networks were used, which are designed to imitate the visual cortex of the brain and recognize objects in images and videos. These networks contain a series of hierarchical and specialized layers that allow to identify and differentiate one object from another, so it was feasible to classify vehicles according to their typology, this contributed to obtain a complete and reliable data.

Keywords: *Artificial intelligence, computer vision, neural networks, deep learning, vehicular gauging, vehicular traffic study, detection algorithm, pattern recognition, convolutions.*

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS Y SIGLAS

SÍMBOLOS

$\star$	:	Operador de convolución.
$\left[A_{ij}\right]$	:	Matriz A con *i* filas y *j* columnas.
$\sum$	:	Sumatoria.
$f(x)$	:	Función de x.
max	:	Función máximo.
$\geqslant x$	:	Mayor o igual que x
$< x$	:	Menor que x
(x, y)	:	Coordenadas x, y
$[i]$	:	Indice del objeto i
γ_i	:	Confianza del objeto i
γ_{min}	:	Confianza mínima
Q_{max}	:	Flujo máximo

SIGLAS

.py	:	Formato de un script en código Python
.xls	:	Formato de un archivo de Excel
BGR	:	Blue, green and red
COCO	:	*Common Object in Context* o Objeto Común en Contexto
CPU	:	*Central Processing Unit* o Unidad de Procesamiento Central
CV	:	*Computer Vision* o Visión Artificial
DL	:	*Deep Learning* o Aprendizaje Profundo
DROI	:	*Detection Region of Interest* o Región de Detección de Interés
FHMD	:	Factor Horario de Máxima Demanda
IA	:	Inteligencia Artificial
GPU	:	*Graphics Processing Unit* o Unidad de Procesamiento Gráfico
LC	:	Línea de Conteo
ML	:	*Machine Learning* o Aprendizaje Automático
OpenCV	:	*Open Computer Vision Library* o Biblioteca abierta de Visión Artificial
QR	:	*Quick Response* o Código de barras
ReLU	:	*Rectified Linear Unit* o Unidad Lineal Rectificada
RGB	:	Red, green and blue
RNA	:	Redes Neuronales Artificiales
RNC	:	Redes Neuronales Convolucionales
UCP	:	Unidades Coche Patrón
UNI	:	Universidad Nacional de Ingeniería
Yolo	:	You Only Look Once

CAPÍTULO I: INTRODUCCIÓN

1.1 GENERALIDADES

En los últimos años, las ciudades han tenido un crecimiento explosivo tanto poblacional como de uso de suelo trayendo consigo el aumento del parque automotor y el de la oferta y demanda de transporte. Por tal razón, los propietarios y concesionarios viales han destinado mayores esfuerzos en la construcción, ampliación y modernización de su infraestructura vial a fin de ofrecer un servicio seguro y eficiente. Para ello, realizan estudios de flujo vehicular que les permitan conocer el grado de ocupación y condiciones en las que una vía funciona, así como definir las tendencias de crecimiento y el tiempo a partir del cual dejará de prestar un servicio adecuado.

El conteo y clasificación vehicular tienen un papel fundamental dentro del estudio de tráfico ya que a partir de estos datos se realiza el diagnostico de una vía. Para obtener estos datos comúnmente se realiza una aforo manual en la que el conteo y clasificación es realizado por un grupo de personas que son puestas in situ durante largas horas del día, multiplicándose la cantidad de personas por la cantidad de lugares, intersecciones o vías se desea estudiar.

Últimamente, el campo de la Inteligencia Artificial (IA) ha generado mucho interés por su potencial en la simulación de procesos de la inteligencia humana, como aprender, ver, razonar y retroalimentarse a sí mismo. Sus disciplinas de Machine Learning (*Aprendizaje Automático*) y Deep Learning (*Aprendizaje Profundo*) han impulsado la solución de diversos problemas en el campo computacional debido a su potencial y rendimiento que ofrecen en la automatización de procesos. La presente investigación propone aplicar técnicas de IA que permitan automatizar el proceso de detección y clasificación vehicular a través del área de visión computacional.

1.2 DESCRIPCIÓN DEL PROBLEMA DE INVESTIGACIÓN

Los problemas de tráfico, como la movilidad, seguridad y el consumo eficiente, han sido siempre una preocupación entre la población. Esto ha generado que en los últimos años aumente el interés por disminuir o eliminar estos problemas. La saturación de las vías ocasiona que se formen en las intersecciones los denominados cuellos de botella, lo cual aumenta el tiempo de permanencia en el sistema y alargue de las horas de viaje, evitando el aprovechamiento en actividades productivas, mientras que en países en vías de desarrollo tratan de que las horas de transporte de la población sean aprovechadas al máximo (Montenegro, 2018).

Las empresas o entidades dedicadas a la concesión vial necesitan de herramientas para elaborar estudios de tráfico que les permita planificar y diseñar correctamente una construcción, rehabilitación o mejora de una obra vial. Para ello, se requiere de datos de flujo vehicular en la vía a fin de pronosticar la demanda futura, por lo que, una forma de hacerlo es que dicha toma de datos y análisis sea realizado por un grupo considerado de personas en campo, lo que supone una inversión elevada de recursos financieros y logísticos (Peña, 2017).

Si bien es cierto que el proceso de recolección de datos comúnmente se realiza de forma manual, existe una alternativa poca profundizada que es el campo computacional. En los últimos años, por la evolución de los procesadores y creación de computadoras más rápidas y autónomas, la inteligencia artificial a impulsado la solución de diversos problemas más complejos debido a las bondades y rendimiento que ofrecen sus técnicas de Machine Learning y Deep Learning. La tarea consiste en aplicar estas técnicas a imágenes digitales y videos con el fin de detectar y clasificar (reconocer) los objetos que aparecen en ella. La automatización de dicho proceso podría aumentar la efectividad y reducir los costes en la determinación de la demanda de una vía.

1.3 OBJETIVOS DEL ESTUDIO

1.3.1 Objetivo General

Desarrollar un sistema automatizado de detección y clasificación vehicular basado en redes neuronales de Aprendizaje Profundo que permita realizar aforos vehiculares a través del uso de videograbaciones.

1.3.2 Objetivos Específicos

- Identificar modelos de redes neuronales de Aprendizaje Profundo, entrenados previamente, para la predicción y detección de objetos.
- Desarrollar un algoritmo basado en Visión Artificial que en conjunto con los modelos de redes neuronales permita la detección y clasificación vehicular a través del procesamiento de imágenes y videos digitales.
- Procesar los modelos de redes neuronales en el algoritmo de Visión Artificial para detectar y clasificar vehículos en videograbaciones de tránsito.
- Determinar el aforo vehicular de una vía haciendo uso de redes neuronales artificiales y algoritmos de visión artificial.
- Analizar y comparar los resultados obtenidos de los aforos vehiculares con los diferentes modelos de redes neuronales.

- Determinar el rendimiento y precisión de resultados de cada una de las redes neuronales utilizadas.

1.4 HIPÓTESIS DEL ESTUDIO

1.4.1 Hipótesis General

Automatizar la detección y clasificación vehicular basado en redes neuronales de aprendizaje profundo permite recolectar datos de tránsito de vehículos según su tipología para un aforo vehicular.

1.4.2 Hipótesis Específicas

- Es viable desarrollar un sistema de detección y clasificación vehicular basado en redes neuronales de aprendizaje profundo.
- Con modelos entrenados de redes neuronales de Aprendizaje Profundo es posible predecir y detectar objetos.
- Un algoritmo basado en Visión Artificial puede detectar y clasificar vehículos a través del procesamiento de videos digitales.
- Es posible procesar modelos de redes neuronales, previamente entrenados, en algoritmos de Visión Artificial para detectar y clasificar vehículos en videograbaciones de tránsito.
- Es factible obtener un aforo vehicular en una vía haciendo uso de redes neuronales artificiales y algoritmos de visión artificial.

CAPÍTULO II: FUNDAMENTO TEÓRICO Y CONCEPTUAL

En el estudio de la Inteligencia Artificial (IA), las Redes Neuronales Convolucionales (RNC) es un subcampo de las disciplinas de Machine Learning (ML) y Deep Learning (DL). La presente tesis comprende la aplicación de las RNC en el desarrollo de un sistema de detección y clasificación vehicular. En ese sentido, la revisión bibliográfica está comprendida en dos partes. En la primera, se abordará la evolución de la IA y su influencia en el desarrollo de las RNC y en la segunda se buscará comprender el funcionamiento de una RNC y su aplicación.

2.1 INTELIGENCIA ARTIFICIAL

Definir la Inteligencia Artificial (IA) es una tarea compleja, sobre todo, porque el concepto en sí depende de la definición de inteligencia, que al día de hoy sigue teniendo múltiples definiciones. Lasse (2018) la define como la capacidad que tienen las máquinas para procesar diversos algoritmos, aprender de ello y utilizar lo aprendido para la toma de decisiones tal y como lo hace un ser humano, con la diferencia que no necesitan descansar y pueden analizar grandes cantidades de información a la vez.

La IA se basa en el funcionamiento de la inteligencia humana y, en su efecto, en la del cerebro. El cerebro humano es el sistema de reconocimiento de patrones más complejo y eficiente que se conoce, su complejidad escala a niveles tales que, sus funciones cognitivas se realizan mediante la activación coordinada de más de 90,000,000,000 células nerviosas interconectadas mediante enlaces sináptico (Benítez et al., 2014). El objetivo de la IA es que los ordenadores realicen las mismas competencias que puede realizar el cerebro humano, como razonar, predecir, percibir, planificar, etc. La inteligencia humana no es una dimensión única sino un espacio profundamente estructurado de diversas capacidades para procesar la información, del mismo modo, la IA utiliza diferentes técnicas para resolver una gran variedad de tareas (Boden, 2017).

2.1.1 Machine Learning - Aprendizaje Automático

El Aprendizaje Automático (*del inglés, Machine Learning*) es un enfoque de la IA, en la que los ordenadores o las máquinas tienen la capacidad de aprender por sí solas y auto programarse aprendiendo de su propia experiencia, mejorando su rendimiento en función a los resultados anteriores de manera automática (Lasse, 2018).

2.1.2 Redes Neuronales Artificiales

Las millones de neuronas interconectadas del cerebro humano es una estructura compleja y maravillosa capaz de transmitir información entre sus diferentes neuronas a través del proceso de sinapsis. Las neuronas biológicas están compuestas de tres componentes: el cuerpo de la célula, las dendritas, que actúan como canales de recepción, y el axón, que es el canal de transmisión. La sinapsis se produce en el punto de unión de una dendrita con el axón de otra célula. Al existir un estímulo (o impulso eléctrico) en estás conexiones se producen intercambios de sustancias, reacciones químicas y liberación de neurotransmisores, los cuales, dependiendo de su tipo, pueden excitar o inhibir a las células receptoras (Becerra, 2014).

Inspirándose en el comportamiento y funcionamiento de las redes neuronales biológicas se crearon modelos matemáticos capaces de solucionar problemas difíciles mediante el uso de técnicas algorítmicas, a estos modelos se le llamaron Redes Neuronales Artificiales (RNA). Las RNA se organizan en capas y son capaces de reconocer relaciones y patrones complejos y a partir de ellos aproximarlas a funciones matemáticas y predecir resultados (Vector ITC, 2018). Para que las RNA tengan un funcionamiento efectivo requieren de una gran cantidad de información y una potente capacidad de procesamiento, así podrá predecir con mayor eficacia (Lasse, 2018).

Las unidades básicas de las RNA son las neuronas, las cuales imitando a las neuronas biológicas se conectan entre sí y trabajan en conjunto recibiendo cada una de ellas una serie de entradas, con cierto peso, y emitiendo una salida que se convertirá en una próxima entrada. El desafío está en encontrar los valores y combinaciones de parámetros que mejor se ajusten a los resultados, para ello existe el proceso de entrenamiento (aprendizaje) de la red neuronal cuya exactitud definirá la precisión del algoritmo. El proceso es iterativo y secuencial, donde la neurona va reforzando ciertas conexiones y en cada iteración va ajustando la relación de pesos hasta cierto punto donde el error de los resultados sea menor al establecido por el usuario. Una vez entrenada la red es posible utilizarla para realizar predicciones o clasificaciones (Vector ITC, 2018).

2.1.3 Deep Learning - Aprendizaje Profundo

El Aprendizaje Profundo (*del inglés, Deep Learning*) es un subcampo de Machine Learning que utiliza modelos organizados en capas (redes neuronales) para predecir relaciones y patrones de datos disponibles. Su aplicación requiere de una gran cantidad de información y una potente capacidad de procesamiento. Entre sus principales aplicaciones se tiene el reconocimiento de voz, la visión artificial, los sistemas de asistencia automatizados, entre otros (Lasse, 2018).

En Deep Learning, el término *profundo* se refiere a la cantidad de capas de una red neuronal. Una red con más de tres (3) capas podría considerarse un algoritmo de aprendizaje profundo.

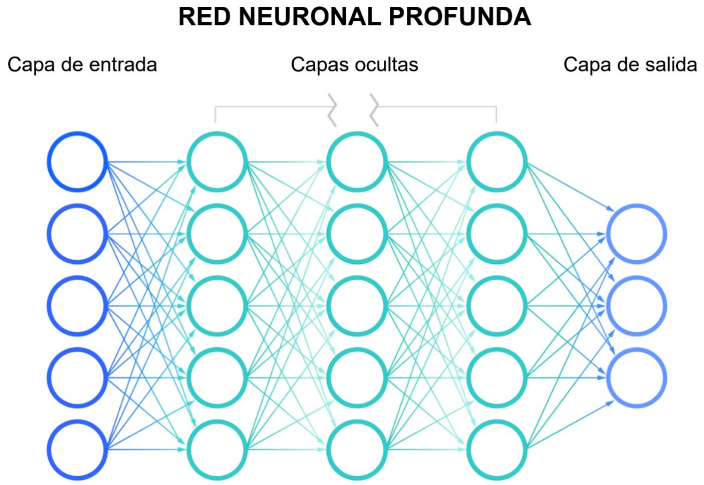


FIGURA N° 2.1: Red Neuronal de Aprendizaje Profundo. Adaptado de IBM Cloud Education (2020)

Es común que muchos utilicen indistintamente o confundan los términos Inteligencia Artificial, Machine Learning, Redes Neuronales y Deep Learning; sin embargo, cada uno de ellos desempeña un rol fundamental dentro del otro (IBM Cloud Education, 2020). Una forma fácil de pensar en ellos y diferenciarlos es imaginándolos como una matrioshka, o también llamada *muñeca rusa*, que es un conjunto de muñecas que en su interior albergan una muñeca más pequeña, esta a su vez a otra y así sucesivamente, siendo cada una esencialmente una componente de la anterior. Es así que, dentro del campo de la Inteligencia Artificial encontraremos el subcampo de Machine Learning y este a su vez contiene el subcampo del Deep Learning, el cual utiliza un tipo de Redes Neuronales como columna vertebral de sus algoritmos (IBM Cloud Education, 2020).

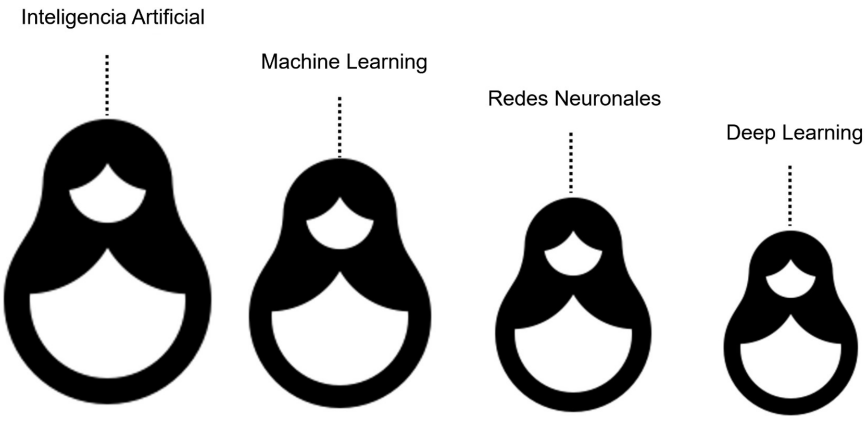


FIGURA N° 2.2: Relación entre Inteligencia Artificial, Machine Learning, Redes Neuronales Artificiales y Deep Learning. Adaptado de IBM Cloud Education (2020)

2.1.4 Bibliotecas de Deep Learning

Existen bibliotecas de Deep Learning de código abierto especializadas en construir, entrenar y evaluar redes neuronales. Entre las bibliotecas más utilizadas por la comunidad de investigadores tenemos a PyTorch y TensorFlow/Keras.

2.1.5 Tipos de aprendizaje automático

Según Lasse (2018), existen diferentes formas de llevar a cabo el aprendizaje automático de las redes neuronales, estas se pueden clasificar en: aprendizaje supervisado, no supervisado y de refuerzo. Para explicar la diferencia de estas tres clases utilizaremos el siguiente ejemplo: Imaginemos que tenemos que organizar 5 000 fotografías de diferentes tipos de vehículos y los algoritmos tienen que identificar las fotos en las que aparecen buses.

- Aprendizaje supervisado: El algoritmo utiliza datos previamente etiquetados u organizados y utiliza este aprendizaje para predecir cómo tendría que ser clasificada la nueva información. En nuestro ejemplo, introduciríamos al algoritmo fotos etiquetadas y le enseñaríamos en cuales aparecen buses, a partir de ello, la red identifica patrones de los datos (tamaño, forma, etc.) y realiza predicciones en fotos similares siendo corregido por el operador.
- Aprendizaje no supervisado: Con este método el algoritmo debe encontrar la manera de clasificar la información por sí mismo a partir de la identificación de patrones en los datos. No utiliza información previamente etiquetada u organizada, por lo que, no interviene el humano. Para el ejemplo, el algoritmo debería clasificar por sí solo todas las fotos en diferentes categorías (buses, autos, motos etc.)
- Aprendizaje por refuerzo: Muy similar al proceso de ensayo y error, el algoritmo aprende de la experiencia, en la que se les da un refuerzo positivo cada vez que acierta. Se puede comparar a cuando les damos una recompensa a un perro cuando aprende a saltar sobre un valla.

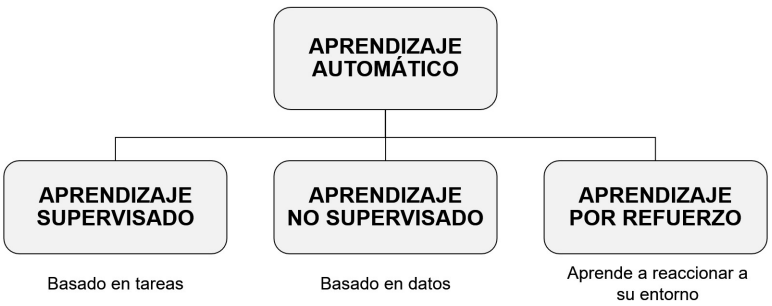


FIGURA N° 2.3: Tipos de aprendizaje automático. Adaptado de Lasse (2018)

2.1.6 Computer Vision - Visión Artificial

El término Visión Artificial (*del inglés, Computer Vision*) es un campo de la IA que consiste en dotar a los ordenadores la capacidad ver y comprender el contenido de imágenes y vídeos a través del reconocimiento de patrones y características. A simple vista puede parecer una tarea sencilla, ya que el ser humano aprende a diferenciar objetos a una temprana edad; sin embargo, a nivel computacional la matemática asume un papel fundamental, sobre todo el álgebra lineal, ya que a través de diversos algoritmos transforma y extrae información valiosa de las imágenes.

2.1.7 Bibliotecas de Visión Artificial

Una de las bibliotecas de visión artificial y de código abierto más utilizadas es The Open Computer Vision Library (llamado también *OpenCV*). La librería OpenCV proporciona herramientas de alto nivel para el análisis y procesamiento de imágenes y videos haciendo uso de la visión aritificial (Cea, Pincheira, y Figueroa, 2018).

2.2 REDES NEURONALES CONVOLUCIONALES (RNC)

2.2.1 ¿Qué son las RNC?

Un caso particular del Deep Learning y las RNA son las Redes Neuronales Convolucionales (RNC). Estas redes están diseñas para imitar a las neuronas de la corteza visual del cerebro y dotarle a un ordenador la capacidad de *ver*; esto las ha convertido en una de las técnicas más populares dentro del campo de Machine Learning para resolver problemas de visión artificial a través del reconocimiento de objetos en imágenes y vídeos (Núñez, 2016).

Las RNC son un tipo de red con aprendizaje supervisado que contiene varias capas ocultas jerarquizadas y especializadas con un objetivo en sí, es decir, a modo de ejemplo, las primeras capas pueden detectar líneas, curvas, formas y mientras se va llegando a capas más profundas estas se especializan en reconocer formas mucho más complejas como un rostro, autos, animales, objetos, etc (Bagnato, 2020).

2.2.2 Funcionamiento de las RNC

Al igual que toda RNA, las RNC reciben una entrada, en este caso una imagen, que puede ser representado en forma matricial multidimensional y que será transformada a través de una serie de filtros (kernels). Una imagen tiene ancho (*número de columnas*) y altura (*número de filas*), como una matriz, pero a diferencia de las matrices tradicionales, las imágenes tienen *profundidad*, que es representado por sus canales de color que contienen los valores numéricos de sus píxeles. Una imagen a color RGB (*red, green, blue*) tiene una profundidad de tres canales: rojo, verde y

azul y una a escala de grises, solo uno; estos tienen píxeles con valores que van del 0 al 255, los cuales, para un mejor procesamiento, conviene convertirlos a valores entre 0 y 1, para ello cada píxel debe ser dividido entre 255.

2.2.3 Convoluciones

El proceso distintivo de las RNC son sus llamadas *convoluciones*, el cual puede parecer un término complicado, pero en la práctica no lo es tanto. Como ejemplo, si alguna vez aplicó un difuminado, suavizado o incluso un enfoque en alguna imagen, podemos decir que usted aplicó una convolución. En términos matemáticos, una convolución es la suma de términos de la matriz resultante de la multiplicación simple de términos de dos matrices (una matriz indica una región de una imagen y la otra, un *kernel*). Un kernel, o también llamado *filtro*, lo podemos definir como una matriz pequeña (en comparación a la matriz que genera una imagen) que se utiliza para enfocar, desenfocar, suavizar, detectar líneas, formas, bordes, entre otros, y que se desplaza de 1 pixel por todas la neuronas de entrada, de izquierda a derecha y de arriba a abajo, obteniendo una nueva matriz de salida (nueva imagen), que sería una nueva capa de neurona oculta.

$$C = \sum \left[A_{ij} \star B_{ij} \right] \tag{2.1}$$

Donde:

- C : Convolución
- A_{ij} : Región de una imagen
- B_{ij} : Kernel

2.2.4 Comprensión de las convoluciones

Para un mejor entendimiento, la Figura N° 2.4 presenta un ejemplo de convolución de una imagen de 6x6 (*36 neuronas*) con un kernel 3x3, cuya operación se describe en la ecuación 2.2. Para todo caso, se debe colocar el centro del kernel en la coordenada a filtrar *(x,y)*, se multiplican los elementos de la región con los del kernel, se suman los resultados y se ubica el número resultante (*261*, para la Figura N° 2.4) en la misma coordenada *(x,y)*. La Figura N° 2.5 muestra el proceso completo de las convoluciones de la imagen de entrada de 6x6, a la que se aplicó 16 convoluciones para un mismo kernel de 3x3, dando como resultando una matriz de salida de 4x4 (*16 neuronas*). El proceso inicia en la parte superior izquierda de la imagen y continúa desplazándose de izquierda a derecha y de arriba a abajo. El resultado de aplicar un kernel a una imagen de entrada es obtener una matriz de salida que contenga información filtrada de la imagen inicial en la que se identifiquen patrones

que ayuden a reconocer formas u objetos. Para mantener las dimensiones de la matriz de salida igual a la de entrada suelen agregarse pixeles con valores igual a cero en los bordes de la imagen de salida.

FIGURA N° 2.4: Ejemplo gráfico de una convolución con un kernel de 3x3. Adaptado de PyImageSearch Gurus

Ejemplo de una operación de convolución a partir de la Figura N° 2.4:

$$C = \sum_{a_{11}}^{a_{ij}} \left[\begin{bmatrix} 93 & 139 & 101 \\ 26 & 252 & 196 \\ 135 & 230 & 48 \end{bmatrix} \star \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \right] \tag{2.2}$$

$$C = \sum_{a_{11}}^{a_{ij}} \begin{bmatrix} 93 \times (-1) & 139 \times 0 & 101 \times 1 \\ 26 \times (-2) & 252 \times 0 & 196 \times 2 \\ 135 \times (-1) & 230 \times 0 & 48 \times 1 \end{bmatrix} \tag{2.3}$$

$$C = \sum_{a_{11}}^{a_{ij}} \begin{bmatrix} -93 & 0 & 101 \\ -52 & 0 & 392 \\ -135 & 0 & 48 \end{bmatrix} = 261 \tag{2.4}$$

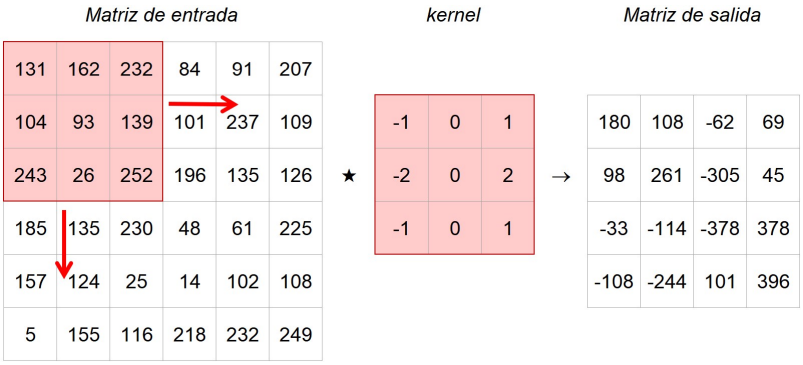


FIGURA N° 2.5: Aplicación de convoluciones a una matriz de entrada de 6x6 con un kernel de 3x3, dando como resultando una matriz de salida de 4x4.

Algo a considerar es que para una misma imagen se pueden aplicar varios kernels (conjuntos de filtros), lo que dará como resultado un conjunto de matrices de salida a las cuales llamaremos *feature mapping*. Es así que, si aplicó 24 filtros para una misma imagen, tendré 24 matrices de salida (feature mapping) con información filtrada de la imagen de entrada que ayudarán en las capas más profundas a distinguir entre un objeto del otro (personas, animales, cosas, vehículos, etc.).

2.2.5 Función de Activación

Similar al proceso de sinapsis de la biología humana, en las RNA existe una *función de activación* que modifica el valor de una neurona para activarla y emita un resultado a la siguiente conexión (sinapsis). Esta función determina si el valor de una neurona (los valores de matriz de salida) supera el limite de un umbral para que sea transmitida a la siguiente capa (Bagnato, 2020). Para el caso de las redes neuronales profundas (RNC), la función de activación *más exitosa* y con mejores resultados es la Rectified Linear Unit (ReLU, Unidad Lineal Rectificada) (Ramachandran, Zoph, y Le, 2017). La ecuación 2.5 define a la función ReLU:

$$f(x) = max(0, x) = \begin{cases} 0 & \text{, si } x < 0 \\ x & \text{, si } x \geqslant 0 \end{cases} \tag{2.5}$$

La Figura 2.6 muestra un ejemplo de aplicación de la función de activación ReLU a la matriz de salida de la 2.5. La función ReLU se encarga de transformar una neurona (píxel), de tal modo que, si el valor de entrada es negativo este se transforma a cero, caso contrato, mantiene su valor. En el ejemplo de la Figura 2.6 los valores de entrada negativos: -62, -305, -33, -114, -378, -108, -244, fueron transformados a cero (0), los demás, se mantuvieron.

180	108	-62	69
98	261	-305	45
-33	-114	-378	378
-108	-244	101	396

$\xrightarrow{f(x) = max(0,x)}$

180	108	0	69
98	261	0	45
0	0	0	378
0	0	101	396

FIGURA N° 2.6: Aplicación de la función de activación ReLU.

2.2.6 Pooling - Agrupación

El *Pooling* es una herramienta estratégica para la reducción del tamaño de una imagen. Su función principal es reducir la cantidad de neuronas (píxeles) y la complejidad de cálculos antes de realizar una nueva convolución. Consiste en agrupar varios píxeles en uno solo, conservando información relevante de la muestra.

Existen diversos tipos de pooling, por ejemplo: Max Pooling (que utiliza el valor máximo de los píxeles de la subregión), Min Pooling (que utiliza el valor mínimo de los píxeles de la subregión) y Average Pooling (que utiliza el valor promedio de los píxeles de la subregión); sin embargo, el Max Pooling es una de las herramientas de muestreo más eficaces utilizadas en las RNC para resumir características sobre una región. Esta operación consiste en encontrar el valor máximo de una muestra y tomar dicho valor como el resumen de las características sobre esa área. Si utilizamos un Max Pooling de 2x2 lo que estaríamos haciendo es reduciendo el tamaño de la imagen en un factor de 2, es decir, a la mitad.

La Figura N° 2.7 muestra un ejemplo de aplicación de Max Pooling de 2x2 realizado a la matriz de salida de la Figura N° 2.6. Para la primera región de 2x2 (en rojo) el valor máximo es 261, para la segunda región (en verde), 69, para la tercera región (en azul), 0 y para la cuarta región (en amarillo), 396. La imagen de entrada de 4x4 se redujo a una de 2x2.

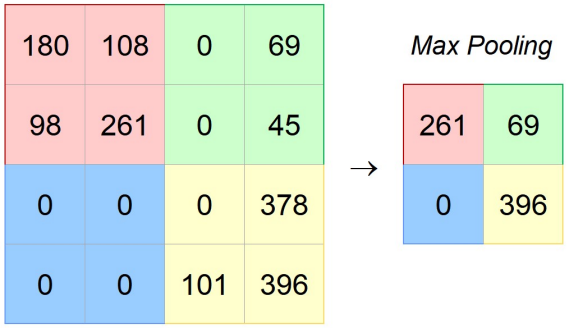


FIGURA N° 2.7: Aplicación de un Max Pooling de 2x2

Finalmente, luego de haber aplicado un kernel de 3x3 y un Max Pooling de 2x2 a nuestra imagen de entrada de 6x6, hemos obtenido una imagen de salida de 2x2 que contiene información filtrada y relevante de la imagen inicial y que nos ayudará a identificar y reconocer patrones de formas u objetos. Tal como se mencionó anteriormente, es posible aplicar más de un kernel (filtro) a una imagen de entrada, algunos de estos pueden ayudarnos a identificar bordes, otros sombras, otros cuervas, otros líneas horizontales, otros verticales, y otros dependiendo de su función. Al aplicar más de un kernel a una imagen esta generará igual cantidad de matrices de salida (feature mapping) lo que aumentará en profundidad nuestras capas neuronales.

2.2.7 Operando una RNC

Entendido el funcionamiento y proceso de las RCN, podemos definir que una convolución en una RNC está compuesta por una imagen de entrada, un conjunto de filtros (kernels), una función de activación y la aplicación de un Max-Pooling. A medida que se hagan más convoluciones, se crearan mayores capas neuronales capaces de identificar formas más complejas, esto se realiza hasta que la capa final pueda clasificar las imágenes encontradas en alguna clase o grupo con cierta probabilidad de certeza.

A continuación, la Tabla N° 2.1 presenta los resultados de aplicar 3 convoluciones a una imagen de 48x48 pixeles en escala gris (48x48x1), utilizando 15 filtros de 3x3 para la primera convolución, 30 filtros de 3x3 para la segunda y 45 filtros de 3x3 para la tercera, además, de un Max Pooling de 2x2 en cada convolución.

TABLA N° 2.1: Proceso de convoluciones a una imagen de 48x48 píxeles

DESCRIPCIÓN	1era Convolución	2da Convolución	3era Convolución
1) Entrada: Imagen	48x48x1	24x24x15	12x12x30
2) Kernel (filtros)	*15 filtros 3x3x1*	*30 filtros 3x3x15*	*45 filtros 3x3x30*
3) Feature Mapping	48x48x15	24x24x30	12x12x45
4) Func. de Activación	ReLU	ReLU	ReLU
5) Max Pooling	de 2x2	de 2x2	de 2x2
6) Salida: Convolución	24x24x15	12x12x30	6x6x45

La imagen de entrada de 48x48 píxeles (48x48x1) tiene un total de 2 304 neuronas, en primera instancia (para la 1era convolución o 1era capa) se le aplicó 15 filtros de 3x3 generando 15 Feature Mapping de 48x48x1 (48x48x15), a ello se aplicó un Max Pooling de 2x2, obteniendo como resultado 15 imágenes de salida de 24x24 píxeles (24x24x15), que serían 8 640 neuronas.

La 2da convolución (2da capa) inicia con las imágenes de salida de la 1era convolución, imágenes de 24x24 píxeles (15 en total) a la que se le aplicarán 30 filtros de 3x3, esto generará 30 Feature Mapping de 24x24 (24x24x30) y aplicándoles un Max Pooling de 2x2 se verá reducida a 30 imágenes de salida de 12x12 (12x12x30), 4 320 neuronas. Como se puede apreciar la 1era convolución inició con 2 304 neuronas y su salida fue de 8 640, pareciera que los cálculos por cada capa aumentan; sin embargo, aplicando una 2da convolución se obtiene que con 8 640 neuronas de entrada se puede obtener 4 320 de salida, la mitad.

La 3era convolución inicia imágenes de 12x12 píxeles (30) a las que le aplicaremos 45 filtros de 3x3, generándose 45 Feature Mapping de 12x12 (12x12x45), y si le

aplicamos un Max Pooling de 2x2 obtendriamos finamente 45 imágenes de salida de 6x6 (6x6x45), que serían 1 620 neuronas. Comparando las neuronas de entrada de la 1era convolución (2 340) con las que se obtienen al final de la 3era (1 620) podemos deducir que las convoluciones lograron reducir la cantidad de neuronas a evaluar con la particularidad que estas neuronas contienen información relevante e importante que nos ayudará a identificar formas (boca, nariz, rostro, orejas, chasis, puertas, etc.) y así poder identificar objetos (personas, vehículos, animales, etc.) de la imagen inicial.

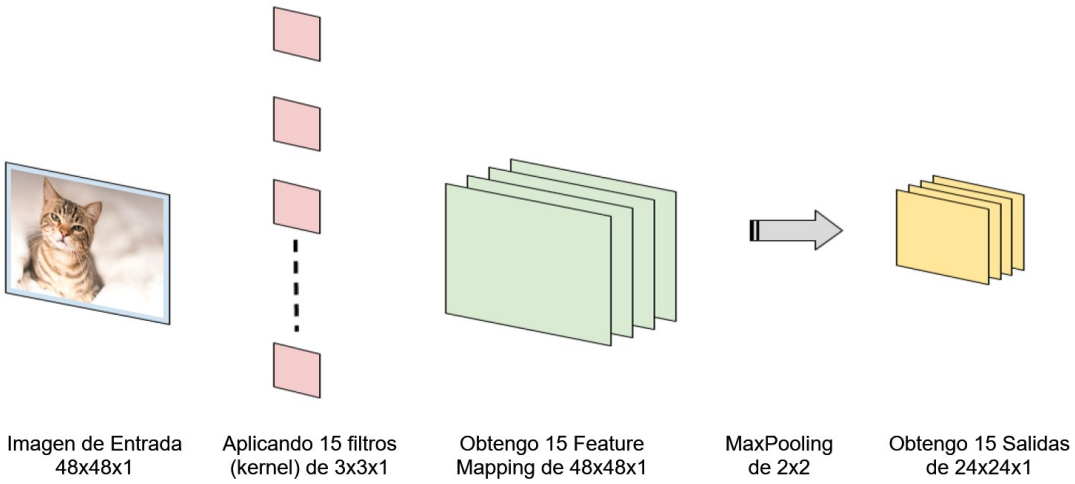


FIGURA N° 2.8: Primera convolución a una imágen de 48x48 píxeles con 15 filtros (Tabla N° 2.1). Adaptado de Bagnato (2020)

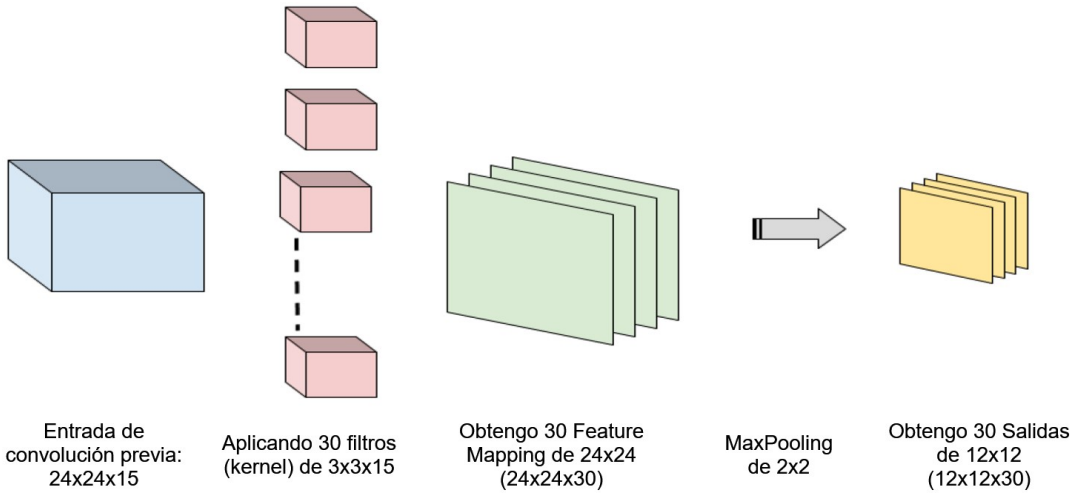


FIGURA N° 2.9: Segunda convolución a una imágen de 48x48 píxeles con 30 filtros (Tabla N° 2.1). Adaptado de Bagnato (2020)

CAPÍTULO III: ESTRATEGIA Y METODOLOGÍA DE SOLUCIÓN

El capítulo presenta los modelos de redes neuronales previamente entrenados y el desarrollo de los algoritmos que serán utilizados para la clasificación y detección vehicular, así como, la estrategia de solución aplicado a imágenes y videos digitales.

3.1 CONSIDERACIONES INICIALES

Para el desarrollo de los algoritmos se utilizó el lenguaje de programación Python y las bibliotecas de código abierto PyTorch y OpenCV. Además, se utilizó el sistema operativo Windows 10, 64 bits.

3.1.1 Instalación de Python

Para instalar Python en su ordenador descargue el instalador desde la página oficial. Para ello, diríjase a https://www.python.org/downloads/, en la pestaña *Downloads* seleccione el sistema operativo de su ordenador y busque la versión Python de su preferencia (para la presente tesis se utilizó Python 3.8.2), luego, descargue el instalador y ejecute como administrador. Finalmente, tendrá instalado Python en su ordenador.

3.1.2 Instalación de OpenCV

Para instalar PyTorch diríjase al cuadro de búsqueda de Windows, escriba «*CMD*» y de click en el icono *«Simbolo del Sistema»*, dentro de él, ejecute el siguiente comando:

```
pip install opencv-contrib-python
```

3.1.3 Instalación de PyTorch

Para instalar OpenCV proceda a abrir el *«Simbolo del Sistema»* y ejecute el siguiente comando:

```
pip install torch torchvision
```

3.1.4 Especificaciones del Procesador utilizado

Los cálculos y procesamiento se realizó en un equipo portátil de las siguientes características: Intel(R) Core(TM) i7-7700HQ (7th Gen), 2.80GHz, RAM 16.0 GB y Tarjeta de video GeForce GTX1050.

3.2 DETECCIÓN Y CLASIFICACIÓN VEHICULAR EN IMÁGENES

En esta sección se desarrollarán los algoritmos para la detección y clasificación de objetos en imágenes, asimismo, se compararán los resultados y eficacia de las redes neuronales utilizadas (en velocidad y aciertos). Para fines de la presente tesis utilizaremos tres (3) modelos de redes: Faster R-CNN (ResNet50), RetinaNet (ResNet50) y YOLOv3 y aplicaremos los algoritmos de autoría del PhD. Adrian Rosebrock desarrollados en su sitio web Pyimagesearch.

A continuación, se presenta el diagrama de flujo del algoritmo utilizado para la detección y clasificación de objetos en imágenes:

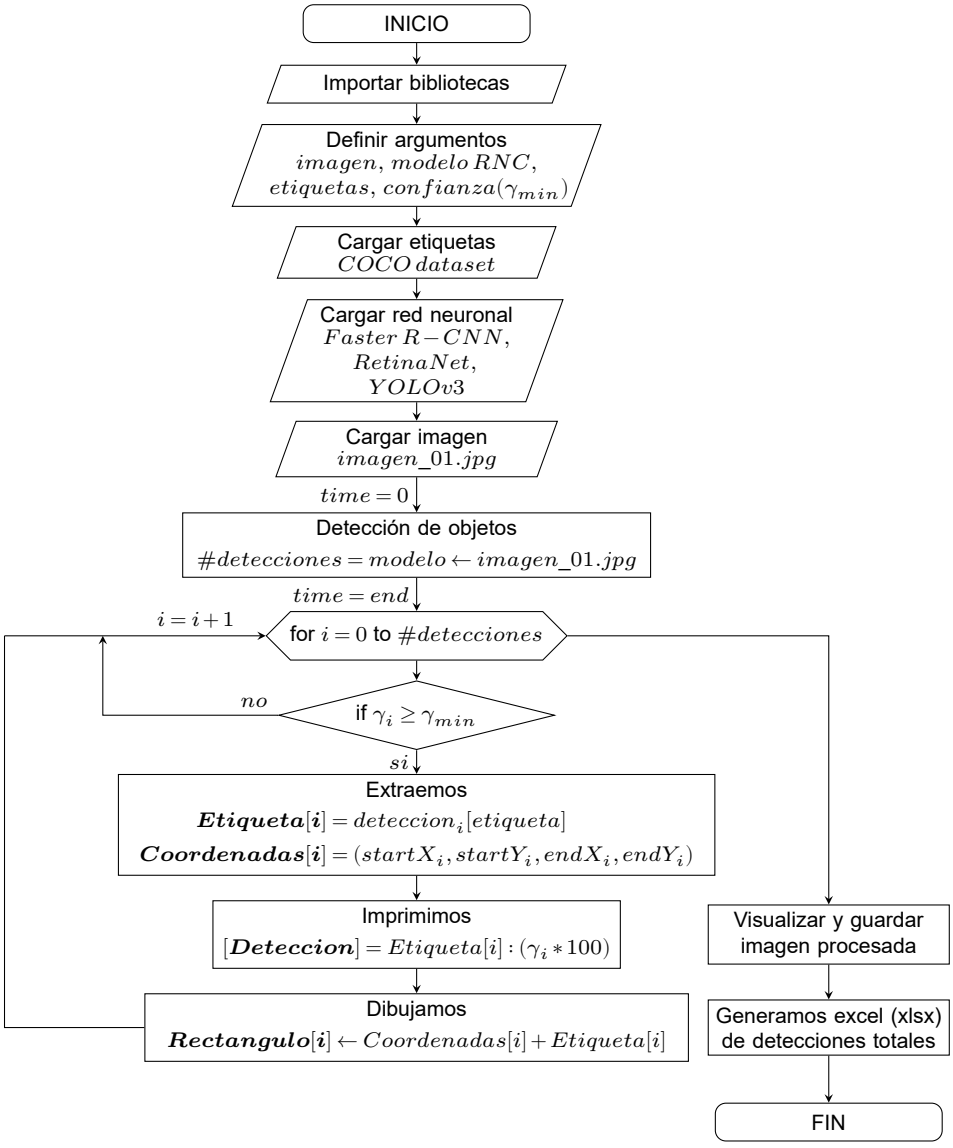


FIGURA N° 3.1: Algoritmo de detección y clasificación de objetos en imágenes

El diagrama de flujo de la Figura N° 3.1 detalla el proceso para detectar y clasificar vehículos en imágenes. En primer lugar, se importan las blibliotecas de Pyhthon y OpenCV y se procede a definir y cargar los argumentos de entrada, como imagen que se evaluará, modelo de red neuronal que se utilizará, etiquetas de clase (carro, bus, camión, motocicleta) y la confianza mínima (γ_{min}) de los objetos para ser considera una detección, los objetos identificados con confianzas menores a la mínima son considerandos detecciones débiles por lo que se proceden a eliminar.

Parámetros de entrada:

```
imagen = Imagen de Entrada #(Ej.: imagen_01.jpg)
modelo RNC = Modelo de Red Neuronal Convolucional #(Ej. YoloV3)
etiquetas = Etiquetas de Clase #(Ej. carro, bus, camión)
Gamma min = Confianza mínima #G min
```

Luego de identificar las detecciones totales se procede a evaluar cada una de ellas, para ello, el algoritmo pasa por un proceso de decisión, se compara la confianza del objeto detectado con la confianza mínima de detección. Si la confianza del objeto detectado (γ_i) es mayor a la confianza mínima (γ_{min}) se continua con el proceso de extracción de etiqueta del objeto (tipo de vehículo) y coordenadas de ubicación; si no, se elimina la detección.

$$Confianza\ (\gamma_i) = \begin{cases} \gamma_i \geq \gamma_{min} & Extrac.: etiqueta, coordenada \\ \gamma_i < \gamma_{min} & EliminaDeteccion \end{cases} \tag{3.1}$$

Luego de extraer la etiqueta de clase y las coordenadas del vehículo se procede a dibujar el cuadro delimitador del objeto detectado y a imprimir los datos de la detección: tipo de objeto detectado y confianza de detección.

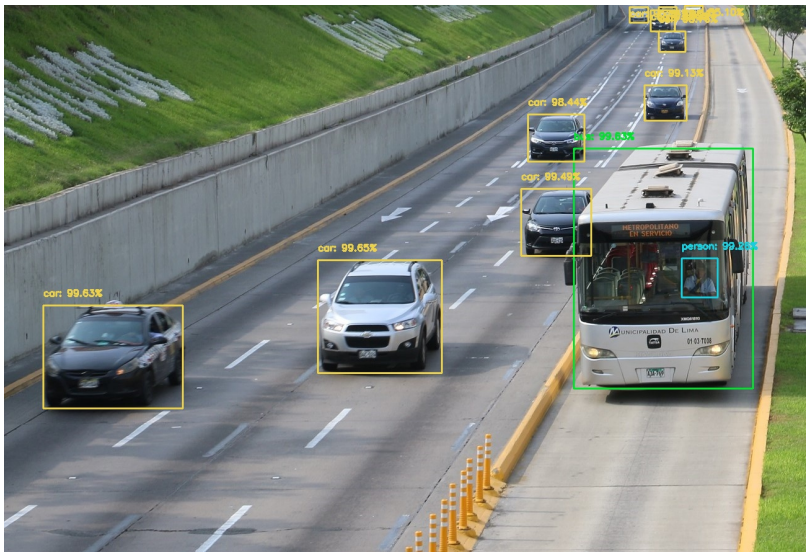


FIGURA N° 3.2: Imagen procesada por el algoritmo de detección y clasificación vehicular

Finalmente, se generará una imagen procesada con cada uno de los vehículos cuyas confianzas de detección fueron superiores a la mínima. Paralelamente, se generará un excel con la data de las detecciones obtenidas.

3.2.1 Algoritmo de detección y clasificación vehicular en imágenes

A continuación, se desarrollará el código fuente del algoritmo de detección y clasificación vehicular en imágenes presentado en el flujograma de la Figura N° 3.1:

- Primero, descargue los archivos del siguiente enlace: https://t.ly/DcMC (o ingresando al código QR de la Figura N° 3.3) y proceda a ubicarlos en una carpeta directorio en su ordenador.

FIGURA N° 3.3: Código QR descarga archivos para detección en imágenes

  El archivo de descarga contiene tres carpetas: *«coco-dataset»*, *«images»* y *«output»* y un script: *«detect _image.py»*. La carpeta *«coco-dataset»* contiene las etiquetas de clase que las redes neuronales pueden identificar. El conjunto de datos COCO (*Common Object in Context*) consta de 80 diferentes categorías, que incluye: carros, buses, camiones, señales de pare, semáforos, entre otros. La carpeta *«images»* contiene una selección de fotografías de flujo vehícular que se utilizarán de entrada en el algoritmo de detección de objetos. La carpeta *«output»* es una carpeta vacía, en ella se guardarán las imágenes procesadas y los archivos excel (.xlsx) generados luego de ejecutar el algoritmo. El script Python *«detect _image.py»* contiene el algormito de la Figura N° 3.1, el cual realiza la detección de objetos en imágenes.

- Luego de descargar y ubicar los archivos en una carpeta directorio de su ordenador, esta quedará conformada de la siguiente manera:

```
#Directorio: Deteccion_Imagenes
|— coco-dataset
|—  |—  coco.names
|— images
|—  |—  example_01.jpg
|—  |—  ...
|—  |—  example_18.jpg
|— output
|— detect_image.py
```

- Procedemos a detallar y explicar las líneas de comandos del script *«detect _image.py»* que nos permitirá detectar y clasificar vehículos en imágenes utilizando redes neuronales pre-entrenadas de PyTorch (Faster R-CNN y RetinaNet):

  Las **líneas 2-11** importan las bibliotecas de Python necesarias para el funcionamiento de nuestro algoritmo. Las **líneas 14-26** definen nuestros argumentos de entrada: ruta de la imagen de entrada, ruta de la imagen de salida, modelo de red neuronal que se utilizará (Faster R-CNN o ReinaNet), ruta de la lista de categorías (etiquetas) y la confianza mínima (γ_{min}) de los objetos para ser considera una detección. La **línea 29** proporciona la opción de elegir el procesador (GPU o CPU) que utilizaremos en la ejecución del algoritmo.

```
1  # Importamos bibliotecas
2  from torchvision.models import detection
3  import numpy as np
4  import argparse
5  import torch
6  import time
7  import cv2
8  import os
9  import pandas as pd
10 from pandas import ExcelWriter
11 from collections import Counter
12
13 # Definimos argumentos (imagen, modelo RNC, etiquetas,
       confianza mínima)
14 ap = argparse.ArgumentParser()
15 ap.add_argument("-i", "--image", type=str, required=True,
16   help="Ruta de la imagen de entrada")
17 ap.add_argument("-o", "--output", type=str, required=True,
18   help="Ruta de la imagen de salida")
19 ap.add_argument("-m", "--model", type=str,
20   choices=["frcnn-resnet", "retinanet"],
21   help="Modelo de red neuronal que utilizará")
22 ap.add_argument("-l", "--labels", required=True,
23   help="Ruta de la lista de categorías (C O C O dataset)")
24 ap.add_argument("-c", "--confidence", type=float, default
       =0.5,
25   help="Confianza mínima para filtrar detecciones débiles")
26 args = vars(ap.parse_args())
27
28 # Elegimos el procesador a utilizar (GPU o CPU)
29 DEVICE = torch.device("cuda" if torch.cuda.is_available()
       else "cpu")
```

Las **líneas 32-34** cargan del directorio el archivo de etiquetas *«coco.names»* y genera un color aleatorio para cada etiqueta. La **línea 37** define una lista vacía «*Lista_Detectada*» en la que guardaremos todas las etiquetas de los objetos detectados. Las **líneas 41-42** define una lista de modelos de red y sus respectivas funciones de llamada PyTorch. Las **líneas 45-46** cargan el modelo pre-entrenado elegido y lo configura en modo evaluación. Si el modelo seleccionado no se encuentra descargado en el ordenador se mostrará una barra de progreso de descarga cuando se ejecute el algoritmo.

```
31 # Cargamos la lista de etiquetas (C O C O dataset) y
       generamos un color aleatorio para cada etiqueta.
32 labelsPath =os.path.sep.join([args["labels"],"coco.names"])
33 CLASSES = open(labelsPath).read().strip().split("\n")
34 COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
35
36 # Definimos una lista para almacenar los objetos detectados
37 Lista_Detectada=[]
38
39 # Especificamos las funciones PyTorch que llaman a los
       modelos de red
40 MODELS = {
41   "frcnn-resnet": detection.fasterrcnn_resnet50_fpn,
42   "retinanet": detection.retinanet_resnet50_fpn }
43
44 # Cargamos el modelo de red que se utilizará y lo
       configuramos en modo evaluación
45 model = MODELS[args["model"]](pretrained=True, progress=True
       , pretrained_backbone=True).to(DEVICE)
46 model.eval()
```

Con el modelo de red cargado, procedemos a leer y procesar nuestra imagen de entrada:

```
48 # Cargamos la imagen de entrada y creamos una copia
49 image = cv2.imread(args["image"])
50 orig = image.copy()
51
52 # Convertimos la imágen BGR a RGB y ordenamos los canales
53 image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
54 image = image.transpose((2, 0, 1))
55
56 # Agregamos una dimensión a la imagen, escalamos sus pixeles
       al rango de [0,1] y la convertimos a un tensor de tipo
       punto flotante
57 image = np.expand_dims(image, axis=0)
58 image = image / 255.0
59 image = torch.FloatTensor(image)
```

Las **líneas 49-50** cargan la imagen de entrada desde su ruta de ubicación y crean una copia de ella para luego dibujar sus predicciones. Debido a que los modelos de red de PyTorch fueron entrenados en imágenes de orden RGB, las **líneas 53-54** ordenan los canales de color de BGR a RGB y los vuelve a reordenar. Las **lineas 57-59** agregan una columna adicional de ceros a nuestra imagen y escalan sus píxeles del rango de [0,255] a [0,1], para ello divide cada valor entre 255. Finalmente, nuestra imagen es convertida en un tensor del tipo punto flotante.

```
61 # Enviamos la imagen al procesador y la pasamos por la red
       para obtener las detecciones
62 image = image.to(DEVICE)
63 start = time.time()
64 detections = model(image)[0]
65 end = time.time()
66
67 # Imprimimos el tiempo que dura el proceso
68 print("[INFO] El proceso duró {:.6f} segundos".format(end -
       start))
```

Las **líneas 62-65** envían la imagen al procesador (GPU o CPU) y la pasa por la red neuronal seleccionada. Las **líneas 63** y **65** controlan el tiempo desde el inicio del proceso hasta que la imagen sale de la red con las detecciones identificadas. La **línea 68** imprime el tiempo total del proceso.

```
70 # Creamos un bucle para recorrer cada detección
71 for i in range(0, len(detections["boxes"])):
72
73   # Extraemos la confianza asociada con la detección
74   confidence = detections["scores"][i]
75
76   # Filtramos la detección asegurándonos que la confianza
       sea mayor que la confianza mínima
77   if confidence > args["confidence"]:
78
79     # Extraemos la etiqueta del objeto detectado y las
       coordenadas de su cuadro delimitador (x, y)
80     idx = int(detections["labels"][i])
81     box = detections["boxes"][i].detach().cpu().numpy()
82     (startX, startY, endX, endY) = box.astype("int")
83
84     # Imprimimos la predicción en nuestro terminal
85     label ="{}:{:.2f}%".format(CLASSES[idx],confidence*100)
86     print("[DETECCION] {}".format(label))
87
88     # Agregamos el objeto detectado a nuestra lista de
       detecciones
89     Lista_Detectada.append(CLASSES[idx])
```

```
90
91      # Dibujamos el cuadro delimitador y su etiqueta
92      cv2.rectangle(orig, (startX, startY), (endX, endY),
93        COLORS[idx], 2)
94      y = startY - 15 if startY - 15 > 15 else startY + 15
95      cv2.putText(orig, label, (startX, y),
96        cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
```

Las **líneas 71-96** son un bucle que recorren todas las detecciones identificadas. Primero, la **línea 74** extrae la confianza asociada con el objeto detectado y la filtra (**línea 77**) comparándola con la confianza mínima (γ_{min}), si la confianza no pasa el filtro se prosigue con la siguiente detección. Si la confianza es superior al mínimo, en las **líneas 80-82** se extrae la etiqueta asociada al objeto detectado (carro, bus, camión, etc.) y las coordenadas de su cuadro delimitador ($startX_i, startY_i, endX_i, endY_i$). Luego, procedemos a imprimir la predicción en nuestro terminal (**líneas 85-86**) y a guardar el objeto en la lista de detecciones (**línea 89**). A continuación, en la imagen original dibujamos el cuadro delimitador con su respectiva etiqueta (**líneas 92-96**).

```
98 # Contamos los objetos detectados y los clasificamos
99 ConteoEtiquetas=Counter(Lista_Detectada)
100 Data = pd.DataFrame({"[TOTALES]":ConteoEtiquetas})
101 print(Data)
102
103 # Visualizamos y guardamos la imagen procesada
104 cv2.imshow("output", orig)
105 cv2.imwrite(args["output"], orig)
106
107 #Creamos un excel y guardamos la informacion
108 ubicacionexcel = args["output"].replace(".jpg",".xlsx")
109 escritor=pd.ExcelWriter(ubicacionexcel, engine="xlsxwriter")
110 Data.to_excel(escritor)
111 escritor.save()
112
113 print("¡PROCESO EXITOSO!")
114 cv2.waitKey(0)
```

Las **líneas 99-101** cuentan los elementos de la lista de detecciones y los clasifica de acuerdo a sus etiquetas (carro, bus, camión, etc.). Posteriormente, imprime los resultados (*Data*) en el terminal. Las **líneas 106-107** muestran la imagen procesada (*imagen de salida*) con las detecciones identificadas y sus respectivas confianzas y procede a guardar la imagen en la ruta de salida (*output*). Las **líneas 110-113** crean un archivo Excel (.xlsx) en la carpeta de salida, procede a escribir los datos de las detecciones identificadas (Data) y lo guarda. Finalmente, si el algoritmo se ejecutó hasta este punto, se imprimirá *«¡PROCESO EXITOSO!»*.

3.2.2 Red Neuronal Faster R-CNN (ResNet50)

Faster R-CNN (ResNet50) es una red neuronal convolucional de código abierto de la biblioteca PyTorch. En la línea 41 del script *«detect _image.py»* la definimos como: *frcc-resnet*, su función de llamada PyTorch está dada por: *detection.fasterrcnn_resnet50_fpn*.

Para ejecutar el algoritmo «detect _image.py» con la red neuronal pre-entrenada frcc-resnet, desde el Terminal, dirigirse a la ubicación de la carpeta Directorio y ejecutar el siguiente comando (sin saltos de línea):

```
python detect_image.py --model frcnn-resnet --labels coco-dataset
--image images/example_01.jpg --output output/example_frcnn_01.jpg
```

El archivo «coco.names», que contiene las etiquetas de clase, se encuentra ubicado en la carpeta «coco-dataset», por ello, el argumento de entrada [- -labels] llama a esta carpeta. La imagen de entrada utilizada es *«example_01.jpg»*, ubicada en la carpeta «images», el argumento [- -image] define la ruta relativa de esta imagen. Por último, en el argumento [- -output] define la ruta relativa y nombre de la imagen de salida y ubicación del archivo excel (.xlsx) con los datos totales contabilizados. Es importante tener creada la carpeta «output» en el Directorio, caso contrario el algoritmo ejecutará un error.

A continuación, se muestra la imagen de entrada *«example_01.jpg»* y la imagen que obtendremos (imagen de salida) luego de ejecutar el algoritmo de detección de objetos utilizando la red neuronal Faster R-CNN:

(a) Imagen de entrada (b) Imagen de salida

FIGURA N° 3.4: Imagen «example_01.jpg» procesada por la red neuronal Faster R-CNN

Asimismo, luego de ejecutado el algoritmo sobre la imagen «example_01.jpg», del Terminal obtendremos la siguiente salida:

```
[INFO] El proceso duró 3.455722 segundos
[DETECCION] car: 99.65%
[DETECCION] car: 99.63%
[DETECCION] bus: 99.63%
[DETECCION] car: 99.49%
[DETECCION] person: 99.26%
[DETECCION] car: 99.13%
[DETECCION] car: 98.76%
[DETECCION] car: 98.44%
[DETECCION] car: 95.54%
[DETECCION] car: 93.92%
[DETECCION] car: 85.10%
[DETECCION] car: 60.54%
        [TOTALES]
bus              1
car             10
person           1
¡PROCESO EXITOSO!
```

En un tiempo total de 3.455722 segundos, la red Fast R-CNN identificó correctamente los diez (10) vehículos de la foto que transitaban por la vía principal Av. Paseo de la República, Lima, Perú, incluido los que se encuentran parcialmente detrás de otros. Los vehículos que se encuentran en el primer plano han sido identificados con confianzas superiores al 98 % (seis vehículos con confianzas que oscilan entre 98.44 % y 99.65 %), los que se visualizan en un plano intermedio-fondo, varían entre el rango de 93 % a 95 % y los que se visualizan en el fondo, o solo parte de ellos, en el rango de 85 % a 60 %. Es preciso destacar que, la red ha identificado correctamente un (1) bus Metropolitano con una confianza de 99.63 %, incluido su conductor (una persona), el cual fue identificado con una confianza de 99.26 %.

Estos porcentajes de confianza representan la probabilidad determinada por el algoritmo (certeza) para afirmar que la tipología del objeto detectado (carro, bus, camión) efectivamente es la correcta. Por ejemplo, ”car: 99.65” significa que el algoritmo está 99.65 % seguro que el objeto detectado es un carro.

Finalmente, en la carpeta «output» se generarán y guardarán automáticamente los archivos: «example_frcnn_01.jpg» (foto con las detecciones identificadas) y el archivo Excel «example_frcnn_01.xlsx» (con la data de vehículos identificados).

```
1 #Directorio: Deteccion_Imagenes/output
2 |— example_frcnn_01.jpg
3 |— example_frcnn_01.xlsx
```

3.2.3 Red Neuronal RetinaNet (ResNet50)

RetinaNet es una red neuronal de la biblioteca PyTorch de código abierto. En el script *«detect _image.py»* (línea 42) la definimos como: *retinanet* y su función de llamada PyTorch está dada por: *detection.retinaet_resnet50_fpn*.
Para ejecutar el algoritmo «detect _image.py» con la red neuronal RetinaNet, desde el Terminal, dirigirse a la ubicación de la carpeta Directorio y ejecutar el siguiente comando (sin saltos de línea):

```
python detect_image.py --model retinanet --labels coco-dataset
--image images/example_01.jpg --output output/example_retina_01.jpg
```

A continuación, se muestra una comparación entre la imagen de entrada *«example_01.jpg»* y la imagen de salida luego de ejecutar el algoritmo de detección de objetos utilizando la red neuronal RetinaNet:

(a) Imagen de entrada (b) Imagen de salida

FIGURA N° 3.5: Imagen «example_01.jpg» procesada por la red neuronal RetinaNet

Como salida del Terminal obtendremos lo siguiente:

```
[INFO] El proceso duró 3.477699 segundos
[DETECCION] car: 94.57%
[DETECCION] car: 90.98%
[DETECCION] car: 88.51%
[DETECCION] bus: 86.93%
[DETECCION] person: 85.16%
[DETECCION] car: 83.63%
[DETECCION] car: 77.82%
[DETECCION] car: 72.35%
[DETECCION] car: 68.47%
[DETECCION] car: 64.85%
        [TOTALES]
bus             1
car             8
person          1
¡PROCESO EXITOSO!
```

En un tiempo total de 3.477699 segundos, la red RetinaNet identificó correctamente ocho (8) de los diez (10) vehículos que transitan por la vía. Los vehículos del primer plano han sido identificados con confianzas que van del 77.82 % a 94.57 % (cinco vehículos). En el plano del fondo solo se identificaron tres (3) de los cinco (5) vehículos que pueden ser visualizados a simple vista, estos tienen confianzas que van entre los 64.85 % a 72.35 %. El bus Metropolitano fue identificado con una confianza de 86.93 %, asimismo, su conductor (persona) fue identificado con una confianza de 85.16 %. Estos porcentajes de confianza representan la probabilidad determinada por el algoritmo (certeza) para afirmar que la tipología del objeto detectado (carro, bus, camión) efectivamente es la correcta.

En la carpeta «output» se generarán y guardarán automáticamente los archivos: «example_retina_01.jpg» (imagen de salida con las detecciones identificadas) y el archivo Excel «example_retina_01.xlsx» (con la data de vehículos identificados).

```
#Directorio: Deteccion_Imagenes/output
|— example_retina_01.jpg
|— example_retina_01.xlsx
```

3.2.4 Red Neuronal YoloV3

You only look once (YOLO) es un sistema de detección de objetos desarrollado por Joseph Redmon y Ali Farhadi. Fue publicado por primera vez en el 2015 y mejorado en el 2016 con la versión *YOLO9000* (YOLOv2). En el 2018, Redmon y Farhadi (2018) publicaron la versión *YOLOv3: An incremental Improvenment*, el cual contenía mejoras en el rendimiento de la red.

YOLO aplica una única red neuronal a la imagen completa y divide la imagen en regiones para predecir cuadros delimitadores y probabilidades para cada región. Para la presente tesis utilizaremos YOLOv3, particularmente, la entrenada con el conjunto de datos COCO dataset.

Para descargar el código fuente y los archivos YOLO ingrese al siguiente enlace: https://t.ly/HS24 o escanee el código QR de la Figura N° 3.6 Posteriormente, proceda a ubicarlos en una carpeta directorio en su ordenador.

FIGURA N° 3.6: Código QR descarga archivos para detección en imágenes con YOLO

Luego de descargar y ubicar los archivos en su ordenador, la estructura de la carpeta quedará conformada de la siguiente manera:

```
#Directorio: Deteccion_Imagenes_YOLOv3
|— images
|— |—  example_01.jpg
|— |—  ...
|— |—  example_18.jpg
|— output
|— yolo—coco
|— |—  coco.names
|— |—  yolov3.cfg
|— |—  yolov3.weights
|— yolo.py
```

Las carpetas *«images»* y *«output»* son las mismas utilizadas en los algoritmos de las redes de PyTorch (Faster R-CNN y RetinaNet). *«Images»* contiene una lista de dieciocho (18) fotografías de flujo vehicular, *«output»* es una carpeta vacía, en ella se guardarán las imágenes procesadas y los archivos excel (.xlsx) generados luego de ejecutar el algoritmo. La carpeta *«yolo-coco»* contiene los archivos del modelo de la red YOLOv3, entre ellos, la lista de etiquetas de la base de datos COCO dataset (*«coco.names»*), la configuración de la red neuronal (*«yolov3.cfg*) y los pesos pre-entrenados por Redmon y Farhadi (*«yolov3.wights»*). Finalmente, encontrará el script *«yolo.py»* conteniendo el algoritmo de detección de objetos.

Para ejecutar el detector de objetos a la imagen *«example_01.jpg»* con la red YOLO v3 («yolo.py») , desde el Terminal, dirigirse a la ubicación de la carpeta Directorio y ejecutar el siguiente comando (sin saltos de línea):

```
python yolo.py --image images/example_01.jpg --yolo yolo-coco
--output output/example_yolo_01.jpg
```

(a) Imagen de entrada (b) Imagen de salida

FIGURA N° 3.7: Imagen «example_01.jpg» procesada por la red neuronal YOLOv3

Como salida del Terminal obtendremos lo siguiente:

```
[INFO] El proceso YOLO duró 1.069401 segundos
[DETECCION] bus: 99.48%
[DETECCION] car: 99.29%
[DETECCION] car: 99.27%
[DETECCION] car: 99.02%
[DETECCION] car: 94.41%
[DETECCION] car: 93.97%
[DETECCION] car: 93.73%
[DETECCION] person: 91.96%
[DETECCION] car: 75.91%
[DETECCION] car: 70.60%
        [TOTALES]
bus             1
car             9
person          1
¡PROCESO EXITOSO!
```

En un tiempo total de 1.069401 segundos (un tercio del tiempo que demoran las redes Faster R-CNN y RetinaNet), la red YOLOv3 identificó correctamente nueve (9) de diez (10) vehículos en la imagen. Cuatro (4) vehículos fueron identificados con confianzas superiores al 99 %, tres (3) por encima del 93.73 % y dos (2) en el rango de 70 % y 80 %. Un (1) vehículo que se encontraba en el plano de fondo no fue identificado, esto puede ser por diversos factores, entre ellos, no se visualizaba completamente y se encontraba detrás de otro vehículo. El bus Metropolitano fue identificado con una confianza del 99.48 %, su conductor (persona), con una confianza del 91.96 %. Estos porcentajes de confianza representan la probabilidad determinada por el algoritmo (certeza) para afirmar que la tipología del objeto detectado (carro, bus, camión) efectivamente es la correcta.

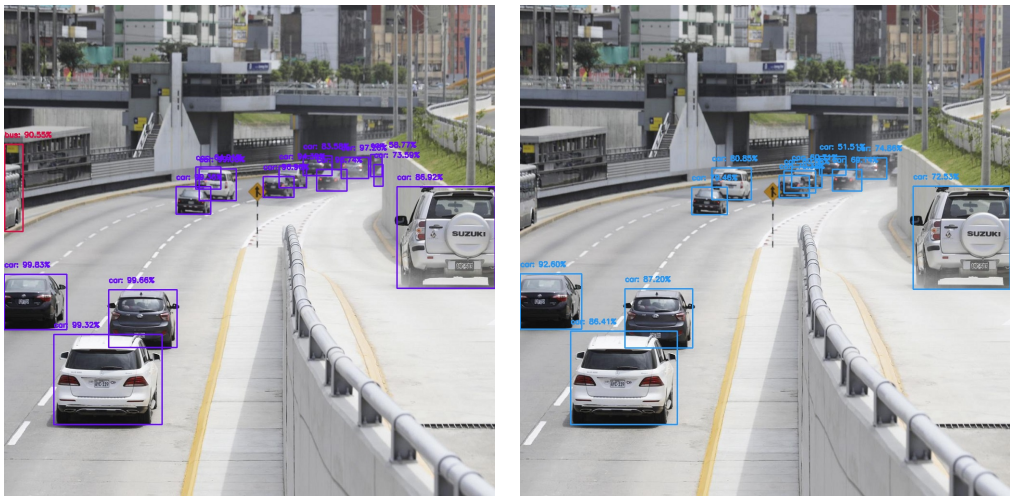
YOLOv3 mejora significativamente el tiempo de proceso sin perder precisión, es tres veces más rápido que las redes pre-entrenadas de PyTorch. Estas variables son importantes a considerar si se desea analizar y procesar grandes cantidades de datos.

3.2.5 Resultados de aplicación

A continuación, se presentan los resultados de aplicar las tres (3) redes neuronales a las imágenes de la carpeta *«images»*:

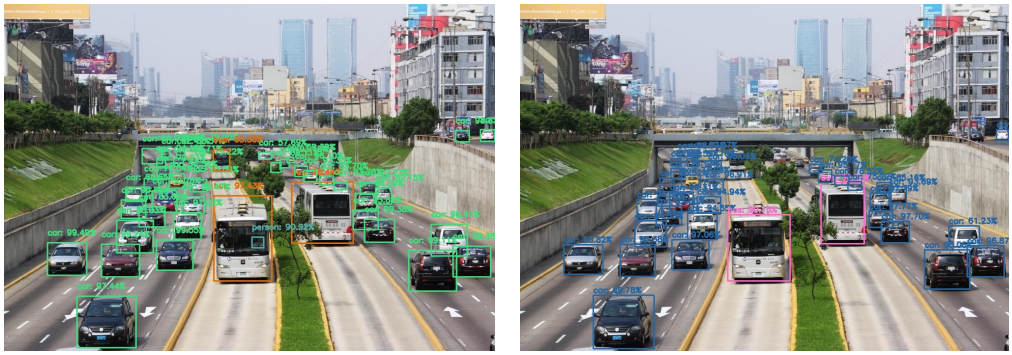
TABLA N° 3.1: Comparación de resultados de detección de vehículos en imágenes - Parte I

Imagen	Red Neuronal	Tiempo (s)	Veh. Detectad	Veh. No Detec	Falso Positivo	Total Existent.
Example_01	Faster R	3.4557	10 carros 1 bus			10 carros 1 bus
	RetinNe	3.4776	8 carros 1 bus	2 carros		
	YoloV3	1.0694	9 carros 1 bus	1 carro		
Example_02	Faster R	2.5760	14 carros 1 bus		1 carro	13 carros 1 bus
	RetinNe	2.4057	12 carros	2 carros 1 bus	1 carro	
	YoloV3	0.9992	10 carros 1 bus 1 camión	3 carros	1 camión	
Example_03	Faster R	3.4087	43 carros 3 buses	3 carros	1 carro	45 buses 3 buses
	RetinNe	3.3738	22 carros 2 buses	23 carros 1 bus		
	YoloV3	1.0282	32 carros 2 bus	13 carros 1 bus		
Example_04	Faster R	3.3750	23 carros 2 camión	1 carro	2 carros	22 carros 2 camión
	RetinNe	3.3800	20 carros 2 camión	3 carros	1 carro	
	YoloV3	1.3647	19 carros 2 camión	4 carros 2 camión	1 carro 2 camión	
Example_05	Faster R	3.7338	10 carros 1 camión		1 camión	10 carros
	RetinNe	3.7798	10 carros			
	YoloV3	1.1228	10 carros			



(a) Red Faster R-CNN (b) Red RetinaNet

FIGURA N° 3.8: Imagen «example_02.jpg» procesada por la red neuronal Faster R-CNN y RetinaNet



(a) Red Faster R-CNN (b) Red YoloV3

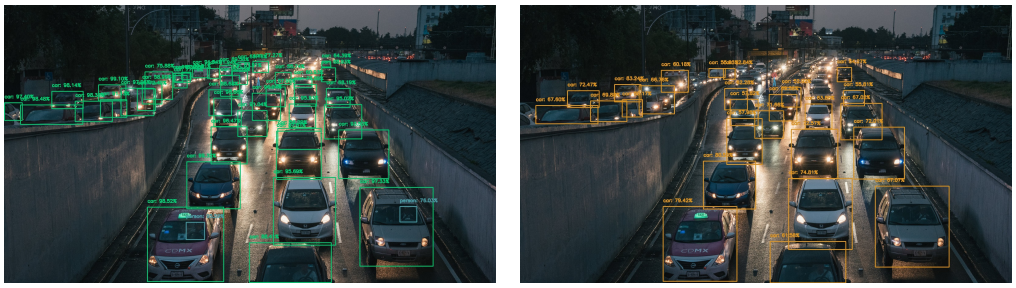
FIGURA N° 3.9: Imagen «example_03.jpg» procesada por la red neuronal Faster R-CNN y YoloV3

(a) Red RetinaNet (b) Red YoloV3

FIGURA N° 3.10: Imagen «example_04.jpg» procesada por la red neuronal RetinaNet y YoloV3

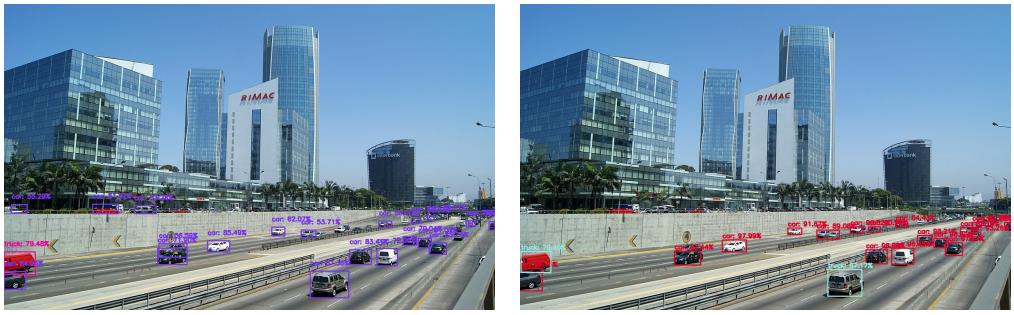
TABLA N° 3.2: Comparación de resultados de detección de vehículos en imágenes - Parte II

Imagen	Red Neuronal	Tiempo (s)	Veh. Detectad	Veh. No Detec	Falso Positivo	Total Existent.
Example_06	Faster R	4.9129	43 carros	7 carros		50 carros
	RetinNe	3.6850	26 carros	24 carros		
	YoloV3	1.0694	23 carros	27 carros		
Example_07	Faster R	4.0271	78 carros 4 buses 2 camión	3 carros 1 bus	1 camión	81 carros 5 buses 1 camión
	RetinNe	4.0410	25 carros 3 buses 1 camión	56 carros 2 buses		
	YoloV3	1.3820	38 carros 4 bus	43 carros 1 bus 1 camión		
Example_08	Faster R	3.9344	42 carros 1 camión	1 camión	10 carros	32 carros 2 camión
	RetinNe	4.0310	21 carros 1 camión	12 carros 1 camión	1 carro	
	YoloV3	1.0272	18 carros 2 camión	14 carros 1 camión	1 camión	
Example_09	Faster R	3.9004	39 carros 3 buses 1 camión	2 camión	4 carros 1 camión	35 carros 3 buses 2 camión
	RetinNe	3.1274	24 carros 2 buses	11 carros 1 bus		
	YoloV3	1.0444	24 carros 2 buses	11 carros 1 bus		
Example_10	Faster R	2.8663	22 carros 1 motoc. 2 camión	2 camión	5 carros	17 carros 1 motoc. 4 camión
	RetinNe	2.8401	11 carros 1 motoc. 3 camión	6 carros 1 camión		
	YoloV3	0.9985	16 carros 1 motoc. 3 camión	2 carro 2 camión	1 carro 1 camión	



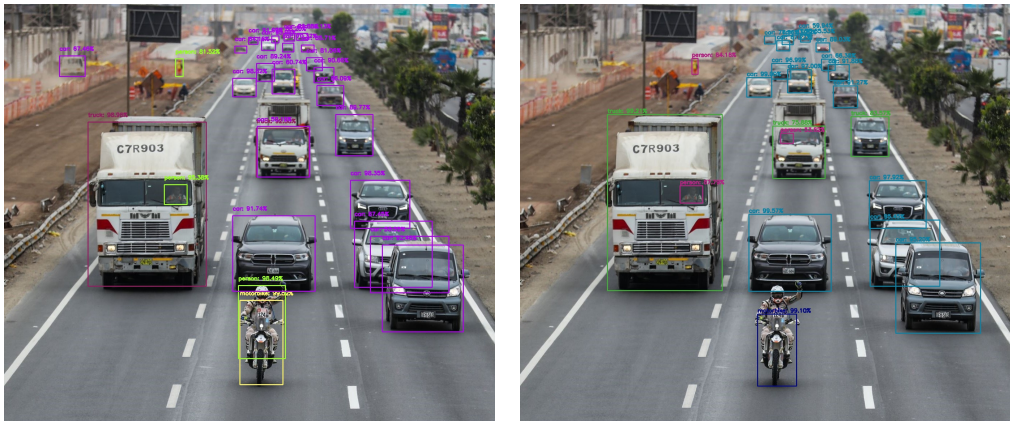
(a) Red Faster R-CNN (b) Red YoloV3

FIGURA N° 3.11: Imagen «example_06.jpg» procesada por la red neuronal Faster R-CNN y YoloV3



(a) Red RetinaNet (b) Red YoloV3

FIGURA N° 3.12: Imagen «example_08.jpg» procesada por la red neuronal RetinaNet y YoloV3

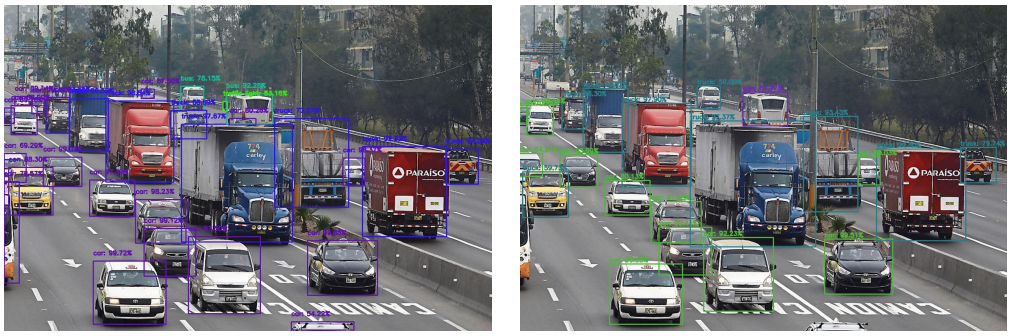


(a) Red Faster R-CNN (b) Red YoloV3

FIGURA N° 3.13: Imagen «example_10.jpg» procesada por la red neuronal Faster R-CNN y YoloV3

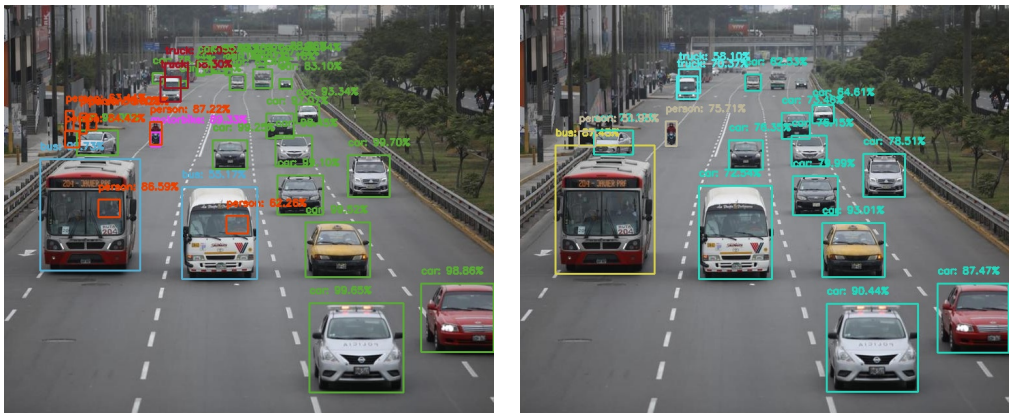
TABLA N° 3.3: Comparación de resultados de detección de vehículos en imágenes - Parte III

Imagen	Red Neuronal	Tiempo (s)	Veh. Detectad	Veh. No Detec	Falso Positivo	Total Existent.
Example_11	Faster R	2.8133	22 carros 1 bus 6 camión	6 carros 5 buses 2 camión	3 camión	28 carros 6 buses 5 camión
	RetinaNe	2.7129	13 carros 1 bus 4 camión	15 carros 5 buses 2 camión	1 camión	
	YoloV3	1.0009	20 carros 2 buses 3 camión	10 carros 3 buses 3 camión	2 carros 1 camión	
Example_12	Faster R	3.5311	18 carros 2 buses 8 camión	2 carros 1 bus 1 camión	2 carros	18 carros 3 buses 9 camión
	RetinaNe	3.4948	9 carros 2 buses 7 camión	9 carros 1 bus 3 camiones	1 camión	
	YoloV3	1.2554	11 carros 1 bus 9 camión	7 carros 2 buses 2 camiones	2 camión	
Example_13	Faster R	3.1668	22 carros 2 buses 2 camión 1 motoc.	2 buses 1 camión	7 carros 1 camión	15 carros 4 buses 2 camión 1 motoc.
	RetinaNe	2.8227	12 carros 1 bus 2 camión	4 carros 3 buses 1 camión	1 carro 1 motoc. 1 camión	
	YoloV3	1.0767	19 carros 1 bus 1 motoc.	3 buses 2 camión	4 carros	
Example_14	Faster R	3.5898	38 carros 2 motoc.		2 carros	36 carros 2 motoc.
	RetinaNe	4.4344	31 carros 2 motoc. 2 camión	6 carros	1 carro 2 camión	
	YoloV3	1.5662	36 carros 1 motoc.	1 motoc.		



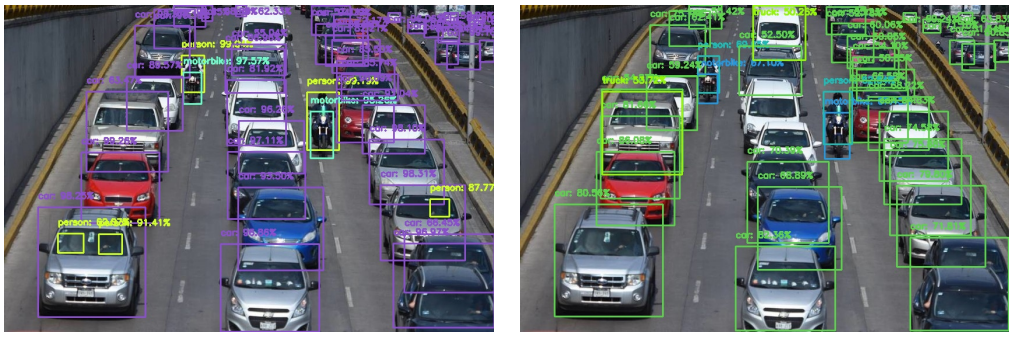
(a) Red Faster R-CNN (b) Red YoloV3

FIGURA N° 3.14: Imagen «example_12.jpg» procesada por la red neuronal Faster R-CNN y YoloV3



(a) Red Faster R-CNN (b) Red RetinaNet

FIGURA N° 3.15: Imagen «example_13.jpg» procesada por la red neuronal Faster R-CNN y RetinaNet

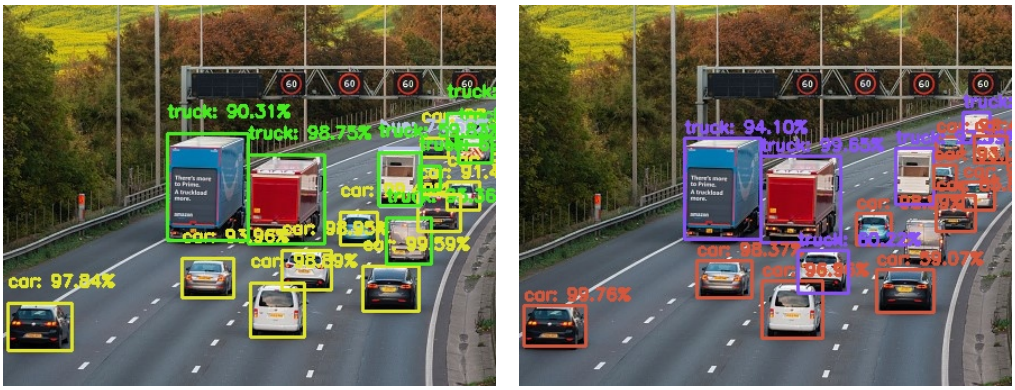


(a) Red Faster R-CNN (b) Red RetinaNet

FIGURA N° 3.16: Imagen «example_14.jpg» procesada por la red neuronal Faster R-CNN y RetinaNet

TABLA N° 3.4: Comparación de resultados de detección de vehículos en imágenes - Parte IV

Imagen	Red Neuronal	Tiempo (s)	Veh. Detectad	Veh. No Detec	Falso Positivo	Total Existent.
Example_15	Faster R	3.8589	9 carros 1 bus 7 camión	1 carro	1 bus 1 camión	10 carros 6 camión
	RetinaNe	4.3616	9 carros 2 buses 6 camión	1 carro 1 camión	2 buses 1 camión	
	YoloV3	0.9833	10 carros 5 camión	1 carro 2 camión	1 carro 1 camión	
Example_16	Faster R	3.7883	77 carros 2 buses 1 motoc. 5 camión	16 carros 2 motoc.	1 carro	92 carros 5 camión 2 buses 3 motoc.
	RetinaNe	3.3354	47 carros 2 buses 1 motoc. 5 camión	45 carros 1 camión 2 motoc.	1 camión	
	YoloV3	1.3731	51 carros 3 buses	42 carros 5 camión 3 motoc.	1 carro 1 bus	
Example_17	Faster R	3.7097	75 carros 9 camión	10 carros	1 carro 2 camión	84 carros 7 camión
	RetinaNe	3.6703	46 carros 1 bus 2 camión	38 carros 5 camión	1 bus	
	YoloV3	1.3731	31 carros 1 bus 6 camión	44 carros 1 camión	1 bus 1 carro	
Example_18	Faster R	3.5337	62 carros 1 motoc.	9 carros 3 motoc.	2 carros	69 carros 4 motoc.
	RetinaNe	3.4727	41 carros	28 carros 4 motoc.		
	YoloV3	1.0295	55 carros	14 carros 4 motoc.		

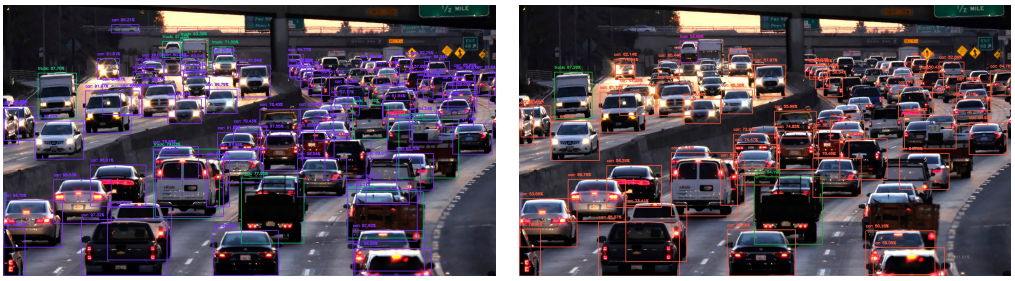


(a) Red Faster R-CNN (b) Red YoloV3

FIGURA N° 3.17: Imagen «example_15.jpg» procesada por la red neuronal Faster R-CNN y YoloV3

(a) Red Faster R-CNN (b) Red YoloV3

FIGURA N° 3.18: Imagen «example_16.jpg» procesada por la red neuronal Faster R-CNN y YoloV3



(a) Red Faster R-CNN (b) Red RetinaNet

FIGURA N° 3.19: Imagen «example_17.jpg» procesada por la red Faster R-CNN y RetinaNet

(a) Red Faster R-CNN (b) Red YoloV3

FIGURA N° 3.20: Imagen «example_18.jpg» procesada por la red neuronal Faster R-CNN y YoloV3

3.3 DETECCIÓN Y CLASIFICACIÓN VEHICULAR EN VIDEOS

En esta sección se desarrollará la metodología para la detección y clasificación vehicular en videos. Para ello, se utilizará el algoritmo de conteo de objetos de código abierto *«Ivy»*, el cual fue desarrollado por Nicholas Kajoh, ingeniero de software de Nigeria, y cuyo código original puede ser encontrado en el siguiente repositorio GitHub: https://github.com/nicholaskajoh/ivy.

3.3.1 Algoritmo de detección y clasificación vehicular en videos

En la Figura 3.22 se desarrolla el diagrama de flujo del algoritmo que se utilizará para la detección y clasificación vehicular mediante la utilización de videograbaciones. Siguiendo las pautas del diagrama de flujo, en primer lugar, se debe clonar o descargar el repositorio original GitHub de Nicholas Kajoh o en su defecto, descargar el algoritmo con el que trabajará la presente tesis, el cual mantiene la esencia del código original con ciertas modificaciones relacionadas al área de visualización y tratamiento de la información procesada. Para acceder al repositorio de la tesis, descargue los archivos del siguiente enlace: https://t.ly/Un1za o ingrese al código QR de la Figura N° 3.21. Posteriormente, proceda a ubicarlos en una carpeta directorio en su ordenador y cree un entorno virtual.

FIGURA N° 3.21: Código QR descarga archivos para detección en videos

Para ejecutar el algoritmo de detección se utilizó Python versión 3.8.2 y las librerías de OpenCV y PyTorch, el proceso de instalación de estos paquetes se explica en la sección 3.1 de la presente tesis. Para facilitar el trabajo de programación se utilizó el editor de código fuente «Visual Studio Code», desarrollado por Microsoft en el 2015 y de código abierto, el cual permite trabajar con diferentes lenguajes de programación, entre ellos Python, y aporta funcionalidades prácticas al momento de programar .

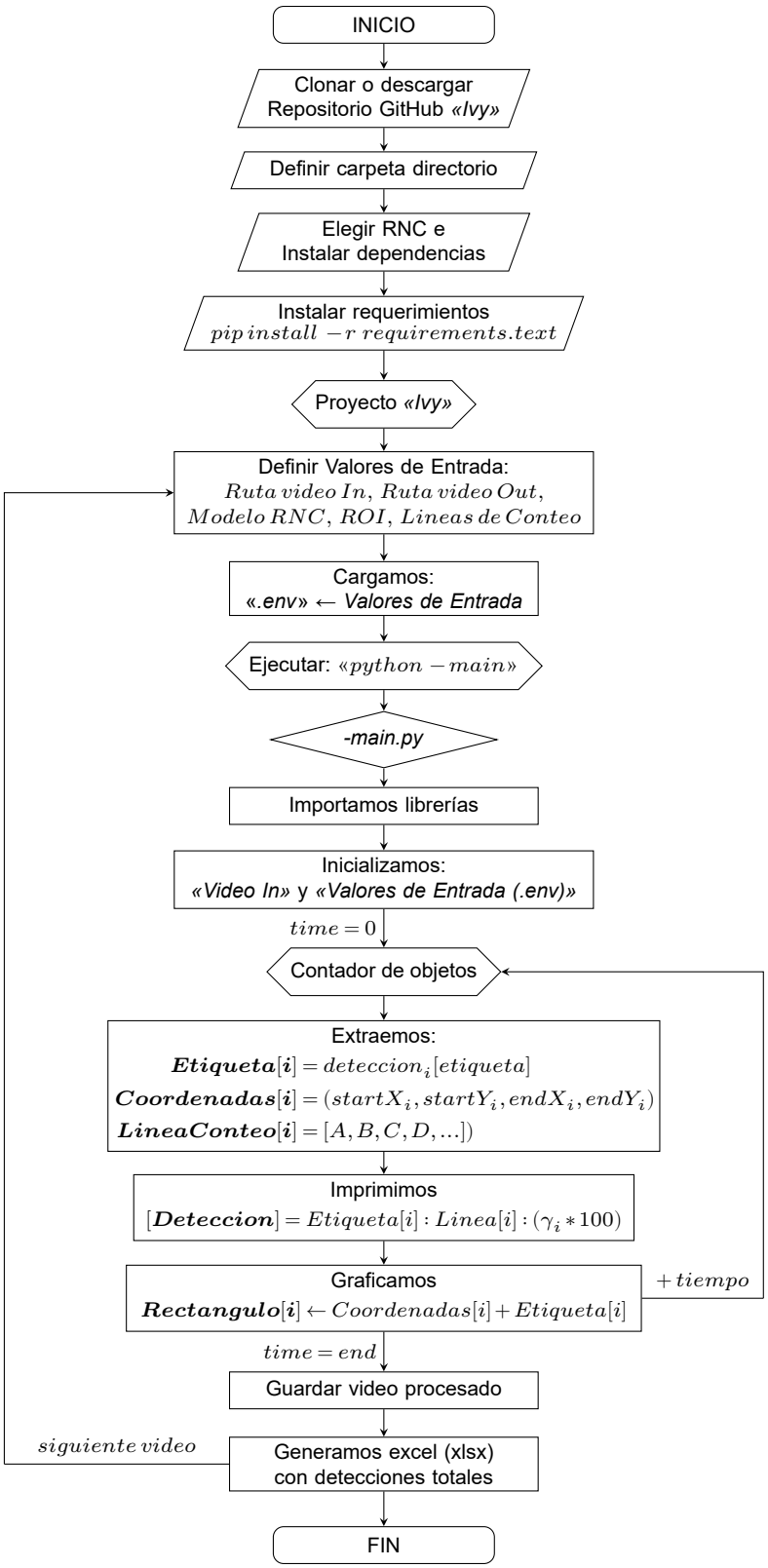


FIGURA N° 3.22: Algoritmo de detección y clasificación de objetos en videos

Luego de descargar los scripts del algoritmo «Ivy», ya sea desde el GitHub de Nicholas Kajoh o del repositorio Drive de la presente tesis, la carpeta directorio quedará compuesta por los siguientes archivos:

```
# Directorio Proyecto ivy-master
|- []. github/workflows
|- [] data
|- [] detectors
|- [] tests
|- [] util
|- .converagerc
|- .env
|- .gitignore
|- .pylintrc
|- counter.py
|- LICENSE
|- main.py
|- ObjectCounter.py
|- pytest.ini
|- README.md
|- requirements.txt
|- settings.py
|- tracter.py
```

3.3.2 Redes Neuronales para detección vehicular

Dentro de la carpeta *«data»* se ubicará la carpeta *«detectors»*, la cual contendrá las dependencias (arquitectura de la red, pesos, configuración, etc.) de las redes neuronales que se utilizarán para detectar los vehículos. Para fines de la presente tesis utilizaremos dos redes pre-entrenadas: YoloV3 y Detectron2.

3.3.2.1 Detector YoloV3

You only look once v3 (YoloV3) es una red neuronal entrenada con el conjunto de datos COCO dataset, fue desarrollado y publicado por Joseph Redmon y Ali Farhadi en el 2018,en el capítulo *3.2.4 Red Neuronal YoloV3* de la presente tesis se brinda un mayor detalle acerca de esta red. Para utilizar YoloV3 es necesario descargar los pesos y configuración de la red, para ello, una de las maneras es dirigiéndose al sitio web oficial: https://pjreddie.com/darknet/yolo/ y descargarlos desde allí, o si descargó el algoritmo «Ivy» desde al repositorio de la tesis (Figura N° 3.21) los encontrará ya descargados en la carpeta *«./data/detectors/»*.

Las dependencias necesarias para ejecutar el detector y clasificador de vehículos con la red YoloV3 son: *«yolov3.cfg»* y *«yolov3.wights»*. Es así que, la carpeta *«data»* quedará compuesta de la siguiente manera:

```
# Carpeta «data» – Directorio Proyecto ivy–master
|– [] data
|– |– [] detectors
|– |– |– [] ...
|– |– |– [] yolo
|– |– |– |– yolov3.cfg
|– |– |– |– yolov3.weights
```

3.3.2.2 Detector Detectron2

Detectron2 es la biblioteca de inteligencia artificial de Facebook AI Research (FAIR) desarrollada en el 2019, proporciona algoritmos y modelos de detección y segmentación de objetos de última generación y que son utilizados en varios proyectos de investigación de visión artificial y aplicaciones en Facebook, entre sus modelos entrenados disponibles están: Faster R-CNN, Mask R-CNN, RetinaNet, Cascade R-CN, Tensor Mask, entre otros. (Wu, Kirillov, Massa, Lo, y Girshick, 2019). Para mayor detalle puede visitar el sitio oficial del repositorio GitHub *«detectron2»* en el siguiente enlace: https://github.com/facebookresearch/detectron2.

Para utilizar los modelos de redes neuronales preentrenadas de Detecton2 es necesario instalar sus dependencias, para ello, desde su Terminal ejecute el siguiente comando (sin saltos de línea):

```
python -m pip install 'git+https://github.com/facebookresearch
    /detectron2.git'
```

O puede descargar o clonar en su disco local el repositorio GitHub Detectron2 y ejecutar el siguiente comando:

```
git clone https://github.com/facebookresearch/detectron2.git
python -m pip install -e detectron2
```

Para mayor información de la instalación de Detectron2 consulte las instrucciones de instalación en su sitio web oficial: https://detectron2.readthedocs.io/en/latest/tutorials/install.html.

Luego de instalado Detectron2 se debe elegir la red neuronal que se utilizará para detectar y clasificar los vehículos en nuestro algoritmo «Ivy». Para ello, Detectron2 ofrece un amplio repositorio de redes pre-entrenadas con las bases de datos COCO dataset e ImageNet. Para fines de la presente tesis, se eligió una red neuronal Faster R-CNN con una red troncal ResNet + FPN, ya que estas redes suelen tener mejores rendimientos en relación velocidad-precisión. El nombre de la red a utilizar

es *«R50-FPN»* (id modelo: 137257794), la arquitectura y sus pesos de esta red, y el de todas las redes entrenados por Detectron2, pueden ser encontrados y descargados desde su repositorio de modelos, a través del siguiente enlace: https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md.

Las dependencias necesarias para ejecutar el detector y clasificador de vehículos con la red *R50-FPN* de Detectron2 son: *« Base-RCNN-FPN.yaml»* y *«R50-FPN_model_final_b275ba.pkl»*; en caso, descargó el repositorio Drive de la tesis (Figura N° 3.21) encontrará estos archivos ya descargados en la carpeta *«./data/-detectors/»*. De este modo, la carpeta *«data»* quedará compuesta de la siguiente manera:

```
# Carpeta «data» – Directorio Proyecto ivy–master
|– [] data
|– |– [] detectors
|– |– |– [] ...
|– |– |– [] detectron2
|– |– |– |– Base–RCNN–FPN.yaml
|– |– |– |– R50–FPN_model_final_b275ba.pkl
```

3.3.3 Instalación de requerimientos

Continuando con el proceso de preparación del entorno virtual, para que el algoritmo «Ivy» se ejecute de manera correcta, es necesario instalar las siguientes bibliotecas:

```
|– numpy==1.16.4
|– opencv–contrib–python==4.1.0.25
|– python–dotenv==0.10.3
|– pathlib2==2.3.4
|– pytest==5.1.3
|– pytest–cov==2.7.1
|– pylint==2.4.4
|– python–json–logger==0.1.11
|– joblib==0.14.1
```

Para ello, diríjase a su entorno virtual y desde su Terminal ejecute el siguiente comando, el cual activará el script *«requirements.txt»* conteniendo las bibliotecas con sus respectivas versiones a utilizar y las procederá a instalar:

```
pip install -r requirements.txt
```

3.3.4 Algoritmo Ivy

El algoritmo «*Ivy*» será el responsable de cargar la información de entrada, las redes neuronales, procesarla la información y emitir los resultados de conteo y clasificación de vehículos en videos. Para que Ivy se ejecute correctamente es necesario contar con los archivos del repositorio GitHub de Nicholas Kajoh o los del repositorio Drive de la presente tesis (Figura N° 3.21), en cualquier de los casos, si realizó correctamente los pasos previos de las subsecciones 3.3.1, 3.3.2 y 3.3.3, Ivy se ejecutará de manera exitosa.

3.3.4.1 Definir valores de entrada

En esta sección se definirán los parámetros y valores de entrada que utilizará Ivy para detectar y clasificar vehículos. Entre estos parámetros se encuentran los siguientes:

- **VideoIn**: Ruta absoluta/relativa del video o cámara de entrada con la escena de flujo vehicular. Acepta formato mp4, avi, entre otros.
- **Región de Interés de Detección**: Del inglés *Detection Region of Interest (DROI)*. Es el área en el video que representa un flujo vehícular en la que el algoritmo de detección se centrará en identificar y detectar los vehículos mediante la red neuronal seleccionada.
- **Líneas de conteo**: Son líneas dibujadas en la escena del video que nos permitirá contabilizar los vehículos detectados cada vez que un vehículo pase sobre estas. Están conectadas a un contabilizador que se actualiza instantáneamente con el registro de un nuevo vehículo identificado.
- **Detector**: Modelo de red neuronal que se utilizará apara detectar los vehículos y clasificarlos según su tipología; entre las opciones a elegir tenemos: *Yolov3* y *Detectron2*; en la sección 3.3.2 se precisan a mayor detalle.
- **GPU_Aceleration**: Si cuenta con una tarjeta gráfica externa, puede activar su aceleración GPU para obtener un mejor rendimiento.
- **Tracker**: Algoritmo utilizado para el seguimiento de vehículos (opc.: kcf, csrt).
- **VideoOut**: Ruta absoluta/relativa del video de salida con la escena de flujo vehicular procesada, se visualizan los vehículos identificados y su conteo. Acepta formato mp4, avi, entre otros.
- **Ventana de Depuración**: Mientras se ejecuta el algoritmo Ivy puede optar por visualizar el proceso de identificación, detección y conteo de vehículos a través de una ventana en modo depuración (ventana externa).

3.3.4.2 Cargar valores de entrada

Para introducir los valores de entrada al algoritmo, primero, en su directorio raíz «Ivy» cree un script llamado «*.env*», esto a partir del script «*.env.example*»; es decir, duplique o modifique el script «*.env.example*» y cámbiele de nombre a «*.env*». Si obtuvo el algoritmo Ivy a partir del reposito Drive de la tesis, ya tendrá creado el script *«.env»* en su directorio raíz.

$$[.env.example] \overset{copy}{\rightarrow} [.env.example.(copy)] \overset{modif.}{\rightarrow} [.env] \tag{3.2}$$

A continuación, se detalla el contenido del script «*.env*», el cual contiene información de los valores de entrada y configuración que utilizará *«Ivy»* para ejecutarse, entre ellos: Ruta absoluta o relativa del video de entrada, video de salida, DROI, líneas de conteo, detector (red neuronal), etc.

```
1  VIDEO = "D:/Universidad Nacional de Ingenieria/TESIS UNI/CONTADOR
       VEHICULAR_VFINAL/ivy-masterv3/ivy-master/data/videos/Habich_Trim
       .mp4"
2  DROI  = [(319, 162), (537, 83), (1909, 553), (1791, 843)]
3  USE_DROI     = True
4  SHOW_DROI    = True
5  SHOW_COUNTS = True
6  MCDF  = 2
7  MCTF  = 3
8  DI    = 10
9  ENABLE_GPU_ACCELERATION = True
10 DETECTOR  = "yolo"
11 TRACKER   = "kcf"
12 RECORD    = True
13 OUTPUT_VIDEO_PATH = "./data/videos/output.avi"
14 HEADLESS  = False
15 COUNTING_LINES  = [{'label': 'A', 'line': [(540, 232), (687, 137)]},
       {'label': 'B', 'line': [(951, 403), (1143, 264)]}]
```

La **línea 1** define la ruta absoluta del video de entrada (.mp4, .avi, u otro formato) que será procesado y del cual se obtendrán los datos de detección y clasificación vehicular. De acuerdo a la estructura del directorio, los videos son cargados en la carpeta *«./data/videos/»*. La **línea 2** define las coordenadas (píxeles) de la región de detección (DROI), que es el plano geométrico del video en donde se aplicará únicamente la detección de vehículos, es decir, solo los objetos (carros, buses, camiones, etc.) que ingresen a esta región pasarán por el proceso de detección y clasificación, esto se hace con el fin de sectorizar la dirección de los flujos vehiculares y determinar los vehículos que transitan en una dirección especifica. Las **líneas 3-9** sirven para configurar la visualización del DROI, el conteo de vehículos, valores de intervalo para la detección de objetos y uso del GPU.

La **línea 10** define la red neuronal que se utilizará para detectar los vehículos, acepta como valor de entrada "$yolo$" y "$detectron2$". La **línea 11** define el tracker para el seguimiento de los vehículos detectados, acepta "kcf" y "$csrt$". La **línea 12** define si se desea obtener un video resultante luego de ejecutado el algoritmo (video de salida), acepta "$True$" y "$False$". La **línea 13** define la ruta relativa y el nombre del video de salida, acepta formatos .mp4, .avi y otros.

La **línea 15** define las coordenadas de las líneas de conteo, que son líneas dibujadas en el video y que sirven de contadores. Podemos crear "n" líneas y se recomienda dibujarlas en las entradas y salidas de flujos vehiculares.

```
16 YOLO_WEIGHTS_PATH="./data/detectors/yolo/yolov3.weights"
17 YOLO_CONFIG_PATH="./data/detectors/yolo/yolov3.cfg"
18 YOLO_CLASSES_PATH="./data/detectors/coco_classes.txt"
19 YOLO_CLASSES_OF_INTEREST_PATH="./data/detectors/
       coco_classes_of_interest.txt"
20 YOLO_CONFIDENCE_THRESHOLD=0.5
21 DETECTRON2_CONFIDENCE_THRESHOLD=0.5
22 DETECTRON2_CONFIG_PATH="./data/detectors/detectron2/Base-RCNN-FPN.
       yaml"
23 DETECTRON2_WEIGHTS_PATH="./data/detectors/detectron2/R50-
       FPN_model_final_b275ba.pkl"
24 DETECTRON2_NUM_CLASSES=80
25 DETECTRON2_CLASSES_PATH="./data/detectors/coco_classes.txt"
26 DETECTRON2_CLASSES_OF_INTEREST_PATH="./data/detectors/
       coco_classes_of_interest.txt"
27 ENABLE_CONSOLE_LOGGER=True
28 ENABLE_FILE_LOGGER=True
29 ENABLE_LOGSTASH_LOGGER=False
30 LOG_FILES_DIRECTORY="./data/logs/"
31 LOG_IMAGES=False
32 DEBUG_WINDOW_SIZE=(1458, 680)
33 HUD_COLOR=(255, 0, 0)
```

Las **líneas 16-20** definen las rutas relativas de los archivos que contienen los pesos pre-entrenados, configuración, clases de detección y confianza de la red neuronal *«YoloV3»*, la instalación de las dependencias de esta red fueron explicadas en la sección 3.3.2.1 de la presente tesis. Del mismo modo, las **líneas 21-26** definen las rutas relativas de los archivos correspondientes a la red neuronal *«Detectron2»*, la información e instalación de esta red están explicadas en la sección 3.3.2.2.

Asimismo, las **líneas 27-31** contienen configuración de los archivos de salida «.logs» incluyendo información de los conteos realizados por el algoritmo (tipo de objeto contado, ubicación del objeto, confianza de detección, línea que lo contó, entre otros), los cuales se guardarán en la carpeta *«./data/logs/»*.

Finalmente, las líneas **líneas 32-33** configuran el tamaño y corrige el color de la ventana de depuración emergente, en esta ventana externa se puede visualizar el proceso de identificación, seguimiento (tracker), y conteo de los vehículos mientras se ejecuta el algoritmo Ivy.

NOTA: Todos los valores y configuraciones del script «$.env$» pueden ser modificados y su cambio depende de la escena del video y los flujos vehiculares que se requiere analizar.

3.3.4.3 Ejecutar «Ivy»

Para ejecutar correctamente *«Ivy»*, primeramente, debe ubicar el video que se analizará en la carpeta *«./data/videos/»* y luego definir los valores de entrada en el script *«.env»*, cumplido esto, procedemos a ejecutar el script principal de activación. Para ello, diríjase a su entorno virtual y desde su Terminal ejecute el comando líneas abajo, el cual activará el script *«main.py»* conteniendo las líneas del código principal cuya tarea fundamental es integrar y ejecutar las funciones programadas del algoritmo de conteo y clasificación vehicular.

```
python -m main
```

Una vez iniciado el script *«main.py»*, en su Terminal aparecerá el mensaje líneas abajo, el cual significa que se ha cumplido con todos los requerimientos y las funciones han sido ejecutadas con éxito. Asimismo, a partir de este instante se inicia el contabilizador de tiempo en ”0”.

```
═══════════════════════ INICIALIZANDO... \ ═══════════════════════
[2022-03-04 21:30:07,704] INFO: ═══ ¡PROCESO INICIALIZADO! ═══ {}
```

A continuación, se procede a detallar y explicar las líneas de código del script principal *«main.py»*:

Las **líneas 7-9** importan bibliotecas de Python, entre ellas: sys (incluye variables y funciones del intérprete de python), $time$ (función para trabajar con fechas y horas) y $cv2$ (para utilizar la biblioteca de *OpenCV*). Las **líneas 11-12** importan y ejecutan la bibioteca $load\_dotenv$, necesario para leer valores de un archivo formato $.env$, en este caso para cargar el script *«.env»*, el cual contiene los valores de entrada que utilizará el algoritmo (para mayor información leer ítem 3.3.4.2). Las **líneas 14-20** importan scripts del directorio principal, entre ellos, *settings.py, ObjectCounter.py* y de la carpeta *«./util/»*, la cual contiene los scripts *logger.py, image.py, debugger.py y excel.py*.

Las **líneas 22-23** inician un sistema de registro único para el proceso que se analizará, es decir, creará un Id único que identificará el proceso y sus resultados.

```
1  '''
2  Ivy-master (main.py)
3
4  Autor: Nicholas Kajoh
5  Aportes: David Nizama
6  '''
7  import sys
8  import time
9  import cv2
10
11 from dotenv import load_dotenv
12 load_dotenv()
13
14 import settings
15 from util.logger import init_logger
16 from util.image import take_screenshot
17 from util.logger import get_logger
18 from util.debugger import mouse_callback
19 from ObjectCounter import ObjectCounter
20 from util.excel import save_excel
21
22 init_logger()
23 logger = get_logger()
```

A partir de la **línea 26**, hasta la **141**, se define la función run, función principal que importará y ejecutará todos los scripts del algoritmo para detectar y contabilizar vehículos en videograbaciones. Las **líneas 31-37** cargan el video ubicado en la carpeta *«./data/videos/NombreDelVideo.mp4»* o la ruta que se haya definido en el script «$.env$» (Variable: *VIDEO*). De proporcionarse una ruta invalida o el formato del video sea inválido o no se encuentre el archivo, el Terminal imprimirá el siguiente mensaje: Origen de Video Inválido; caso contrario, continuará ejecutándose.

```
26 def run():
27     '''
28     Iniciar el contador de objetos y ejecutar el bucle de conteo
29     '''
30
31     video = settings.VIDEO
32     cap = cv2.VideoCapture(video)
33     if not cap.isOpened():
34         logger.error('Origen de video Invalido %s', video, extra
       ={
35             'meta': {'label': 'INVALID_VIDEO_SOURCE'},
36         })
37         sys.exit()
```

Las **líneas 38-39** extraen las características e información del video, como: frames, alto, ancho. Las **líneas 40-45** cargan los valores y configuración de las variables que fueron definidas en el script «*.env*» relacionados al Tipo de detector, tracker y la región de detección (DROI). De la **línea 46** al **50** se cargan los datos (vertices) para crear el polígono de la región de interés, polígono que será utilizado para identificar y detectar los vehículos que pasen sobre él. Las **líneas 51-52** cargan los datos de las variables de las líneas de conteo, líneas que serán utilizadas para contabilizar los vehículos que crucen sobre ellas. Con los parámetros iniciales cargados y procesados, la **línea 55** inicia el script *«ObjectCounter.py»* y comienza la identificación y conteo de vehículos en el video de entrada.

```
38      retval, frame = cap.read()
39      f_height, f_width, _ = frame.shape
40      detection_interval = settings.DI
41      mcdf = settings.MCDF
42      mctf = settings.MCTF
43      detector = settings.DETECTOR
44      tracker = settings.TRACKER
45      use_droi = settings.USE_DROI
46
47      # Crear poligono de Region de Detección (DROI)
48      droi = settings.DROI \
49              if use_droi \
50              else [(0, 0), (f_width, 0), (f_width, f_height), (0,
        f_height)]
51      show_droi = settings.SHOW_DROI
52      counting_lines = settings.COUNTING_LINES
53      show_counts = settings.SHOW_COUNTS
54      hud_color = settings.HUD_COLOR
55
56      object_counter = ObjectCounter(frame, detector, tracker, droi
        , show_droi, mcdf, mctf, detection_interval, counting_lines,
        show_counts, hud_color)
```

Las **líneas 58-64** inician el proceso de grabar y guardar el video de salida. Si escogió la opción de grabar el conteo, se iniciará la grabación de un video que registrará el conteo y se guardará en la ubicación elegida como ruta de salida (variable *«OUTPUT_VIDEO_PATH»*, en el script «*.env*»).

```
58      record = settings.RECORD
59      if record:
60          # Inicializar un video para Grabar el Conteo
61          output_video = cv2.VideoWriter(settings.OUTPUT_VIDEO_PATH,
62                      \ cv2.VideoWriter_fourcc(*'MJPG'), \ 30, \
63                      (f_width, f_height))
```

Si el algoritmo se ejecutó correctamente hasta este punto y se cargó correctamente las variables y sus valores, en el Terminal y el logger se imprimirá: *«¡PROCESO INICIALIZADO!»*. Asimismo, a partir de este instante se iniciará el contabilizador de tiempo (**línea 68**).

```
66      print ('========== INICIALIZANDO...\ ==========')
67      logger.info('======= ¡PROCESO INICIALIZADO! =======')
68      start=time.time()
```

Las **líneas 70-79** nos permitirán identificar las coordenadas de un pixel dentro de un frame en la ventana de depuración (ventana emergente con el video en proceso) con tan solo clickear sobre ella. Esto nos ayudará a conocer las coordenadas de los pixeles y dibujar el polígono del DROI con mayor exactitud y a nuestra conveniencia.

```
70      headless = settings.HEADLESS
71      if not headless:
72          # Capturar eventos del mouse en la ventana de depuración
73          cv2.namedWindow('Debug')
74          cv2.setMouseCallback('Debug', mouse_callback, {'
        frame_width': f_width, 'frame_height': f_height})
75
76      is_paused = False
77      output_frame = None
78      frames_count = round(cap.get(cv2.CAP_PROP_FRAME_COUNT))
79      frames_processed = 0
```

De la **línea 81** a la **123** es el bucle principal del código para reproducir el video y proceder a identificar y contabilizar los vehículos. Luego de la lectura del video, y en plena ejecución de la ventana de depuración, podemos utilizar diversas teclas que facilitarán el trabajo del usuario en el proceso de detección; por ejemplo, presionar en el teclado la letra "k" pausará el video permitiendo visualizar un punto exacto (frame, segundo exacto, etc.), asimismo, presionar la letra "s" permitirá capturar un screenshot del video, el cual se guardará en la carpeta *«./data/screenshot»*, y la letra "q" finalizará el proceso de detección y procederá a finalizar el video.

```
81      try:
82          # Main Loop - Bucle Principal
83          while retval:
84              k = cv2.waitKey(1) & 0xFF
85              if k == ord('p'): # pause/play, si presiona 'p'
86                  is_paused = False if is_paused else True
87                  logger.info('PAUSAR/REPRODUCIR', extra={'meta':
                      {'EN PAUSA': is_paused,}})
88              if k == ord('s') and output_frame is not None: #tomar
                      screenshot, si presiona 's'
89                  take_screenshot(output_frame)
90              if k == ord('q'): # finalizar video, si presiona 'q'
```

```
91                 logger.info('FINALIZAR', extra={'meta':
                  {'PROCESO': 'FINALIZADO'}})
92                 break
93
94             if is_paused:
95                 time.sleep(0.5)
96                 continue
97
98             _timer = cv2.getTickCount()
```

Las **líneas 100-104** identifican el frame y cuadro delimitador de un objeto identificado y procede a dibujarlo en la ventana de depuración y video de salida para que se visualice. Las **líneas 106-121** cargan y configuran los parámetros de la ventana de depuración para que se visualice mientras se ejecuta el algoritmo. Finalmente, la **línea 123** ejecuta y muestra la ventana de depuración y guarda la información para el video de salida.

```
100            object_counter.count(frame)
101            output_frame = object_counter.visualize()
102
103            if record:
104                output_video.write(output_frame)
105
106            if not headless:
107                debug_window_size = settings.DEBUG_WINDOW_SIZE
108                resized_frame = cv2.resize(output_frame,
                   debug_window_size)
109                cv2.imshow('Debug', resized_frame)
110
111            processing_frame_rate = round(cv2.getTickFrequency()
                       / (cv2.getTickCount() - _timer), 2)
112            frames_processed += 1
113            logger.debug('Frame processed.', extra={
114                'meta': {
115                    'label': 'FRAME_PROCESS',
116                    'frames_processed': frames_processed,
117                    'frame_rate': processing_frame_rate,
118                    'frames_left': frames_count -
                   frames_processed,
119                    'percentage_processed': round((
                   frames_processed / frames_count)*100, 2),
120                },
121            })
122
123            retval, frame = cap.read()
```

Para culminar, las **líneas 124-140** finalizan el proceso de identificación y detección de vehículos, cierran el video (ventana de depuración), guardan la grabación del video de salida y se procede a imprimir en el Terminal y el logger: *«¡PROCESO FINALIZADO!»*. Asimismo, se procede a cerrar el archivo de registro (logger) y a imprimir en el Terminal la data final con las cantidades totales de vehículos contabilizados y se guarda dicha información en un archivo Excel, el cual es guardado en la carpeta *«./data/Excel»*. Finalmente, se imprime el tiempo total de ejecución del algoritmo.

```
124     finally:
125         # Finalizar el video, cerrar la ventana de depruación,
        cerrar el archivo de registro y el contador de objetos.
126         cap.release()
127         if not headless:
128             cv2.destroyAllWindows()
129         if record:
130             output_video.release()
131         logger.info('==== ¡PROCESO FINALIZADO! ====')
132         logger.info('==== RESULTADOS ====', extra={
133             'meta': {'CONTEO FINAL': object_counter.get_counts(),
134 
135             },
136         })
137         end=time.time()
138         print("==== INFO: El proceso duró {:.4f} segundos".
                    format(end - start))
139 
140         save_excel(object_counter.get_counts())
```

Finalmente, definimos que cuando se ejecute el script *«main.py»* se ejecute la función run (línea 26 a 141 del código principal) y con esto iniciar el algoritmo para detectar, clasificar y contabilizar vehículos dentro de un video. De este modo, cumplimos el objetivo principal de la tesis.

```
142 if __name__ == '__main__':
143     run()
```

3.3.4.4 Resultados

Para ejecutar el algoritmo, diríjase a su entorno virtual y desde su Terminal ejecute el comando líneas abajo, el cual activará el script *«main.py»*, conteniendo el código principal para detectar, clasificar y contabilizar vehículos en videos.

```
python -m main
```

Una vez se ejecute *«Ivy»*, el Terminal procesará el video de entrada y emergerá una ventana de depuración mostrando en tiempo real el proceso de detección y clasificación de vehículos. Culminado el tiempo de proceso, se guardará el video procesado en la ubicación definida previamente y se generarán los archivos Excel (**.xls*) y logger (**.log*) conteniendo la información del conteo total de vehículos.

Nota: Más detalle sobre el proceso de detección, resultados, precisiones y casos particulares se verán en el *Capítulo IV y V*, en la que se desarrolla la aplicación de un caso práctico.

A continuación, se presenta un ejemplo de datos de salida (en el *Capítulo IV y V* se desarrollará y comparará con más detalle los resultados obtenidos) que muestra el Terminal mientras se ejecuta el algoritmo en tiempo real utilizando la red neuronal "$yolo$":

```
====================== INICIALIZANDO...\ ======================
[2022-04-29 20:39:53,715] INFO: ==== ¡PROCESO INICIALIZADO! ==== {}
[2022-04-29 20:39:54,631] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '84.97 %'}
[2022-04-29 20:39:54,632] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '80.79 %'}
[2022-04-29 20:39:54,633] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'A', 'CONFIANZA': '76.00 %'}
[2022-04-29 20:39:54,633] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '58.90 %'}
[2022-04-29 20:39:54,634] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '57.52 %'}
[2022-04-29 20:40:00,802] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'A', 'CONFIANZA': '75.80 %'}
[2022-04-29 20:40:01,005] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '66.92 %'}
[2022-04-29 20:40:10,554] INFO    : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '58.03 %'}
[2022-04-29 20:40:10,554] INFO    : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '58.03 %'}
[2022-04-29 20:40:10,555] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '52.13 %'}
[2022-04-29 20:40:11,474] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'A', 'CONFIANZA': '82.08 %'}
```

```
[2022-04-29 20:40:16,883] INFO    : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': 'A', 'CONFIANZA': '56.28 %'}
[2022-04-29 20:40:22,835] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '87.77 %'}
...
...
...
[2022-04-29 20:56:31,615] INFO    : Objeto Contado {'TIPO': 'truck',
     'LÍNEA DE CONTEO': 'B', 'CONFIANZA': '52.13 %'}
[2022-04-29 20:56:53,134] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '52.27 %'}
[2022-04-29 20:57:00,097] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '96.08 %'}
[2022-04-29 20:57:00,271] INFO    : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '93.33 %'}
[2022-04-29 20:57:05,755] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '73.41 %'}
[2022-04-29 20:57:09,931] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '72.82 %'}
[2022-04-29 20:57:21,319] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '81.22 %'}
[2022-04-29 20:58:20,290] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '53.47 %'}
[2022-04-29 21:01:11,180] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '52.01 %'}
[2022-04-29 21:03:32,572] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '54.99 %'}
[2022-04-29 21:03:38,724] INFO    : === ¡PROCESO FINALIZADO! === {}
[2022-04-29 21:03:38,724] INFO    : ========= RESULTADOS ========= {
    'CONTEO FINAL':  {'A': {'car': 58, 'bus': 10, 'truck': 3},
                      'B': {'car': 74, 'bus': 11, 'truck': 2},
                      'C': {'car': 66, 'bus': 14, 'truck': 3}}}
=== TIME ===: El proceso duró 1425.0093 segundos
```

Del ejemplo previo, se observa que el tiempo de procesamiento fue de 1,425 segundos (23.75 minutos) para un video de 1 minuto 41 segundos, es decir, por cada (1) minuto de video que se analiza el algoritmo demora aproximadamente 14.1 minutos en procesarlo.

Asimismo, los vehículos identificados fueron en su mayoría carros, buses y camiones, cuyas confianzas de detección oscilaron entre 51.60 % y 98.20 %, estos porcentajes de confianza representan la probabilidad determinada por el algoritmo (certeza) para afirmar que la tipología del objeto detectado (carro, bus, camión) efectivamente es la correcta. Cabe precisar que, la mayoría de las detecciones fueron verdaderos positivos (se detecto correctamente el vehículo y su tipología); sin embargo, también existieron falsos positivos (se detecto el vehículo, pero no la tipología correcta) y falsos negativos (existió el vehículo, pero no se detecto).

Respecto a la cantidad de líneas de conteo, se pueden crear tantas como sea necesario, esto depende de los flujos vehiculares que se desea contabilizar. Del ejemplo anterior, se definieron tres (3) líneas de conteo: A, B y C; de los resultados, el total de vehículos contabilizados por cada línea es como siguiente a continuación:

TABLA N° 3.5: Resultados de detección y clasificación de vehículos en videos

Línea de Conteo	Tipo de vehículo		
	carros	buses	camión
A	58	10	3
B	74	11	2
C	66	14	3

A continuación, se muestran algunos ejemplos de screentshots tomados de los procesamientos de testeo del algoritmo:

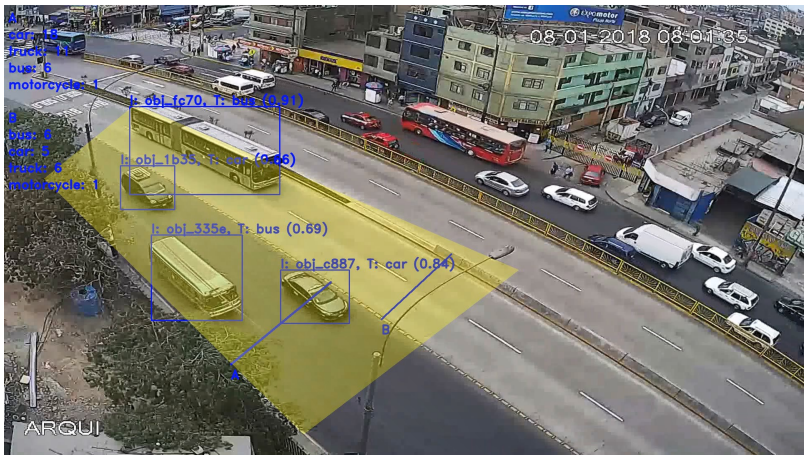


FIGURA N° 3.23: Procesamiento en videos: Algoritmo de Detección y Clasificación Vehicular - Ej. 1

FIGURA N° 3.24: Procesamiento en videos: Algoritmo de Detección y Clasificación Vehicular - Ej. 2

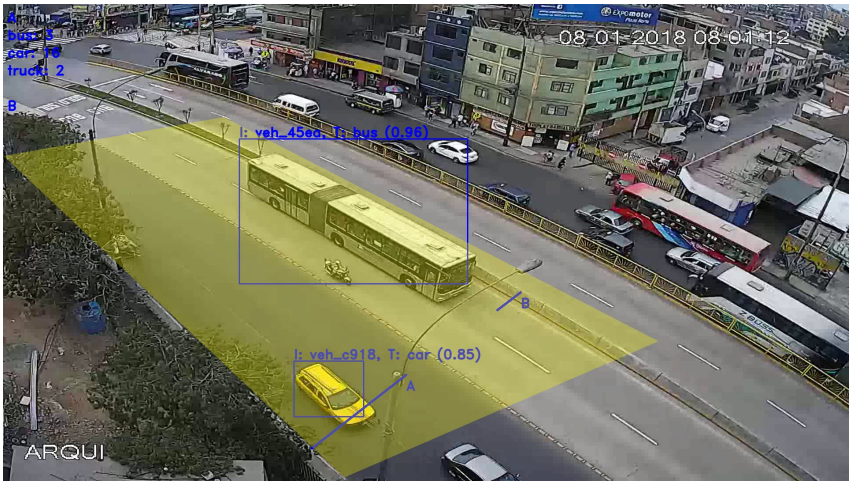


FIGURA N° 3.25: Procesamiento en videos: Algoritmo de Detección y Clasificación Vehicular - Ej. 3

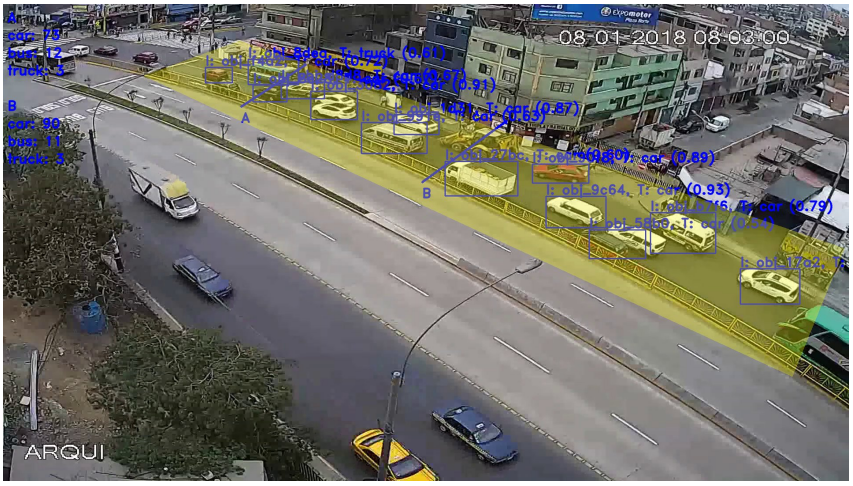


FIGURA N° 3.26: Procesamiento en videos: Algoritmo de Detección y Clasificación Vehicular - Ej. 4

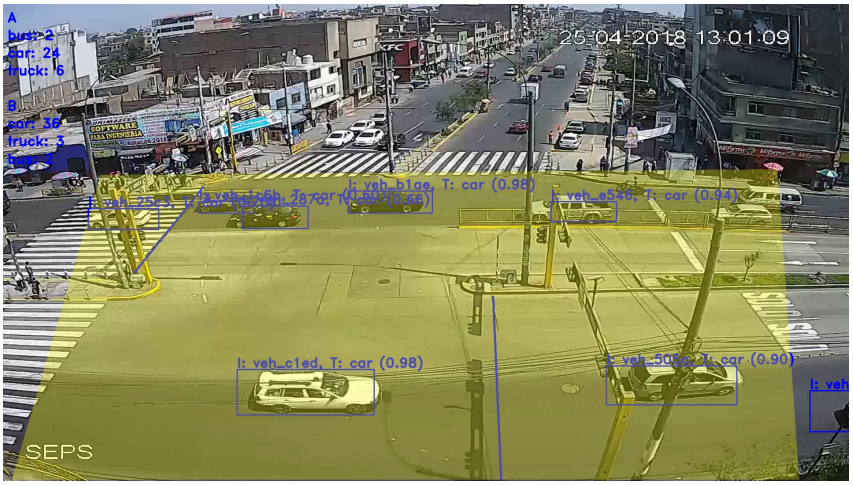


FIGURA N° 3.27: Procesamiento en videos: Algoritmo de Detección y Clasificación Vehicular - Ej. 5

CAPÍTULO IV: PROCESAMIENTO Y OBTENCIÓN DE RESULTADOS

En el presente capítulo se desarrollará la aplicación del algoritmo de detección y clasificación vehicular en una intersección de vías haciendo uso de una grabación (video), posteriormente, se compararán los resultados obtenidos con un conteo manual (visualizando el video). La metodología consiste en: 1) identificar las rutas y direcciones de los flujos vehiculares, 2) obtener y definir las variables de entrada (red neuronal a utilizar, video de entrada, región de detección (DROI), líneas de conteo, entre otros), 3) ejecutar el algoritmo, 4) obtener el aforado total de las vías mediante el uso de las redes neuronales, 5) obtener el aforo vehicular mediante técnica manual y 6) comparación de resultados.

4.1 PROCESO DE DETECCIÓN Y CLASIFICACIÓN VEHICULAR

4.1.1 Condiciones iniciales

El algoritmo fue aplicado en videograbaciones que muestran el flujo vehicular de la intersección de las avenidas Av. Túpac Amaru y Av. Eduardo de Habich, puerta N° 4 de la Universidad Nacional de Ingeneniería (UNI), en el distrito de Rímac, Perú. Ambas vías cuentan con dos (2) sentidos de tres (3) carriles cada uno. En caso de la Av. Túpac Amaru, en medio de ella transita la vía exclusiva de transporte público masivo ”Metropolitano”, vía construida en dos (2) sentidos con dos (2) carriles cada uno, es decir, la Av. Túpac Amaru tiene un total de 5 carriles por sentido, tres (3) de uso general y dos (2) de uso exclusivo para el Metropolitano.

FIGURA N° 4.1: Ubicación de área de estudio: Intersercción Av. Túpac Amaru - Av. Eduardo de Habich, Rímac, Perú. Fuente: Google Earth

Respecto a la grabación, esta fue registrada el día 25 de abril del 2018, en el horario de la 1:00 p.m. El video tiene una duración de 27 minutos y 41 segundos y un ángulo

de visión en dirección a la Av. Eduardo de Habich, por lo que el flujo de vehicular de la Av. Túpac Amaru se visualiza en sentido horizontal. Los tipos de vehículos que más transcurren está vía son: carros, buses, camiones (vehículos pesados) y motocicletas.

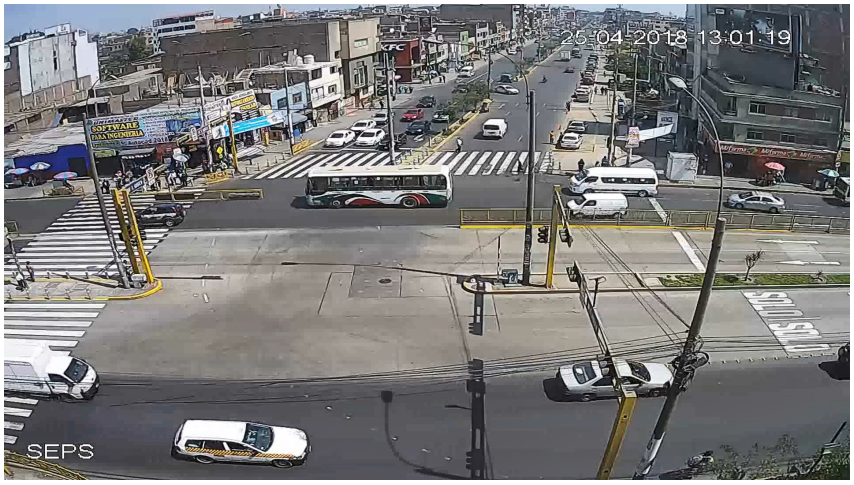


FIGURA N° 4.2: Video de estudio: Inters. Av. Túpac Amaru - Av. Eduardo de Habich, Rímac, Perú.

FIGURA N° 4.3: Imagen 360°: Inters. Av. Túpac Amaru - Av. Eduardo de Habich, Rímac, Perú. Fuente: Street View

Para realizar el aforo vehicular es necesario uniformizar los tamaños de los vehículos a un único vehículo patrón. Para nuestro caso, el *auto* será el vehículo patrón, a partir de él obtendremos las equivalencias de los demás vehículos. Para ello, utilizaremos los factores de conversión Unidad Coche Patrón (UCP), establecidos en el Highway Capacity Manual - HCM, 2010.

TABLA N° 4.1: Unidades de coche patrón (UCP). Fuente: HCM, 2010

Clasificación Vehicular		
Tipo de Servicio	Tipo de Vehículos	Factor UCP
T. Privado	Auto	1
T. Menor	Moto Lineal	0.33
T. Público Regulado	Camioneta Rural	1.25
	Bus	3
T. Carga	Camión	2.5

4.1.2 Rutas y direcciones del flujo vehicular

Se procedió a identificar las rutas y direcciones de flujo vehicular en la zona de estudio. Para la intersección de las avenidas Av. Túpac Amaru y Av. Eduardo de Habich se identificaron un total de siete (7) posibles rutas de flujo vehicular, estas son las correspondientes a (Ver Figura N° 4.4):

- Q1 (en azul): Vehículos que transitan en la Av. Túpac Amaru, provenientes de la zona sur-centro de la ciudad, desde los distritos de Lima, El Agustino y San Juan de Lurigancho.
    - Q11: Continúan de frente en la Av. Túpac Amaru, en dirección a la zona norte de la ciudad, hacia los distritos de Independencia, Los Olivos y Comas.
    - Q12: Se encuentra prohibido girar a la izquierda hacia la Av. Eduardo de Habich. Esta ruta no se tomará en consideración.
    - Q1A: Vía exclusiva de transporte masivo ”Metropolitano”. En dirección al norte de la ciudad de Lima, atraviesa los distritos de Independencia, Los Olivos y Comas.
- Q2 (en naranja): Vehículos que transitan en la Av. Túpac Amaru, provenientes de la zona norte de la ciudad, desde los distritos de Independencia, Los Olivos y Comas.
    - Q21: Continúan de frente en la Av. Túpac Amaru, en dirección a la zona sur-centro de la ciudad, hacia los distritos de Lima, El Agustino y San Juan de Lurigancho.
    - Q22: Giran a la derecha hacia la Av. Eduardo de Habich, en dirección a la zona oeste de la ciudad, hacia los distritos de San Martín de Porres, Cercado y Callao.
    - Q2A: Vía exclusiva de transporte masivo ”Metropolitano”. En dirección al sur-centro de la ciudad de Lima, atraviesa los distritos del centro de Lima, Lince, San Isidro, Surquillo, Miraflores y llega hasta Barranco.
- Q3 (en verde): Vehículos que transitan en la Av. Eduardo de Habich, provenientes de la zona oeste de la ciudad, desde los distritos de San Martín de Porres, Cercado y Callao.
    - Q31: Giran a la izquierda hacia la Av. Túpac Amaru, en dirección a la zona norte de la ciudad, hacia los distritos de Independencia, Los Olivos y Comas.

- Q32: Giran a la derecha hacia la Av. Túpac Amaru, en dirección a la zona sur-centro de la ciudad, hacia los distritos de Lima, El Agustino y San Juan de Lurigancho.

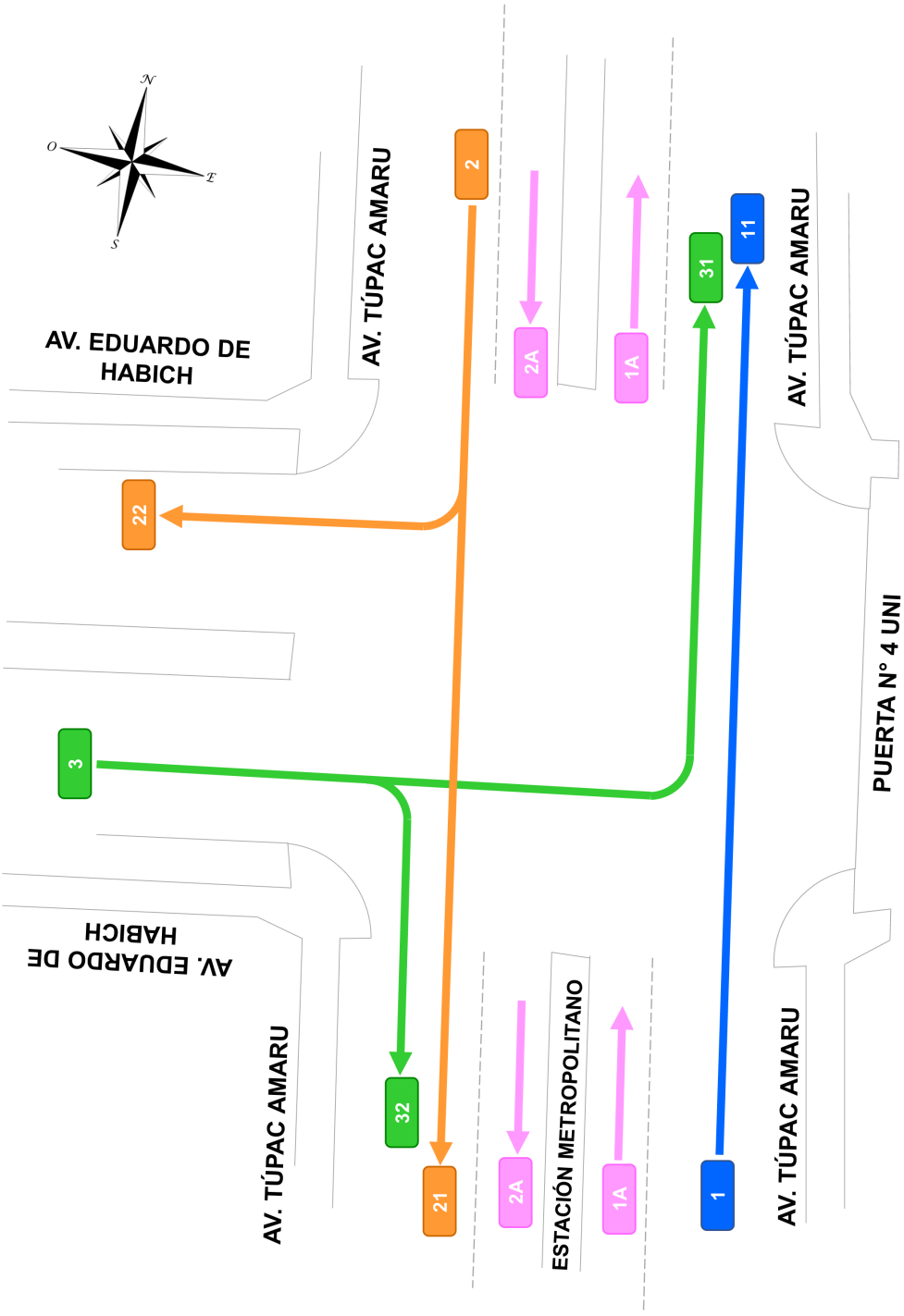


FIGURA N° 4.4: Rutas y direcciones de flujo vehiculares: Intersección Av. Túpac Amaru - Av. Eduardo de Habich, Rímac, Perú. Fuente: Propia

4.1.3 Líneas de Conteo

Una vez conocidas las rutas y direcciones del flujo vehicular (Ver Figura N° 4.4) se procede a definir las líneas de conteo, es decir, los contadores vehiculares. Para el caso en estudio, y con el angulo de enfoque del video, se consideró crear cinco (5) líneas de conteo, estas son (Ver Figura N° 4.5, líneas de conteo en color rojo):

- Línea de Conteo ’1’ (LC: ’1’): Identifica y contabiliza los vehículos de la Av. Túpac Amaru (Q11), provenientes de la zona sur-centro de la ciudad de Lima, de los distritos de Lima (centro), El Agustino y San Juan de Lurigancho. Estos vehículos continúan de frente por la Av. Túpac Amarú, en dirección a la zona norte, hacia los distritos de Independencia, Los Olivos y Comas.
- Línea de Conteo ’2’ (LC: ’2’): Identifica y contabiliza los vehículos de la Av. Túpac Amaru (Q21), provenientes de la zona norte de la ciudad de Lima, de los distritos de Independencia, Los Olivos y Comas, los cuales continúan de frente por la Av. Túpac Amarú (Q21), en dirección a la zona sur-centro, hacia los distritos de Lima (centro), El Agustino y San Juan de Lurigancho
- Línea de Conteo ’B’ (LC: ’B’): Identifica y contabiliza los vehículos que provienen de la Av. Túpac Amaru, de norte a sur, y giran a la derecha, hacia la Av. Eduardo de Habich (Q21). Provienen de la zona norte de la ciudad de Lima, de los distritos de Independencia, Los Olivos y Comas, y giran hacia la Av. Eduardo de Habich, en dirección a la zona oeste, hacia los distritos de San Martín de Porres, Cercado y Callao.
- Línea de Conteo ’3’ (LC: ’3’): Identifica y contabiliza los vehículos de la Av. Eduardo de Habich (Q3), provenientes de la zona oeste de la ciudad de Lima, de los distritos de San Martín de Porres, Cercado y Callao, los cuales pueden eligir entre:
  a. Girar a la izquierda hacia la Av. Túpac Amarú (Q31), en dirección a la zona norte, hacia los distritos de Independencia, Los Olivos o Comas.
  b. Girar a la derecha hacia la Av. Túpac Amarú (Q32), en dirección a la zona sur-centro, hacia los distritos de Lima (centro), El Agustino o San Juan de Lurigancho.
- Línea de Conteo ’C’ (LC: ’C’): Identifica y contabiliza los vehículos de la Av. Eduardo de Habich que giran a la izquierda hacia la Av. Túpac Amarú (Q31). Estos vehículos provienen de la zona oeste de la ciudad de Lima, de los distritos de San Martín de Porres, Cercado y Callao y se dirigen hacia la zona norte de la ciudad, hacia los distritos de Independencia, Los Olivos y Comas.

En caso se demande obtener el flujo vehicular por sentido de los carriles de la vía de transporte público ”Metropolitano”, ubicada en medio de la Av. Túpac Amaru, se recomienda crear dos (2) líneas de conteo adicionales (Ver Figura N° 4.5, líneas de conteo en color rojo). Estas líneas contabilizarían lo siguiente:

- Línea de Conteo ’M1’ (LC: ’M1’): Contabiliza el flujo vehicular Q1A. Identifica los buses metropolitanos que provienen de la zona sur-centro de la ciudad y que se dirigen hacia el norte de Lima. Los distritos destinos son: Independencia, Los Olivos y Comas.
- Línea de Conteo ’M2’ (LC: ’M2’): Contabiliza el flujo vehicular Q2A. Identifica los buses metropolitanos que provienen de la zona norte de la ciudad y que se dirigen hacia el sur-centro de Lima. Los distritos destinos son: Lima (centro), Lince, San Isidro, Surquillo, Miraflores y Barranco.

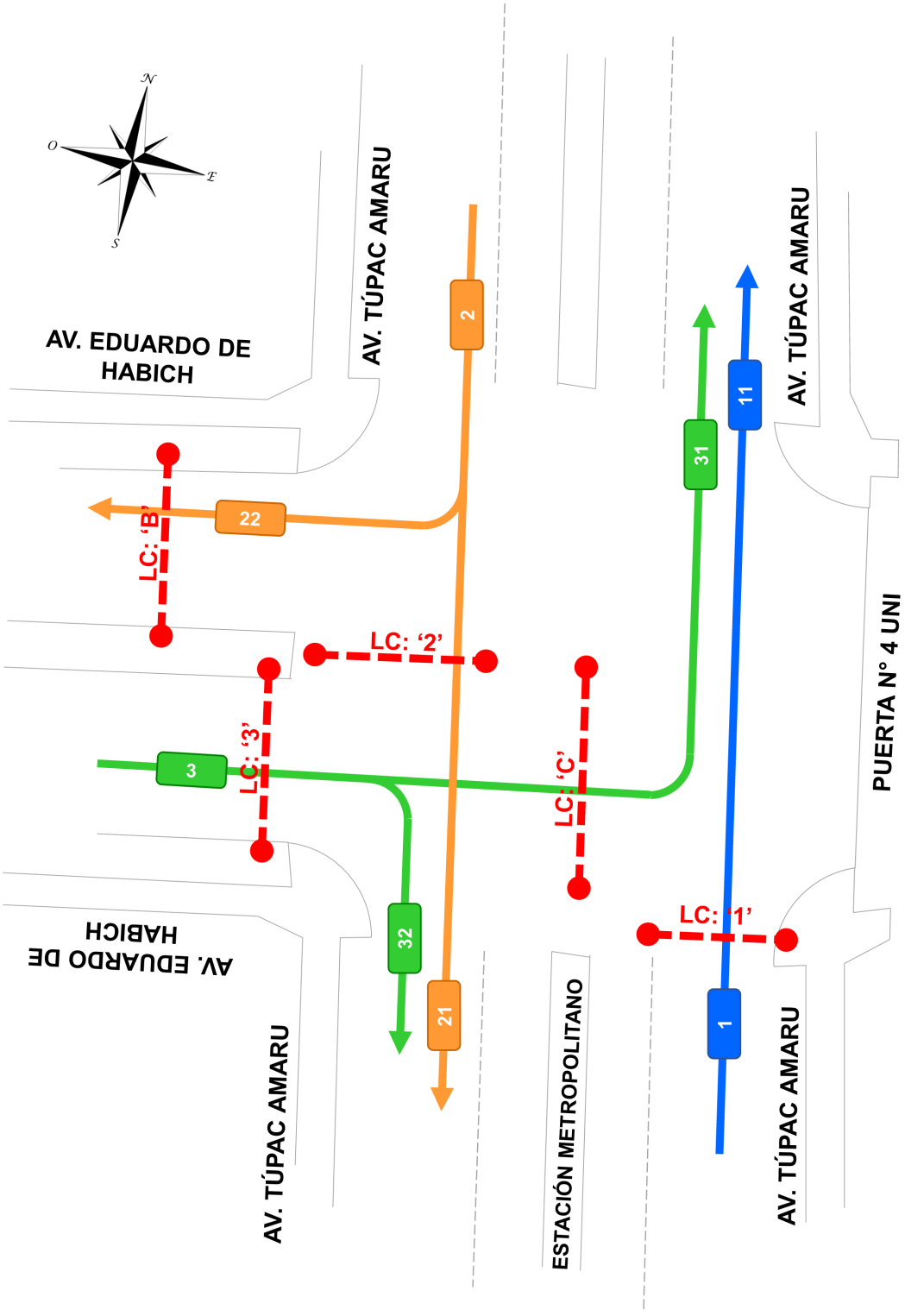


FIGURA N° 4.5: Líneas de conteo vehícular, en rojo. Intersección Av. Túpac Amaru - Av. Eduardo de Habich, Rímac, Perú. Fuente: Propia

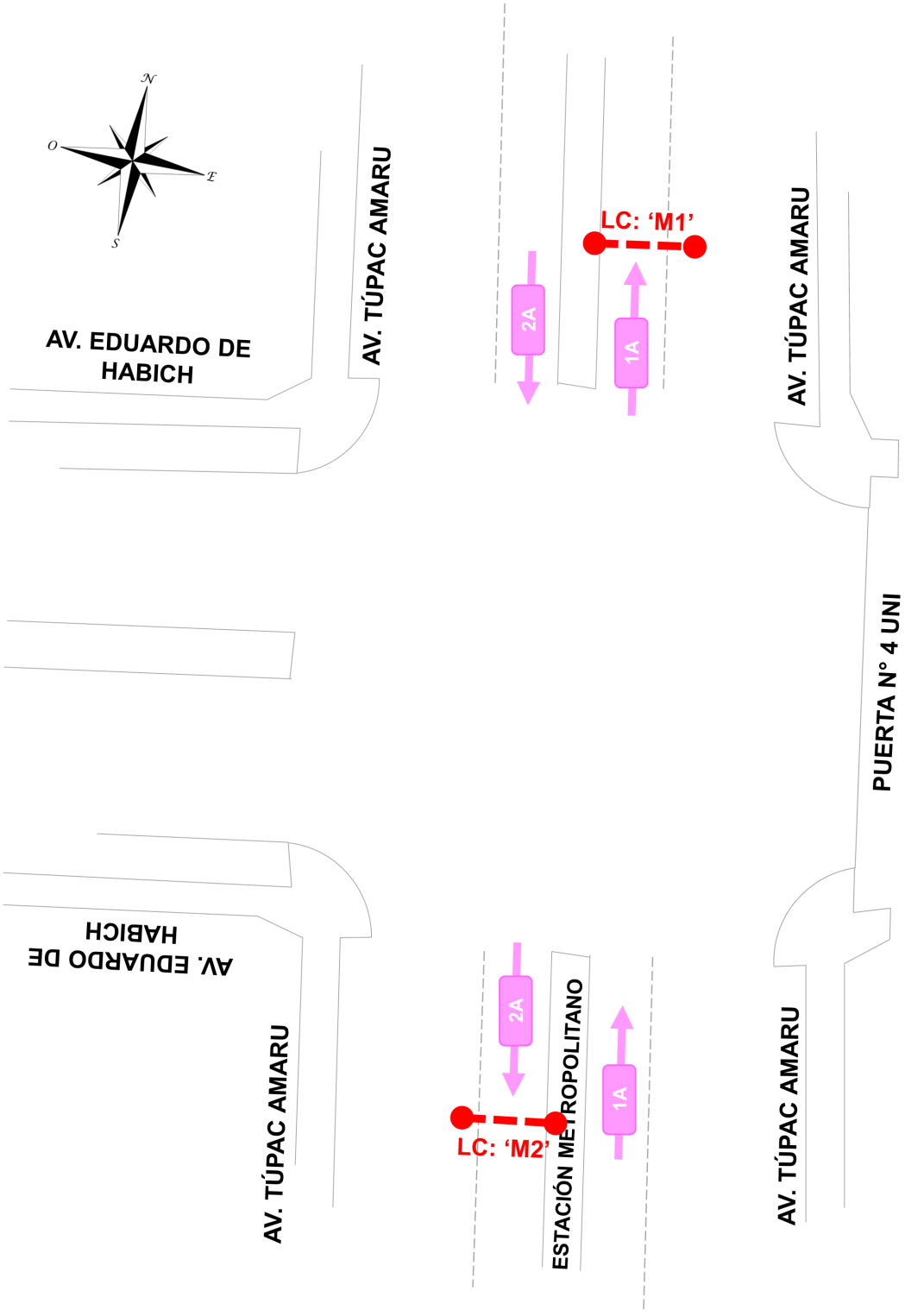


FIGURA N° 4.6: Metropolitano: Líneas de conteo vehícular, en rojo. Intersección Av. Túpac Amaru - Av. Eduardo de Habich, Rímac, Perú. Fuente: Propia

4.1.4 Variables de entrada

Previamente a ejecutar el algoritmo de detección y clasificación vehicular se deben introducir los valores de las variables de entrada, los cuales serán obtenidos de acuerdo a las condiciones del caso particular que se analizará, para mayor información revisar las secciones 3.3.4.1. y 3.3.4.2. de la presente tesis.

a. Video de Entrada

   Guardamos el video que se examinará en la carpeta *«./data/videos/»*. Posteriormente, procedemos abrir el script «$.env$» y en la variable *VIDEO* introducimos la ruta absoluta del video a procesar; por ejemplo, para nuestro caso, el video que analizaremos tiene una duración total de 27 minutos con 41 segundos y lleva por nombre *«Habich_TupacAmaru.mp4»*, por lo que, la ruta absoluta quedaría definida de la siguiente manera:

   ```
   VIDEO = "D:/UNI/TESIS/ivymaster/data/videos/Habich_TupacAmaru.mp4"
   ```

b. Video de Salida

   En el script «$.env$», ubicamos la variable *OUTPUT_VIDEO_PATH* (línea 13) e introducimos la ruta relativa donde se guardará el video procesado, se recomienda que el video sea guardado en la misma carpeta del video de entrada. Para nuestro caso, haremos que el video de salida sea guardado bajo el nombre de: *«Habich_TupacAmaru_Procesado.avi»* y en formato *«.avi»* (por cuestiones de peso del archivo), por lo que la variable quedaría definida de la siguiente manera:

   ```
   OUTPUT_VIDEO_PATH = "./data/videos/Video_Procesado.avi"
   ```

c. Red Neuronal

   Para fines del presente ejemplo de aplicación, se utilizarán las redes neuronales *«YoloV3»*, desarrollado por Joseph Redmon y Ali Farhadi en el 2018 (ver sección 3.3.2.1) y textit«Detectron2», desarrollado por Facebook AI Research en 2019 (ver sección 3.3.2.2), las cuales han dado buenos resultados en precisión y tiempo de proceso a comparación de otras redes.

   Para definir la red reuronal que procesará el video, en el script «$.env$», ubique la variable *DETECTOR* (línea 10) y cambie su valor a: "$yolo$" o "$dectectron2$". Para el caso de utilizar YoloV3, la variable quedaría definida de la siguiente manera:

   ```
   DETECTOR = "yolo"
   ```

Los archivos y rutas relativas de los pesos pre-entrenados, configuración, clases de detección y confianza de las red neuronales se encuentran definidos en el script «$.env$» líneas abajo. No es necesario modificar estos valores, solo seguir los pasos de instalación descritos en las secciones 3.3.2.1 y 3.3.2.2.

d. Creación de Líneas de Conteo

Definidas las líneas de conteo en la Sección 4.1.3, se procede a crearlas en el algoritmo. Para ello, es necesario conocer las coordenadas de los extremos de cada línea. Un manera de encontrar estas coordenadas es ejecutar el algoritmo y en la ventana de depuración clickear en las ubicaciones dónde queremos que se ubiquen los extremos de las líneas, inmediatamente el algoritmo proporcionará en el Terminal las coordenadas de los puntos clickeados.

Para el caso del video en estudio, y siguiendo los pasos mencionados previamente, los extremos de las líneas de conteo se encuentran ubicados en las siguientes coordenadas:

TABLA N° 4.2: Coordenadas de los extremos de las líneas de conteo, para las vías generales. Fuente: Propia

Líneas de Conteo	Coordenadas	
	(X_1, Y_1)	(X_2, Y_2)
LC '1': Q_{11}	$(794, 858)$	$(768, 1072)$
LC '2': Q_{21}	$(895, 381)$	$(880, 502)$
LC 'B': Q_{22}	$(1055, 253)$	$(1279, 265)$
LC '3': Q_{31} + Q_{32}	$(869, 359)$	$(618, 361)$
LC 'C': Q_{31}	$(450, 638)$	$(932, 632)$

TABLA N° 4.3: Coordenadas de los extremos de las líneas de conteo, para las vías del Metropolitano. Fuente: Propia

Líneas de Conteo	Coordenadas	
	(X_1, Y_1)	(X_2, Y_2)
M_1: Q_{1A}	$(1255, 637)$	$(1310, 840)$
M_2: Q_{2A}	$(338, 627)$	$(420, 503)$

Esta información es introducida al script «$.env$», específicamente, en la variable *COUNTING_LINES* (línea 16). Se define el nombre de la línea de conteo y se precisan sus coordenadas; por ejemplo, para la primera línea de conteo sería: *{'label': 'LC: 1', 'line': [(647, 858), (590, 1072)] }*, sucesivamente se introducen los valores y coordenadas de las demás líneas de conteo.

Finalmente, para el caso de análisis de la intersección en general y considerando las seis (6) líneas de conteo (Ver Figura N° 4.5), la variable *COUNTING_LINES* quedaría definida de la siguiente manera:

```
COUNTING_LINES=[
{'label': '1', 'line': [(794, 858), (768, 1072)]},
{'label': '2', 'line': [(895, 381), (880, 502)]},
{'label': 'B', 'line': [(1055, 253), (1279, 265)]},
{'label': '3', 'line': [(869, 359), (618, 361)]},
{'label': 'C', 'line': [(450, 638), (932, 632)]}]
```

Ahora bien, para cuando se procese el algoritmo solo para las vías del Metropolitano (Ver Figura N° 4.6), la variable quedaría modificada de la siguiente manera:

```
COUNTING_LINES = [
{'label': 'M1', 'line': [(1255, 637), (1310, 840)]},
{'label': 'M2', 'line':  [(338, 627), (420, 503)]}]
```

e. Región de Interés de Detección (DROI)

Seguidamente, se procede a definir el área de la Región de Interés de Detección (del inglés *Detection Region of Interest (DROI)*. Esta región es de suma importancia debido a que el algoritmo centralizará su proceso de identificación y detección de vehículos solo en esta área, obviando los vehículos que se encuentren o transiten fuera de ella, esto se hace con el fin de que el usuario introduzca a su conveniencia y particularía del caso los flujos vehiculares de la vía y de, además, ahorrar recursos del computador para un proceso más ligero y rápido.

Para definir el DROI es necesario obtener las coordenadas de los extremos del polígono que lo conforma, para ello, similar al proceso de obtención de coordenadas de los extremos de las líneas de conteo, se procede a ejecutar el algoritmo y en la ventana de depuración se clickea en los lugares donde se ubicarán los vértices del polígono a fin de obtener sus coordenadas.

Para el caso del video en estudio, se trabajará con dos (2) polígonos DROI, uno para la detección y conteo de los vehículos de todas las vías en general de la intersección y el otro para el transporte exclusivo de la vía del Metropolitano.

A continuación, se presentan los vértices de los polígonos DROI que serán utilizados en el ejemplo de aplicación. Para más detalle, ver la Figura N° 4.7 y Figura N° 4.8, en las que se muestran las formas de los polígonos DROI (en color amarillo transparente).:

TABLA N° 4.4: Coordenadas de los extremos del polígono DROI, para las vías en generales. Fuente: Propia

Vertices del DROI	Coordenadas (X_1, Y_1)
V_1	$(5, 1070)$
V_2	$(5, 418)$
V_3	$(479, 299)$
V_4	$(898, 130)$
V_5	$(1312, 149)$
V_6	$(1248, 314)$
V_7	$(1867, 337)$
V_8	$(1915, 1077)$

TABLA N° 4.5: Coordenadas de los extremos del polígono DROI, para las vías del Metropolitano. Fuente: Propia

Vertices del DROI	Coordenadas (X_1, Y_1)
V_1	$(14, 870)$
V_2	$(352, 427)$
V_3	$(1671, 418)$
V_4	$(1871, 827)$

Esta información debe ser introducida al script «$.env$», específicamente, en la variable *DROI* (línea 2). la cual quedaría definida de la siguiente manera:

```
DROI=[(5, 1070), (5, 418), (479, 299), (898, 130), (1312, 149),
    (1248, 314), (1867, 337), (1915, 1077)]
```

Asimismo, para cuando se proceda a identificar y contabilizar los buses de la vía Metropolítano, es posible realizarlo dentro del proceso principal de identificación de todos los vehículos en la intersección, ya que el DROI del vía Metropolitano se encuentra dentro del DROI general; es decir, podríamos realizar todo el conteo en un solo proceso y solo sería necesario crear las líneas de conteo $M1$ y $M2$.

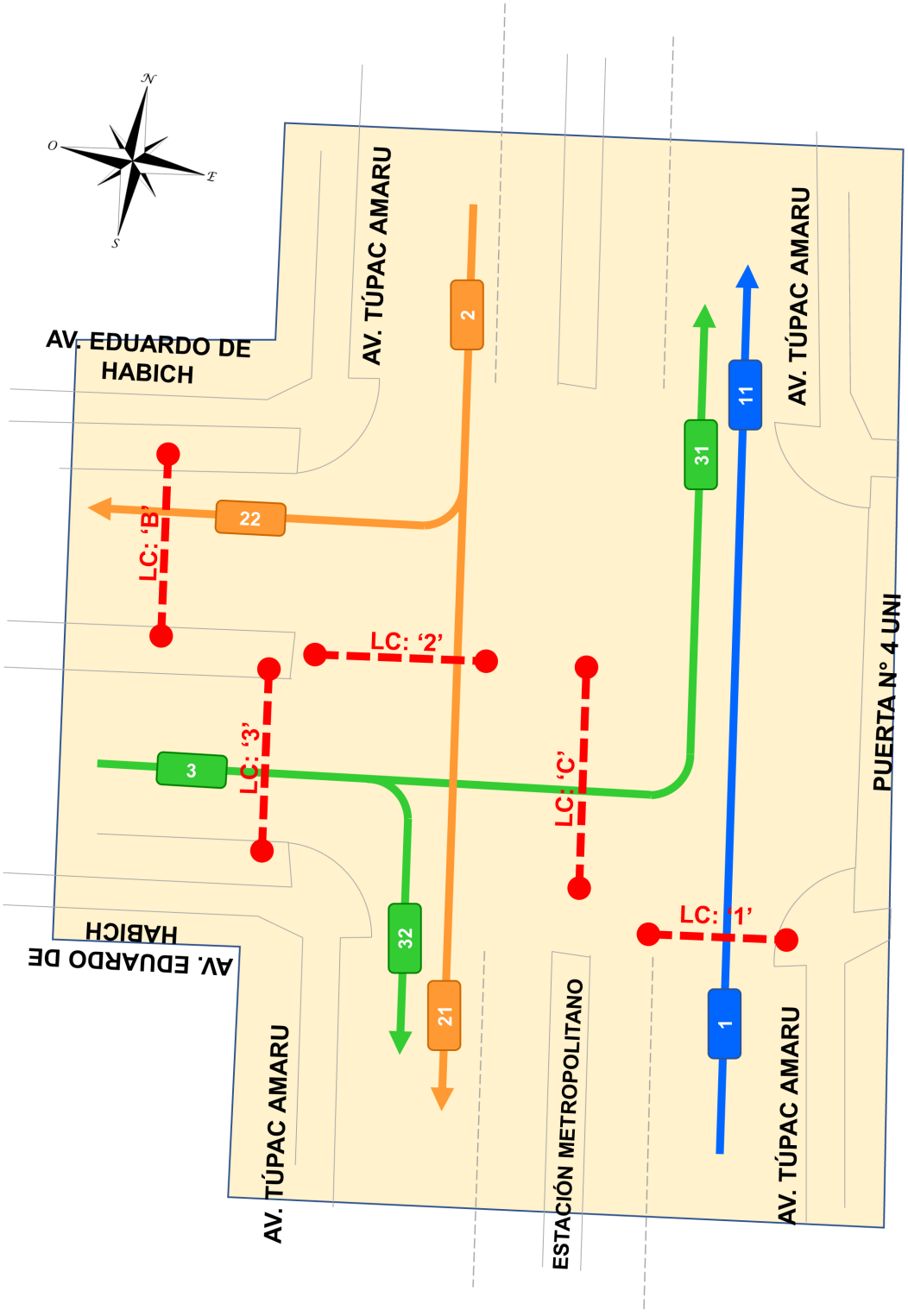


FIGURA N° 4.7: Región de Interés de Detección (DROI), en amarillo transparente. Intersección Av. Túpac Amaru - Av. Eduardo de Habich, Rímac, Perú. Fuente: Propia

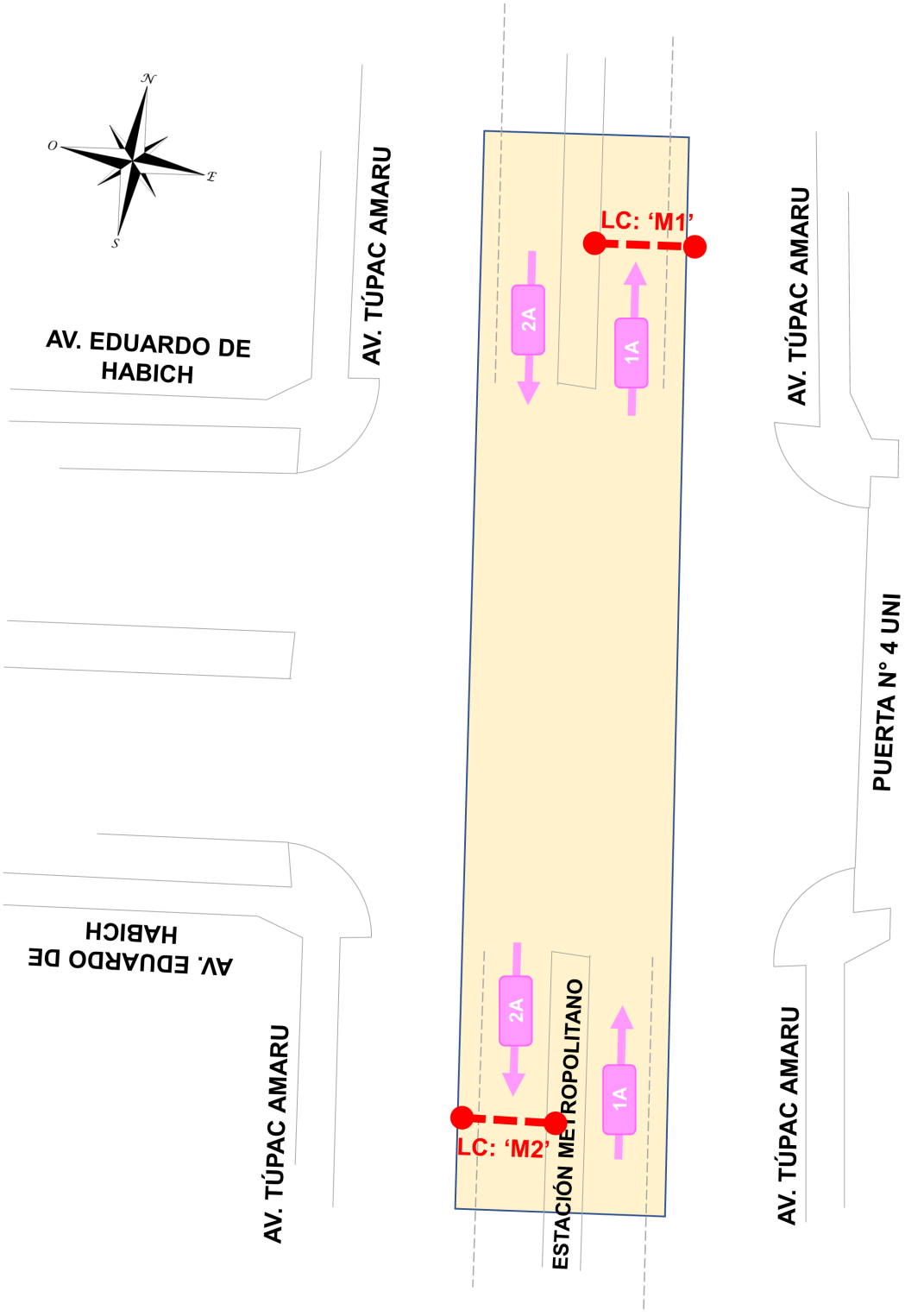


FIGURA N° 4.8: Metropolitano: Región de Interés de Detección (DROI), en amarillo transparente. Intersección Av. Túpac Amaru - Av. Eduardo de Habich, Rímac, Perú. Fuente: Propia

4.1.5 Detección y clasificación vehicular

Luego de determinar y fijar los valores de entrada, procedemos a ejecutar el algoritmo de detección y clasificación vehicular. Para ello, desde el Terminal se debe ejecutar el comando líneas abajo (para mayor información ver la Sección 3.3.4.4 de la presente tesis).

```
python -m main
```

4.1.5.1 Etapas del proceso

Es importante distinguir y detallar las tres (3) etapas que componen el algoritmo cuando este es ejecutado:

1. Detección de vehículos: Es el proceso del algoritmo en la que se identifican los vehículos que se encuentren dentro de la Región de Interés de Detección - DROI, se determina a qué tipo de vehículo corresponden y la posición en la que se encuentran.
2. Seguimiento o *tracking* de vehículos: Es la etapa en la que el algoritmo realiza el seguimiento de la posición de los vehículos identificados en cada instante del video, considerando que se trata del mismo objeto moviéndose a través del plano.
3. Conteo vehicular: Es la parte en la que un vehículo identificado, con su tipología respectiva, cruza una línea de conteo y aumenta el contador vehícular. Para que un vehículo sea contado solo basta que una parte de su cuadro delimitador toque o cruce la línea de conteo.

Estas etapas se realizan de manera simultanea en todo instante del video, por lo que el tiempo del proceso depende principalmente del procesador del equipo de computo y la red neuronal entrenada que se utilizará.

4.1.5.2 Ejecución del algoritmo

Una vez ejecutado el algoritmo, se iniciará el proceso de detección, clasificación, seguimiento y conteo vehicular sobre el video de entrada, asimismo, se emergerá una ventana (ventana de depuración) en el cual se mostrará en tiempo real todo el proceso. Si todo fue realizado con éxito, en el Terminal aparecerá el siguiente mensaje:

```
====================== INICIALIZANDO...\ ======================
[2022-05-08 15:48:18,726] INFO: ==== ¡PROCESO INICIALIZADO! ==== {}
```

4.1.5.3 Proceso del algoritmo

A continuación, se muestra una imagen de la ventana de depuración (Figura N° 4.9) en la que se visualizan las cinco (5) líneas de conteo creadas: {'1', '2', 'B', '3', 'C'}; así como, la Región de Interés de Detección - DROI, en amarillo trasparente. Las coordenadas de los elementos son los mismos que se detallaron en la Sección 4.1.4 Variables de entrada, de la presente tesis.

FIGURA N° 4.9: Video de estudio y sus elementos: Líneas de conteo '1', '2', 'B', '3', 'C', en azul, y Región de Interés de Detección (DROI), en amarillo transparente. Fuente: Propia

Asimismo, como se visualiza en la Figura N° 4.9 y Figura N° 4.10, en el lado izquierdo de la ventana emergente se muestra en tiempo real el contador de vehículos según su tipología por cada línea de conteo.

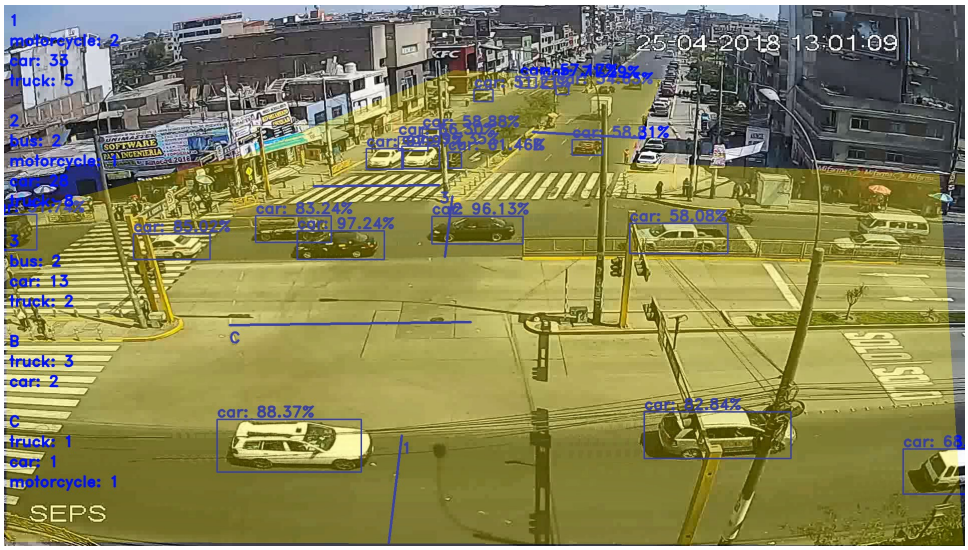


FIGURA N° 4.10: Proceso de detección, clasificación y conteo vehicular. Fuente: Propia

4.1.6 Resultados obtenidos

Una vez ejecutado el algoritmo, en el Terminal irá apareciendo información de los vehículos contabilizados con su respectiva tipología (carro, bus, camión, etc.), confianza de detección y línea de conteo por el que fue contado. Culminado el tiempo de ejecución, se guardará el video procesado en la carpeta elegida y se generarán los archivos Excel (**.xls*) y logger (**.log*) conteniendo información del conteo total de vehículos, sus tipologías y el total de vehículos por cada línea de conteo.

Como se especificó anteriormente en la Sección 4.1.4 Variables de Entrada, el video analizado tiene una duración de 27 minutos y 41 segundos y se utilizaron las redes neuronales *«YoloV3»* y *«Detectron2»*. A continuación, se muestran los resultados de aplicar cada una de las redes neuronales sobre el video.

4.1.6.1 YoloV3

Para ejecutar el algoritmo utilizando la red neuronal *«YoloV3»*, la variable *DETECTOR* (línea 10), del script «$.env$», debe tener su valor en: "$yolo$".

A continuación, se presentan algunos datos de salida obtenidos del Terminal mientras se ejecutaba el algoritmo (se muestran las primeras 20 líneas, 10 intermedias y 6 últimas, de un total de 1,696 que se obtuvieron). Entre la información obtenida por cada vehículo contado se tiene: tipo de vehículo, línea de conteo que contabilizó el vehículo y la confianza que tiene el algoritmo del objeto detectado.

```
====================== INICIALIZANDO...\ ======================
[2022-02-11 17:48:01,994] INFO     : Objeto Contado {'TIPO': 'truck',
     'LÍNEA DE CONTEO': '3', 'CONFIANZA': '85.89 %'}
[2022-02-11 17:48:04,599] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '64.32 %'}
[2022-02-11 17:48:31,525] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '57.49 %'}
[2022-02-11 17:48:38,450] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '68.13 %'}
[2022-02-11 17:48:38,979] INFO     : Objeto Contado {'TIPO': '
    motorcycle', 'LÍNEA DE CONTEO': '1', 'CONFIANZA': '69.07 %'}
[2022-02-11 17:48:59,546] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '82.21 %'}
[2022-02-11 17:49:04,309] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '74.79 %'}
[2022-02-11 17:49:17,716] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '58.57 %'}
[2022-02-11 17:49:23,075] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '71.88 %'}
[2022-02-11 17:49:25,537] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '78.30 %'}
```

```
[2022-02-09 14:59:49,490] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '58.96 %'}
[2022-02-09 14:59:55,497] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '64.31 %'}
[2022-02-09 14:59:59,801] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '82.21 %'}
[2022-02-09 15:00:04,471] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '52.49 %'}
[2022-02-09 15:00:04,828] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '74.79 %'}
[2022-02-09 15:00:06,754] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '59.29 %'}
[2022-02-09 15:00:07,814] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '84.11 %'}
[2022-02-09 15:00:18,317] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '58.57 %'}
[2022-02-09 15:00:22,881] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '71.88 %'}
[2022-02-09 15:00:22,881] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '72.48 %'}
...
...
...
...
...
[2022-02-11 19:11:52,388] INFO     : Objeto Contado {'TIPO': 'truck',
     'LÍNEA DE CONTEO': '1', 'CONFIANZA': '67.70 %'}
[2022-02-11 19:11:54,917] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '99.62 %'}
[2022-02-11 19:11:54,919] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '95.93 %'}
[2022-02-11 19:11:56,297] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '91.58 %'}
[2022-02-11 19:11:59,854] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '87.64 %'}
[2022-02-11 19:12:03,763] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '1', 'CONFIANZA': '87.45 %'}
[2022-02-11 19:12:05,093] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '91.38 %'}
[2022-02-11 19:12:09,435] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '68.45 %'}
[2022-02-11 19:12:09,436] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '57.40 %'}
[2022-02-11 19:12:13,060] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '57.68 %'}
...
...
...
...
```

```
...
...

[2022-02-11 20:37:21,634] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '85.11 %'}
[2022-02-11 20:37:27,562] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '99.46 %'}
[2022-02-11 20:37:30,311] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '98.55 %'}
[2022-02-11 20:37:31,400] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '86.42 %'}
[2022-02-11 20:37:41,064] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '83.94 %'}
[2022-02-11 20:37:42,932] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '83.20 %'}
[2022-02-11 20:38:29,587] INFO     : === ¡PROCESO FINALIZADO! === {}
[2022-02-11 20:38:29,588] INFO     : ========= RESULTADOS =========
{'CONTEO FINAL': {
  '1': {'car': 504, 'bus': 21, 'truck': 57, 'motorcycle': 18},
  '2': {'car': 589, 'bus': 57, 'truck': 72, 'motorcycle': 23},
  'B': {'car': 50, 'bus': 1, 'truck': 18},
  '3': {'car': 185, 'bus': 22, 'truck': 28, 'motorcycle': 10},
  'C': {'car': 42, 'motorcycle': 7}}}
=== TIME ===: El proceso duró 10227.7622 segundos
```

El proceso inició a las 17:48:01 horas y finalizó a las 20:38:29 horas del mismo día, resultando una duración total de procesamiento de 2 horas 50 minutos 28 segundos (10,227.762 segundos). El análisis fue realizado en un video de 27 minutos 41 segundos, por lo que, por cada un (1) minuto de video el algoritmo tardó aproximadamente 6 minutos con 9 segundos en procesarlo, o también, por cada un (1) segundo de video, se tardó 6.1 segundos procesarlo.

TABLA N° 4.6: Tiempo de procesamiento del algoritmo utilizando la red neuronal YoloV3. Fuente: Propia

Red Neuronal	Tiempo Video Real	Tiempo Procesado
$YoloV3$	27 min 41seg	2 h 50 min 28 seg
	1 min	6 min 9 seg
	1 seg	6.1 seg

De la Tabla N° 4.6 se concluye que utilizando la red neuronal *«YoloV3»* el tiempo de procesamiento equivale aproximadamente a seis (6) veces la duración del video que se analizará. Por ejemplo, un video de una (1) hora se procesará en seis (6) horas y uno tres (3) horas, en dieciocho (18) horas.

Al finalizar el proceso, se contabilizó un total de 1,704 vehículos, los cuales se encuentran distribuidos entre las cinco (5) líneas de conteo. La línea de conteo *'1'* identificó un total de 600 vehículos, que convertidos a Unidades Coche Patrón (UCP) son 715 vehículos. La línea *'2'*, un total de 741 vehículos, que en UCP son 948 vehículos. La línea *'B'*, un total de 69 vehículos, que en UCP son 98 vehículos. La línea *'3'*, un total de 245 vehículos, que en UCP son 324 vehículos. Y la línea *'C'*, un total de 49 vehículos, que en UCP son 44 vehículos. Para convertir los vehículos a UCP se utilizaron los factores de conversión especificados en la Tabla N° 4.1.

De acuerdo al reporte final, para la red neuronal *«YoloV3»*, se detalla a continuación los vehículos contabilizados por cada línea de conteo y según su tipología:

TABLA N° 4.7: Conteo total de vehículos utilizando la red neuronal YoloV3. Fuente: Propia

Líneas de Conteo	Tipo de vehículo				Total de vehículos	Vehículos UCP
	carros	buses	camión	motoc.		
1	504	21	57	18	600	715
2	589	57	72	23	741	948
B	50	1	18	0	69	98
3	185	22	28	10	245	324
C	42	0	0	7	49	44
TOTAL	1,370	101	175	58	1,704	2,130

Nota: En la siguiente sección (Sección 4.1.7) se comparan los resultados obtenidos (por el algoritmo) con un aforo manual obtenido directo del video y se determina la precisión (aciertos) de los conteos para cada red utilizada (Sección 5.1).

Para visualizar el video final obtenido del proceso de detección, clasificación y conteo vehicular utilizando la red neuronal *«YoloV3»*, ingrese al siguiente enlace: https://t.ly/Ldf6 o escanee el código QR de la Figura N° 4.12.

FIGURA N° 4.11: Código QR para visualizar el video procesado con la red YoloV3

4.1.6.2 Detectron2

Para ejecutar el algoritmo utilizando la red neuronal *«Detectron»*, se debe modificar la variable *DETECTOR* (línea 10), del script «.*env*», al valor de: "*detectron2*".

A continuación, se presentan algunos datos de salida obtenidos del Terminal mientras se ejecutaba el algoritmo con la red Detectron (se muestran las primeras 20 líneas, 10 intermedias y las últimas 15, de un total de 1,811 vehículos contabilizados).

```
====================== INICIALIZANDO...\ ======================
[2022-02-12 08:54:01,063] INFO: ==== ¡PROCESO INICIALIZADO! ==== {}
[2022-02-12 08:54:01,989] INFO    : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '97.10 %'}
[2022-02-12 08:54:01,990] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '52.03 %'}
[2022-02-12 08:54:24,906] INFO    : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '95.19 %'}
[2022-02-12 08:55:02,067] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '52.13 %'}
[2022-02-12 08:55:03,512] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '70.73 %'}
[2022-02-12 08:55:31,479] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '51.26 %'}
[2022-02-12 08:55:37,628] INFO    : Objeto Contado {'TIPO': '
    motorcycle', 'LÍNEA DE CONTEO': '1', 'CONFIANZA': '78.01 %'}
[2022-02-12 08:55:42,693] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '56.01 %'}
[2022-02-12 08:56:08,414] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '52.17 %'}
[2022-02-12 08:56:26,758] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '79.50 %'}
[2022-02-12 08:56:37,943] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '58.75 %'}
[2022-02-12 08:57:00,395] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '74.37 %'}
[2022-02-12 08:57:09,428] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '64.77 %'}
[2022-02-12 08:57:14,569] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '80.91 %'}
[2022-02-12 08:57:24,755] INFO    : Objeto Contado {'TIPO': 'truck',
     'LÍNEA DE CONTEO': 'C', 'CONFIANZA': '62.84 %'}
[2022-02-12 08:57:24,756] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '73.14 %'}
[2022-02-12 08:57:47,164] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '89.20 %'}
[2022-02-12 08:57:53,442] INFO    : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'C', 'CONFIANZA': '85.43 %'}
```

```
[2022-02-12 08:59:21,020] INFO     : Objeto Contado {'TIPO': '
    motorcycle', 'LÍNEA DE CONTEO': 'C', 'CONFIANZA': '94.49 %'}
[2022-02-12 09:00:54,644] INFO     : Objeto Contado {'TIPO': '
    motorcycle', 'LÍNEA DE CONTEO': '2', 'CONFIANZA': '82.38 %'}
...
...
...
...

[2022-02-12 12:57:54,038] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '63.95 %'}
[2022-02-12 12:57:54,039] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '63.95 %'}
[2022-02-12 12:58:01,161] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '1', 'CONFIANZA': '80.46 %'}
[2022-02-12 12:58:17,194] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '79.49 %'}
[2022-02-12 12:58:17,195] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '3', 'CONFIANZA': '79.49 %'}
[2022-02-12 12:58:18,675] INFO     : Objeto Contado {'TIPO': 'truck',
     'LÍNEA DE CONTEO': '1', 'CONFIANZA': '66.37 %'}
[2022-02-12 12:58:24,004] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '1', 'CONFIANZA': '67.34 %'}
[2022-02-12 12:58:40,412] INFO     : Objeto Contado {'TIPO': '
    motorcycle', 'LÍNEA DE CONTEO': '2', 'CONFIANZA': '73.08 %'}
[2022-02-12 12:58:46,302] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '1', 'CONFIANZA': '88.53 %'}
...
...
...
...

[2022-02-12 15:40:59,693] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '96.26 %'}
[2022-02-12 15:41:49,063] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '80.00 %'}
[2022-02-12 15:41:54,673] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '94.23 %'}
[2022-02-12 15:42:07,372] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '78.74 %'}
[2022-02-12 15:42:39,158] INFO     : Objeto Contado {'TIPO': 'truck',
     'LÍNEA DE CONTEO': '2', 'CONFIANZA': '60.81 %'}
[2022-02-12 15:42:51,757] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '88.27 %'}
[2022-02-12 15:43:11,817] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '90.82 %'}
[2022-02-12 15:43:43,376] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '93.80 %'}
```

```
[2022-02-12 15:44:09,207] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '88.17 %'}
[2022-02-12 15:44:23,541] INFO     : Objeto Contado {'TIPO': 'truck',
     'LÍNEA DE CONTEO': '2', 'CONFIANZA': '51.91 %'}
[2022-02-12 15:44:53,113] INFO     : Objeto Contado {'TIPO': 'bus', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '65.73 %'}
[2022-02-12 15:44:53,599] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '97.78 %'}
[2022-02-12 15:44:54,815] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': '2', 'CONFIANZA': '95.74 %'}
[2022-02-12 15:45:20,678] INFO     : Objeto Contado {'TIPO': 'car', '
    LÍNEA DE CONTEO': 'B', 'CONFIANZA': '85.57 %'}
[2022-02-12 15:45:26,739] INFO     : Objeto Contado {'TIPO': 'truck',
     'LÍNEA DE CONTEO': '2', 'CONFIANZA': '51.83 %'}
[2022-02-12 15:47:25,556] INFO     : === ¡PROCESO FINALIZADO! === {}
[2022-02-12 15:47:25,557] INFO     : ========= RESULTADOS =========
{'CONTEO FINAL': {
'1': {'car': 446, 'bus': 16, 'truck': 80, 'motorcycle': 43},
'2': {'car': 566, 'bus': 54, 'truck': 98, 'motorcycle': 34},
'B': {'car': 55, 'bus': 9, 'truck': 13, 'motorcycle': 2},
'3': {'car': 261, 'bus': 41, 'truck': 26, 'motorcycle': 11},
'C': {'car': 38, 'truck': 9, 'motorcycle': 9}}}
=== TIME ===: El proceso duró 24804.4948 segundos
```

El proceso inició a las 08:54:01 horas (AM) y finalizó a la 15:47:25 horas (PM), resultando una duración total de 6 horas 53 minutos 24 segundos (24,804.495 segundos). El análisis fue realizado en el video de 27 minutos 41 segundos, por lo que, por cada un (1) minuto de video analizado el algoritmo tardó aproximadamente 14 minutos con 56 segundos en procesarlo, o también, por cada un (1) segundo de video, tardó 14.9 segundos procesarlo.

TABLA N° 4.8: Tiempo de procesamiento del algoritmo utilizando la red neuronal Detectron2.
Fuente: Propia

Red Neuronal	Tiempo Video Real	Tiempo Procesado
Dectectron2	27 min 41 seg	6 h 53 min 24 seg
	1 min	14 min 56 seg
	1 seg	14.9 seg

De la Tabla N° 4.8 se concluye que utilizando la red neuronal *«Detectron2»* el tiempo de procesamiento equivale aproximadamente a quince (15) veces la duración del video que se analizará. Por ejemplo, un video de una (1) hora se procesará en quince (15) horas y uno tres (3) horas, en cuarenta y cinco (45) horas.

Al finalizar el proceso, se contabilizó un total de 1,811 vehículos, los cuales se encuentran distribuidos entre las cinco (5) líneas de conteo. La línea de conteo *'1'* identificó un total de 585 vehículos, que convertidos a Unidades Coche Patrón (UPC) son 708 vehículos. La línea *'2'*, un total de 752 vehículos, que en UCP son 984 vehículos. La línea *'B'*, un total de 79 vehículos, que en UCP son 115 vehículos. La línea *'3'*, un total de 339 vehículos, que en UCP son 453 vehículos. Y la línea *'C'*, un total de 56 vehículos, que en UCP son 63 vehículos. Para convertir los vehículos a UCP se utilizaron los factores de conversión especificados en la Tabla N° 4.1.

De acuerdo al reporte final, se detalla a continuación los vehículos contabilizados por cada línea de conteo y según su tipología, para la red neuronal *«Detectron2»*:

TABLA N° 4.9: Conteo total de vehículos utilizando la red neuronal Detectron2. Fuente: Propia

Líneas de Conteo	Tipo de vehículo				Total de vehículos	Vehículos UPC
	carros	buses	camión	motoc.		
1	446	16	80	43	585	708
2	566	54	98	34	752	984
B	55	9	13	2	79	115
3	261	41	26	11	339	453
C	38	0	9	9	56	63
TOTAL	1,366	120	226	99	1,811	2,324

Nota: En la siguiente sección (Sección 4.1.7) se comparan los resultados obtenidos (por el algoritmo) con un aforo manual obtenido directo del video y se determina la precisión (aciertos) de los conteos para cada red utilizada (Sección 5.1).

Para visualizar el video final obtenido del proceso de detección, clasificación y conteo vehicular utilizando la red neuronal *«Dectectron2»*, ingrese al siguiente enlace: https://https://t.ly/Ldf6 o escanee el código QR de la Figura N° 4.12.

FIGURA N° 4.12: Código QR para visualizar el video procesado con la red Detectron2

4.1.7 Aforo vehicular

Luego de comprender el proceso y funcionamiento del algoritmo de detección y clasificación de vehículos, procedemos a realizar un aforo vehicular en dicha intersección para determinados intervalos de tiempo, para ello, utilizaremos el algoritmo de detección y clasificación vehicular (con las redes neuronales) y compararemos sus resultados con un aforo manual. El aforo se realizará en intervalos de 5 minutos (se toma el periodo de 5 min por cuestiones prácticas ya que el video tiene una duración de 27 min), realizamos el corte del video en videos cortos de 5 minutos cada uno y lo procesamos en el algoritmo a fin de determinar los volumen de tráfico que soporta la intersección en esos intervalos de tiempo. Luego comparamos sus resultados con un aforo manual, esto con el objetivo de determinar la precisión que tiene cada red neuronal para detectar y clasificar vehículos.

El aforo vehicular se realizará utilizando las redes neuronales *«YoloV3»* y *«Detectron2»*. En la Sección 4.1.6 se utilizaron estas redes para calcular conteos vehiculares totales en la intersección, a diferencia de esta sección en la que se determinarán aforos vehiculares cada 5 min para dicha intersección.

El aforo vehicular en la intersección permitirá conocer la cantidad de vehículos que ingresan a ella, según su origen, y la dirección a la que se dirigen. Para el aforo se utilizaron las mismas líneas de conteo (*LC*) y *DROI* del ejemplo previo (ver Sección 4.1.6), sus coordenadas se encuentran en la Tabla N° 4.2 y Tabla N° 4.4.

A continuación, se detallan los flujos vehiculares de la intersección y cuales son las líneas de conteo que contabilizan su volumen de tráfico (ver Figura N° 5.1):

$$Q_{11} = LC'1' \tag{4.1}$$

$$Q_{21} = LC'2' \tag{4.2}$$

$$Q_{22} = LC'B' \tag{4.3}$$

$$Q_{31} = LC'C' \tag{4.4}$$

$$Q_{32} = LC'3' - LC'C' \tag{4.5}$$

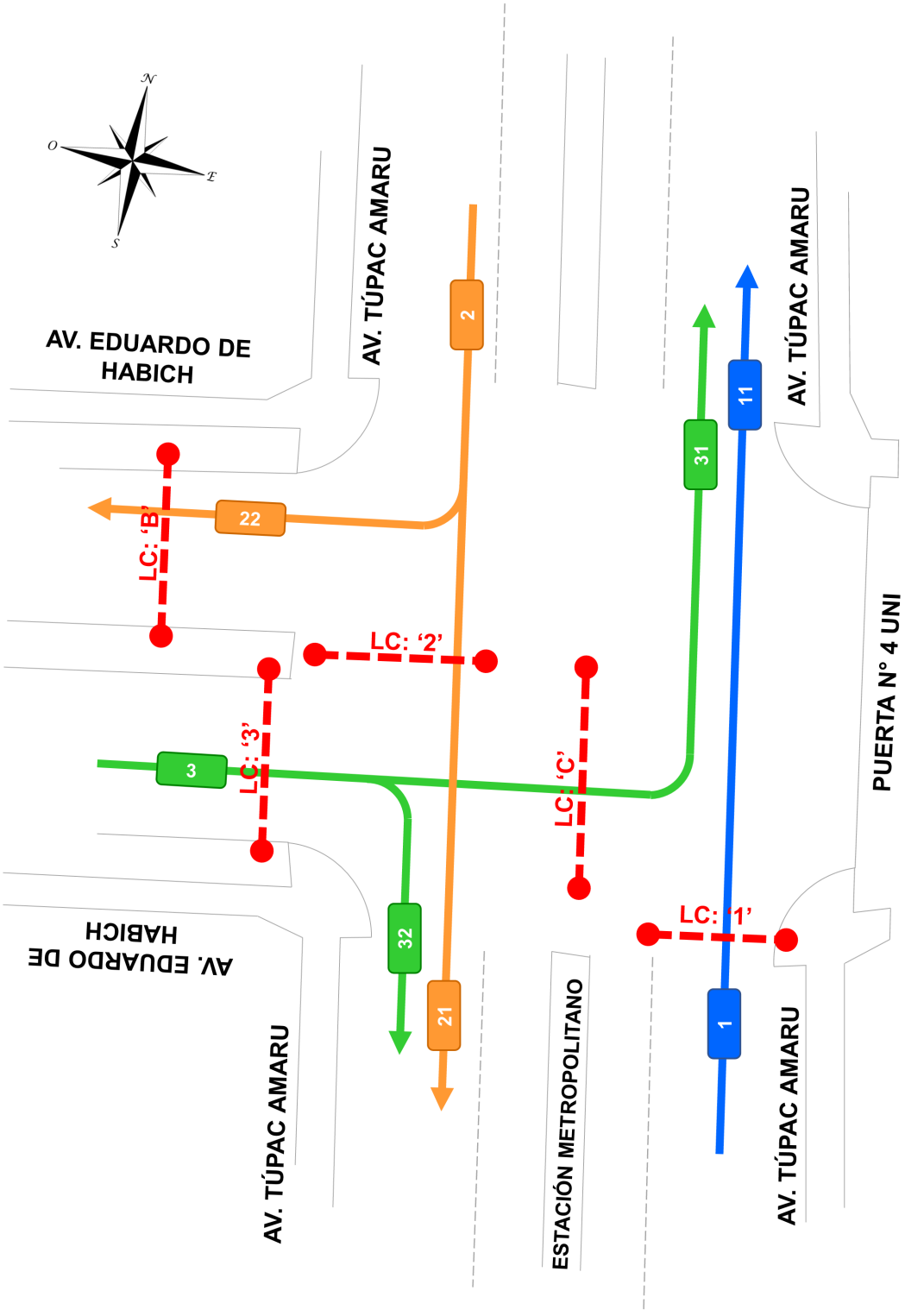


FIGURA N° 4.13: Aforo Vehicular: Intersección Av. Túpac Amaru - Av. Eduardo de Habich. Fuente: Propia

4.1.7.1 Aforo Vehicular con Yolo

Procedemos a evaluar los 5 cortes de video (de 5min cada uno) en el algoritmo de detección y clasificación vehicular haciendo uso de la red *«Yolo»*. A continuación, se detalla la data obtenida del aforo vehicular para cada flujo de vehículos:

TABLA N° 4.10: Aforo Vehicular del flujo Q11 - YoloV3.

Hora	Q11 - RNA: Yolo				Total vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	104	3	19	5	131	162
01:05 - 01:10	99	6	7	5	117	136
01:10 - 01:15	84	4	11	4	103	125
01:15 - 01:20	108	4	9	3	124	143
01:20 - 01:25	100	4	11	1	116	140
TOTAL	**495**	**21**	**57**	**18**	**591**	**706**
	83.76 %	3.55 %	9.64 %	3.05 %		

TABLA N° 4.11: Aforo Vehicular del flujo Q21 - YoloV3.

Hora	Q21 - RNA: Yolo				Total vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	113	11	17	4	145	190
01:05 - 01:10	116	11	18	2	147	195
01:10 - 01:15	110	10	14	6	140	177
01:15 - 01:20	88	11	9	7	115	146
01:20 - 01:25	130	12	14	4	160	202
TOTAL	**557**	**55**	**72**	**23**	**707**	**910**
	78.78 %	7.78 %	10.18 %	3.25 %		

TABLA N° 4.12: Aforo Vehicular del flujo Q22 - YoloV3.

Hora	Q22 - RNA: Yolo				Total vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	8	0	4	0	12	18
01:05 - 01:10	12	1	6	0	19	30
01:10 - 01:15	8	0	4	0	12	18
01:15 - 01:20	14	0	2	0	16	19
01:20 - 01:25	6	0	2	0	8	11
TOTAL	**48**	**1**	**18**	**0**	**67**	**96**
	71.64 %	1.49 %	26.87 %	0.00 %		

TABLA N° 4.13: Aforo Vehicular del flujo Q31 - YoloV3.

Hora	Q31 - RNA: Yolo				Total vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	6	0	0	1	7	6
01:05 - 01:10	12	0	0	1	13	12
01:10 - 01:15	7	0	0	1	8	7
01:15 - 01:20	10	0	0	1	11	10
01:20 - 01:25	7	0	0	3	10	8
TOTAL	**42**	**0**	**0**	**7**	**49**	**44**
	85.71 %	0.00 %	0.00 %	14.29 %		

TABLA N° 4.14: Aforo Vehicular del flujo Q32 - YoloV3.

Hora	Q32 - RNA: Yolo				Total vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	38	6	7	0	51	74
01:05 - 01:10	28	4	7	2	41	58
01:10 - 01:15	19	3	3	1	26	36
01:15 - 01:20	45	6	5	0	56	76
01:20 - 01:25	13	3	6	0	22	37
TOTAL	**143**	**22**	**28**	**3**	**196**	**280**
	72.96 %	11.22 %	14.29 %	1.53 %		

De la Tabla N° 4.10, se observa que el algoritmo contabilizó un total de 591 veh. (706 veh. UCP) en la Av. Túpac Amaru, en dirección de sur a norte. Estos vehículos provenientes de los distritos de Lima (centro), El Agustino y San Lurigancho se dirigían hacia el norte de la ciudad, hacia los distritos de Independencia, Los Olivos y Comas. La configuración del flujo está dada por: 495 carros (83.76 %), 21 buses (3.55 %), 57 camiones (9.64 %) y 18 motocicletas (3.05 %).

Asimismo, de la Tabla N° 4.11 y 4.12, se contabilizó un total de 774 veh. ($Q_{21}: 707$ + $Q_{22}: 67$) en la Av. Túpac Amaru, provenientes de la zona norte de la ciudad. Los distritos origen de estos vehículos fueron Independencia, Los Olivos y Comas. El flujo se dividió en: i) 707 veh. (91.3 %) continuaron de frente por la Av. Túpac Amaru (Tabla N° 4.11), en dirección a la zona sur-centro de la ciudad, hacia los distritos de Lima (centro), El Agustino y San Lurigancho, y ii) 67 veh. (8.7 %) giraron a la derecha a la Av. Eduardo de Habich (Tabla N° 4.12), hacia los distritos de San Martín de Porres, Cercado y Callao.

Finalmente, de la Tabla N° 4.13 y 4.14, se contabilizó un total de 245 veh. ($Q_{31}: 49$ + $Q_{32}: 196$) provenientes de la zona oeste de la ciudad, de los distritos de San Martín de Porres, Cercado y Callao. De estos vehículos, el 20 % (49 veh.) (Tabla N° 4.13) giraron a la izquierda tomando como ruta destino el norte de la ciudad, hacia los distritos de Independencia, Los Olivos y Comas; el otro 80 % (196 veh.)(Tabla N° 4.14) giraron a la derecha, hacia el sur-centro de la ciudad de Lima.

TABLA N° 4.15: Aforo Total Vehicular en la Intersección utilizando la red neuronal YoloV3.

Hora	Volúmenes de Tránsito (UCP)					Total vehículos	%
	Q 11	Q 21	Q 22	Q 31	Q 32		
01:00 - 01:05	162	190	18	6	74	450	22.1 %
01:05 - 01:10	136	195	30	13	58	432	21.2 %
01:10 - 01:15	125	177	18	7	36	363	17.8 %
01:15 - 01:20	143	146	19	10	76	394	19.3 %
01:20 - 01:25	140	202	11	8	37	398	19.6 %
Q total (25 min)	**706**	**910**	**96**	**44**	**281**	**2,037**	**100 %**
q max (5min)	162	202	30	13	76	450	
Q max (25min)	**810**	**1,010**	**150**	**65**	**380**	**2,250**	
FHMD	0.87	0.90	0.64	0.68	0.74	0.91	

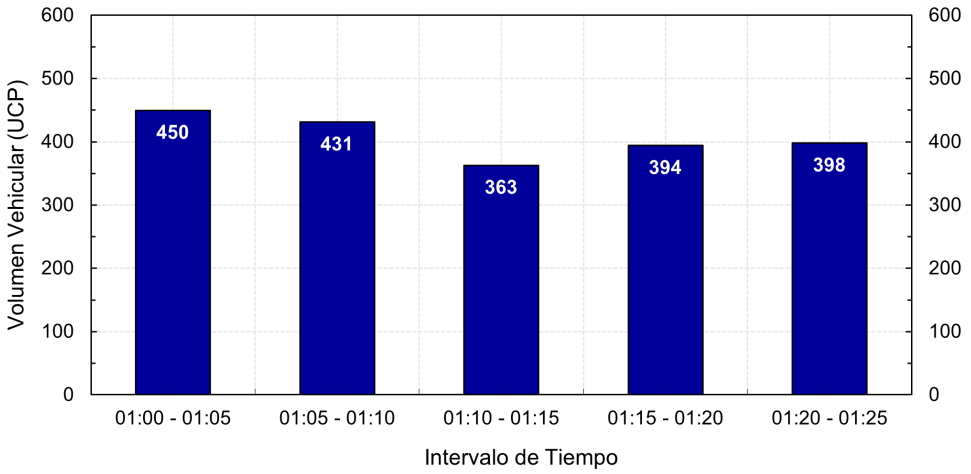


FIGURA N° 4.14: Volumen Vehicular en la Intersección por periodos de 5 min, obtenidos del algoritmo y utilizando la Red Neuronal YoloV3. Fuente: Propia

Como se detalla en la Tabla N° 4.15, el volumen de tránsito en la intersección fue de 2,037 vehículos UCP, de los cuales: 706 veh. (Q_{11}) provinieron de los distritos de Lima (centro), El Agustino y San Lurigancho, 910 veh. (Q_{21}) + 96 veh. (Q_{22}) de los distritos de Independencia, Los Olivos y Comas, y 44 veh. (Q_{31}) + 281 veh. (Q_{32}) de los distritos de San Martín de Porres, Cercado y Callao.

En el horario evaluado, la Av. Túpac Amaru tiene un Factor Horario de Máxima Demanda (FHMD) de 0.87 (en dirección sur-norte) y 0.90 (de norte-sur), indicando que la generación de viajes se sostiene a lo largo del periodo evaluado, es decir que los viajes en ambos sentidos son constantes y estables. Los vehículos que provienen de la Av. Eduardo de Habich y se dirigen al norte de la ciudad (Q_{31}) tiene un FHMD de 0.68 y los que giran hacia el sur-centro, un FHMD de 0.74, esto se traduce a flujos inestables para estas vías.

De la Figura 4.14, se observa que, en el periodo evaluado, el máximo flujo vehicular que sostuvo la vía fue de 450 $veh./5min$, y el mínimo de 363 $veh./5min$.

4.1.7.2 Aforo Vehicular con Detectron

Del mismo modo, procedemos a evaluar los 5 cortes de video, pero ahora utilizando la red neuronal *«Detectron»*, para ello, seguir los pasos detallados en el item 4.1.6.2 de la presente tesis. A continuación, se detalla la data obtenida del aforo vehicular para cada flujo de vehículos con la red *«Detectron»*:

TABLA N° 4.16: Aforo Vehicular del flujo Q11 - Detectron2.

Hora	Q11 - RNA: Detectron				Total vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	92	4	21	6	123	158
01:05 - 01:10	89	4	8	11	112	125
01:10 - 01:15	73	4	19	11	107	136
01:15 - 01:20	94	3	13	7	117	138
01:20 - 01:25	90	1	18	8	117	141
TOTAL	**438**	**16**	**79**	**43**	**576**	**698**
	76.04 %	2.78 %	13.72 %	7.47 %		

TABLA N° 4.17: Aforo Vehicular del flujo Q21 - Detectron2.

Hora	Q21 - RNA: Detectron				Total vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	115	11	22	6	154	205
01:05 - 01:10	113	10	23	0	146	201
01:10 - 01:15	103	6	22	6	137	178
01:15 - 01:20	84	11	11	14	120	149
01:20 - 01:25	124	15	16	8	163	212
TOTAL	**539**	**53**	**94**	**34**	**720**	**944**
	74.86 %	7.36 %	13.06 %	4.72 %		

TABLA N° 4.18: Aforo Vehicular del flujo Q22 - Detectron2.

Hora	Q22 - RNA: Detectron				Total de vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	8	1	5	0	14	24
01:05 - 01:10	11	4	2	0	17	28
01:10 - 01:15	7	2	2	0	11	18
01:15 - 01:20	17	0	2	0	19	22
01:20 - 01:25	10	2	2	2	16	22
TOTAL	**53**	**9**	**13**	**2**	**77**	**113**
	68.83 %	11.69 %	16.88 %	2.60 %		

TABLA N° 4.19: Aforo Vehicular del flujo Q31 - Detectron2.

Hora	Q31 - RNA: Detectron				Total de vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	5	0	3	1	9	13
01:05 - 01:10	11	0	2	2	15	17
01:10 - 01:15	7	0	0	3	10	8
01:15 - 01:20	7	0	3	1	11	15
01:20 - 01:25	6	0	1	2	9	9
TOTAL	**36**	**0**	**9**	**9**	**54**	**61**
	66.67 %	0.00 %	16.67 %	16.67 %		

TABLA N° 4.20: Aforo Vehicular del flujo Q32 - Detectron2.

Hora	Q32 - RNA: Detectron				Total de vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	53	12	5	0	70	102
01:05 - 01:10	51	7	6	1	65	87
01:10 - 01:15	36	4	2	0	42	53
01:15 - 01:20	61	7	1	1	70	85
01:20 - 01:25	24	11	3	0	38	65
TOTAL	**225**	**41**	**17**	**2**	**285**	**391**
	78.95 %	14.39 %	5.96 %	0.70 %		

De la Tabla N° 4.16, se contabilizó un total de 576 veh. (698 veh. UCP) en la Av. Túpac Amaru, en dirección de sur a norte. La configuración del flujo está dada por: 438 carros (76.04 %), 16 buses (2.78 %), 79 camiones (13.72 %) y 43 motocicletas (7.47 %).

Por otra lado, en la dirección norte-sur, se contabilizó un total de 797 veh. (1,057 veh. UCP) que provinieron de los distritos de Independencia, Los Olivos y Comas; de los cuales 720 veh. (90.34 %) (Tabla N° 4.17) continuaron por la Av. Túpac Amaru, cuyo flujo corresponde a: 539 carros (74.86 %), 53 buses (13.06 %), 94 camiones (13.06 %) y 34 motocicletas (4.72 %); y 77 veh. (9.66 %) (Tablas N° 4.18) giraron hacia la Av. Eduardo de Habich, con composición de: 53 carros (68.83 %), 9 buses (11.69 %), 13 camiones (16.88 %) y 2 motocicletas (2.6 %).

Finalmente, 339 veh. (452 veh. UCP) provinieron de la Av. Eduardo de Habich, de los distritos de San Martín de Porres, Cercado y Callao; de ellos, 54 veh. (15.93 %) (Tabla N° 4.19) giraron a la izquierda hacia el norte de la ciudad, hacia los distritos de Independencia, Los Olivos y Comas, y 285 veh. (84.07 %) giraron a la derecha hacia el sur-centro de Lima (Tabla N° 4.20).

TABLA N° 4.21: Aforo Total Vehicular en la Intersección utilizando la red neuronal Detectron2.

Hora	Volúmenes de Tránsito (UCP)					Total vehículos
	Q 11	Q 21	Q 22	Q 31	Q 32	
01:00 - 01:05	158	205	24	13	102	501
01:05 - 01:10	125	201	28	17	87	457
01:10 - 01:15	136	178	18	8	53	393
01:15 - 01:20	138	149	22	15	85	409
01:20 - 01:25	141	212	22	9	65	448
Q total (25 min)	**698**	**944**	**113**	**61**	**391**	**2208**
q max (5min)	158	212	28	17	102	501
Q max (25min)	**792**	**1058**	**140**	**83**	**508**	**2506**
FHMD	0.88	0.89	0.81	0.74	0.77	0.88

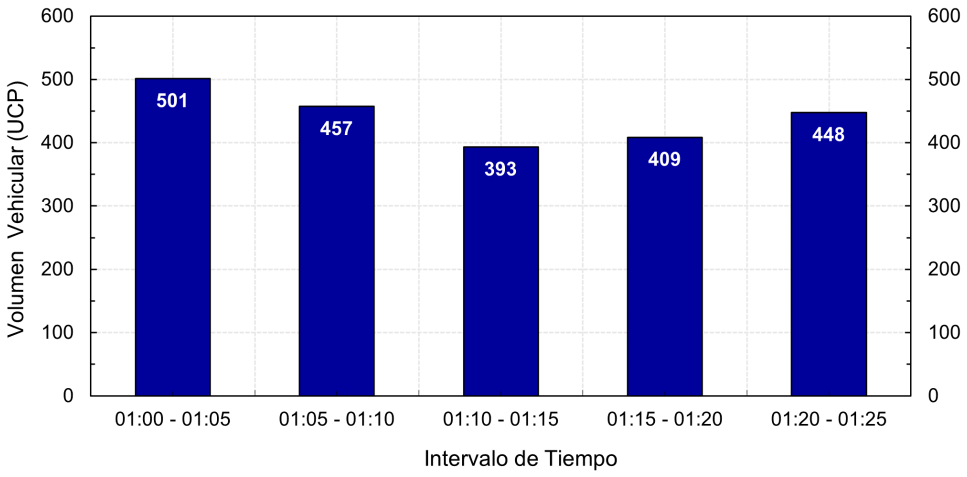


FIGURA N° 4.15: Volumen Vehicular en la Intersección por periodos de 5 min, obtenidos del algoritmo y utilizando la Red Neuronal Detectron2. Fuente: Propia

El volumen total de tránsito en la intersección fue de 2,208 vehículos UCP (Tabla N° 4.21), de los cuales: 698 veh. (Q_{11}) provinieron de los distritos de Lima (centro), El Agustino y San Lurigancho, 944 veh. (Q_{21}) + 113 veh. (Q_{22}) de los distritos de Independencia, Los Olivos y Comas, y 61 veh. (Q_{31}) + 391 veh. (Q_{32}) de los distritos de San Martín de Porres, Cercado y Callao.

En el horario evaluado, la Av. Túpac Amaru tiene un FHMD de 0.88 (en dirección sur-norte) y 0.89 (de norte-sur), indicando que la generación de viajes se sostiene a lo largo del periodo evaluado, es decir, es uniforme y homogéneo. Los vehículos que provienen de la Av. Eduardo de Habich y se dirigen al norte de la ciudad (Q_{31}) tiene un FHMD de 0.74 y los que giran hacia el sur-centro, un FHMD de 0.77, esto se traduce a flujos variables para estas vías. De la Figura 4.15, se observa que, en el periodo evaluado, el máximo flujo vehicular que sostuvo la vía fue de 501 $veh./5min$, y el mínimo de 393 $veh./5min$.

4.1.7.3 Aforo Vehicular Manual

Finalmente, realizaremos un aforo *«manual»* con la finalidad de contrastar los resultados obtenidos de las redes neuronales con los datos de conteo reales.

TABLA N° 4.22: Aforo Vehicular del flujo Q11 - Manual.

Hora	Q11 - Manual				Total de vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	115	7	11	7	140	166
01:05 - 01:10	101	5	9	11	126	142
01:10 - 01:15	94	5	10	11	120	138
01:15 - 01:20	103	5	7	8	123	138
01:20 - 01:25	111	6	8	8	133	152
TOTAL	**524**	**28**	**45**	**45**	**642**	**735**
	90.97 %	4.86 %	7.81 %	7.81 %		

TABLA N° 4.23: Aforo Vehicular del flujo Q21 - Manual.

Hora	Q21 - Manual				Total de vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	114	8	20	4	146	189
01:05 - 01:10	122	5	16	1	144	177
01:10 - 01:15	115	5	14	4	138	166
01:15 - 01:20	85	7	8	16	116	131
01:20 - 01:25	128	8	15	8	159	192
TOTAL	**564**	**33**	**73**	**33**	**703**	**856**
	78.33 %	4.58 %	10.14 %	4.58 %		

TABLA N° 4.24: Aforo Vehicular del flujo Q22 - Manual.

Hora	Q22 - Manual				Total de vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	9	0	7	2	18	27
01:05 - 01:10	8	1	5	3	17	24
01:10 - 01:15	10	1	5	1	17	26
01:15 - 01:20	16	0	6	3	25	32
01:20 - 01:25	7	1	3	3	14	18
TOTAL	**50**	**3**	**26**	**12**	**91**	**128**
	64.94 %	3.90 %	33.77 %	15.58 %		

TABLA N° 4.25: Aforo Vehicular del flujo Q31 - Manual.

Hora	Q31 - Manual				Total de vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	6	0	0	1	7	6
01:05 - 01:10	11	0	0	1	12	11
01:10 - 01:15	7	0	0	3	10	8
01:15 - 01:20	10	0	0	1	11	10
01:20 - 01:25	7	0	0	2	9	8
TOTAL	**41**	**0**	**0**	**8**	**49**	**44**
	75.93 %	0.00 %	0.00 %	14.81 %		

TABLA N° 4.26: Aforo Vehicular del flujo Q32 - Manual.

Hora	Q32 - Manual				Total de vehículos	Vehículos UCP
	carros	buses	camión	motoc		
01:00 - 01:05	44	3	3	1	51	61
01:05 - 01:10	46	0	9	3	58	69
01:10 - 01:15	31	1	1	1	34	37
01:15 - 01:20	57	2	1	2	62	66
01:20 - 01:25	20	3	4	1	28	39
TOTAL	**198**	**9**	**18**	**8**	**233**	**273**
	69.47 %	3.16 %	6.32 %	2.81 %		

TABLA N° 4.27: Aforo Total Vehicular en la Intersección utilizando el método manual.

Hora	Volúmenes de Tránsito (UCP)					Total vehículos
	Q 11	Q 21	Q 22	Q 31	Q 32	
01:00 - 01:05	166	189	27	6	61	449
01:05 - 01:10	142	177	24	11	69	425
01:10 - 01:15	138	166	26	8	37	375
01:15 - 01:20	138	131	32	10	66	378
01:20 - 01:25	152	192	18	8	39	409
Q total (25 min)	**735**	**856**	**128**	**44**	**273**	**2036**
q max (5min)	166	192	32	11	69	449
Q max (25min)	**829**	**961**	**160**	**57**	**347**	**2247**
FHMD	0.89	0.89	0.80	0.77	0.78	0.91

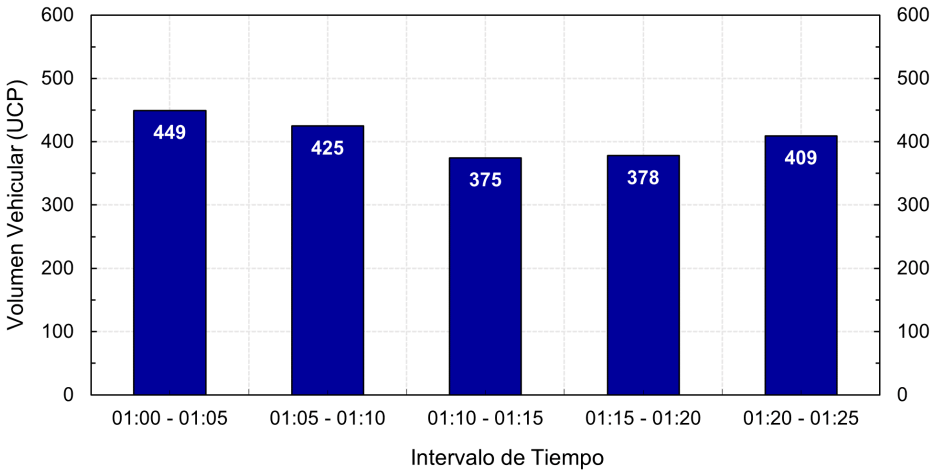


FIGURA N° 4.16: Volumen Vehicular en la Intersección por periodos de 5 min, obtenidos del aforo vehicular utilizando el método manual. Fuente: Propia

En el periodo evaluado, el volumen total de tránsito real en la intersección fue de 2,036 vehículos UCP (Tabla N° 4.27), de los cuales: 735 veh. (Q_{11}) provinieron de los distritos de Lima (centro), El Agustino y San Lurigancho, 856 veh. (Q_{21}) + 128 veh. (Q_{22}) de los distritos de Independencia, Los Olivos y Comas, y 44 veh. (Q_{31}) + 273 veh. (Q_{32}) de los distritos de San Martín de Porres, Cercado y Callao.

La Av. Túpac Amaru tiene un FHMD de 0.89 en ambos sentidos, indicando que la generación de viajes se sostiene a lo largo del periodo evaluado, es decir, es constante y uniforme. Los vehículos que provienen de la Av. Eduardo de Habich y giran hacia el norte de la ciudad (Q_{31}) tienen un FHMD de 0.74 y los que giran hacia el sur, un FHMD de 0.77, esto se traduce a flujos variables. De la Figura 4.15, se observa que el máximo flujo vehicular que sostuvo la intersección fue de 449 $veh./5min$, y el mínimo de 375 $veh./5min$.

CAPÍTULO V: ANÁLISIS Y DISCUSIÓN DE RESULTADOS

Finalmente, luego de obtener el aforo vehicular en la intersección de la vía utilizando los tres métodos (*«Yolo»* (RNA), *«Detectron»* (RNA) y *«Manual»*), procedemos a análizar y comparar los resultados conseguidos y a determinar la precisión que tienen las redes neuronales para detectar y contabilizar vehículos.

5.1 ANÁLISIS DE RESULTADOS

5.1.1 Comparativa de Aforos

A continuación, se muestra la Tabla N° 5.1 en la que se presenta la comparativa de resultados obtenidos por los tres métodos de aforo vehicular en la intersección. Los volúmenes de tránsito son representados en Unidades Coche Patrón (UCP).

Asimismo, se determinará la precisión que han obtenido las redes neuronales para identificar, detectar y contabilizar vehículos, para ello, se tendrá como línea base los datos del aforo manual, que serían los datos reales del tráfico en la vía.

La precisión se calculará de la siguiente manera:

$$Precision = \left[1 - \frac{|RNA - Manual|}{Manual}\right] * 100\,\% \tag{5.1}$$

TABLA N° 5.1: Comparativa de Aforo Total Vehicular en la Intersección. Fuente: Propia

Hora	Volúmenes de Tránsito (UCP)			Diferencia de Conteo		Precisión	
	Manual	YoloV3	Detetr2	Yolo.	Detetr2	Yolo.	Detetr2
01:00 - 01:05	449	450	501	1	52	99.8 %	88.4 %
01:05 - 01:10	425	431	457	6	32	98.6 %	92.5 %
01:10 - 01:15	375	363	393	-12	18	96.8 %	95.2 %
01:15 - 01:20	378	394	409	16	31	95.8 %	91.8 %
01:20 - 01:25	409	398	448	-11	39	97.3 %	90.5 %
TOTAL	**2036**	**2036**	**2208**	**0**	**172**	**97.6 %**	**91.7 %**

De la Tabla N° 5.1, se aprecia que en el aforo *manual* se obtuvo un total de 2,036 veh. UPC, cantidad similar obtenida utilizando la red neuronal *YoloV3* (2,036 veh), mientras que para la red *Detectron2* se obtuvo un total de 2,208 veh. (diferencia de 172 veh. con el aforo manual).

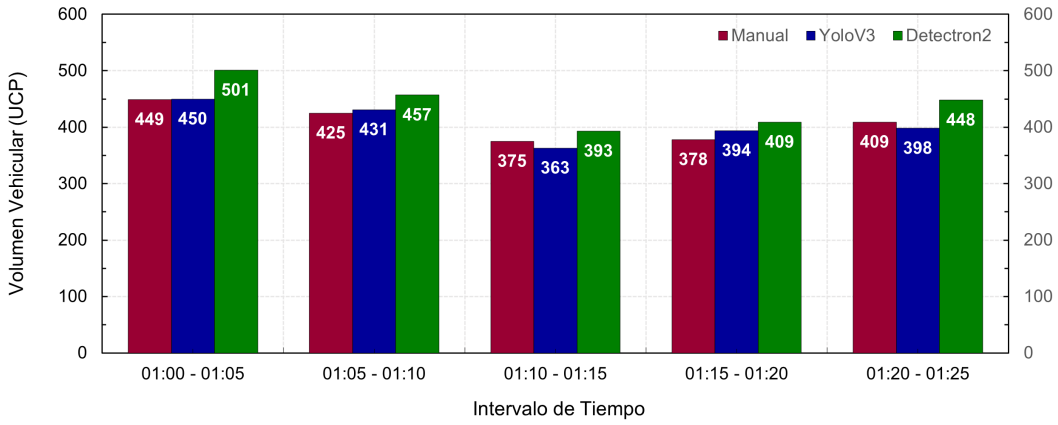


FIGURA N° 5.1: Comparativa de Aforo Total Vehicular en la Intersección. Fuente: Propia

5.1.2 Diferencias de conteo

La diferencia de conteos entre el aforo manual y Yolo fue de: {1, 6, -12, 16, -11} veh., respectivamente para cada periodo. En algunos periodos se contó vehículos menos y en otros vehículos más, y en algunos casos el algoritmo no acertó en la tipológia del vehículo. No obstante, la suma absoluta de estos resultados fue de 0, es decir, existió una compensación de conteo a lo largo del periodo evaluado.

Respecto a los resultados del aforo manual y Detectron, la diferencia de conteo fue de: {52, 32, 18, 31, 39} veh., esto quiere decir que el algoritmo contabilizó más vehículos de los que pasaron por la vía, esto se explica debido a que el tracking (seguimiento) de los vehículos, en muchos casos, perdía el rastro de un vehículo y cuando lo volvía a identificar lo consideraba como otro y no como el mismo vehículo, contabilizandolo más de una vez. La suma absoluta de estos resultados fue de 172 veh. contados de más.

5.1.3 Precisión de las Redes Neuronales

La precisión de las redes neuronales artificiales (RNA) se calculó con la formula N° 5.1. Para el caso de la red Yolo, su precisión por cada periodo fue de: 99.8 %, 98.6 %, 96.8 %, 95.8 %, 97.3 %, respectivamente, y en promedio de 97.6 %. Respecto a la red Detectron, su precisión en cada periodo fue de: 88.4 %, 92.5 %, 95.2 %, 91.8 %, 90.5 %, y en promedio del 91.7 %, por debajo de los resultados de la red Yolo.

En un caso práctico, se espera que utilizando la red Yolo para un aforo vehicular, los resultados obtenidos sean 97.6 % correctos, y para la red Detectron, 91.7 % correctos.

5.1.4 Tiempo de procesamiento de las Redes Neuronales

Es necesario traer a este punto y recodar los tiempos de procesamiento que les toman a cada red realizar el análisis y evaluación de un video. Los detalles de esta información fueron especificados y detallados en la Sección 4.1.6.1 y 4.1.6.2 de la presente tesis.

De la Tabla N° 4.6 *"Tiempo de procesamiento utilizando la red YoloV3"*, del Capitulo IV), se concluye que para la red Yolo, el tiempo de procesamiento equivale aproximadamente a seis (6) veces la duración del video analizado, es decir, por un video 1 h., el algoritmo dará sus resultados en 6h. Por otra parte, para la red Detectron, y de la Tabla N° 4.8 *"Tiempo de procesamiento utilizando la red Detectron2"*, del Capitulo IV), se concluye que utilizando la red Detectron el tiempo de procesamiento equivale aproximadamente a quince (15) veces la duración del video, es decir, por un video 1 h., el algoritmo dará sus resultados en 15h,

TABLA N° 5.2: Comparativa del tiempo de procesamiento con YoloV3 y Detectron2

Red Neuronal	Tiempo Video Real	Tiempo Procesado
$YoloV3$	1h	6h 09min
$Detectron2$	1h	14h 56min

El tiempo de procesamiento de la red Detectron demora aproximadamente 2.5 veces más de lo que le toma a la red Yolo. Por lo que, es correcto concluir que, la red *Yolov3* es más ligera que *Dectetron2* si se hablase de tiempos de procesamiento.

5.2 DISCUSIÓN FINAL

- El uso de la Inteligencia Artificial, a través de las redes neuronales artificiales, es una alternativa accesible, rentable y económica para realizar un aforo vehicular de una vía y/o intersección. La presente tesis ha logrado implementar y aplicar técnicas de IA en la automatización del proceso de detección y clasificación vehicular a través del área de visión computacional.

- Tomando en consideración la precisión y tiempo de procesamiento de cada red neuronal que se ha analizado, se concluye que para el análisis en videos la red *Yolov3* es mucho más efectiva y precisa en calidad de resultados y reducción de tiempo de procesamiento en comparación a la red *Detectron2*. Los resultados de Yolo son muy confiables al realizar un aforo vehicular. Tal y como se muestra de la Tabla N° 5.1, la precisión de resultados de la red Yolo fue del 97.6 % en promedio y 91.7 % para la red Dectetron2.

- Los costos operativos son bajos en comparación a un aforo tradicional, en la que se traslada personal humano al lugar de estudio y se les mantiene por largas horas de trabajo recopilando data del flujo vehicular. Esta cantidad de personas es multiplicada por la cantidad de flujos, tipología de vehículos, intersecciones y vías que se estudiarán; resultando costos elevados en comparativa al procesamiento desarrollado en la presente tesis, en la que es más que suficiente realizar una grabación del lugar en estudio, tener un equipo de computo y tiempo para procesar el algoritmo para finalmente obtener la data deseada y con resultados verídicos y confiables.

- Los resultados obtenidos del aforo vehicular es un input necesario para elaborar estudios de tráfico, pronosticar la demanda futura de una vía, determinar el nivel de servicio o planificar y diseñar correctamente una construcción, rehabilitación o mejora de una obra vial. Por lo que, la alternativa que se presenta en la presente tesis suma esfuerzos para mejorar el proceso de recolección de datos de manera eficiente y con menores costos.

- La evolución de los procesadores y la creación de computadoras más rápidas y autónomas permitirá que los tiempos de procesamiento disminuya en gran manera, derivando a un análisis mucho más ligero y veloz. Incluso podríamos pensar en una detección y clasificación en tiempo real, para ello, es necesario desarrollar redes neuronales cuyo procesamiento sea más ligero sin que pierdan precisión de detección y, un plus que aportaría en demasía, sería la evolución de procesadores y tarjetas gráficas más potentes a las actuales.

CONCLUSIONES

1. En la presente tesis se desarrolló un sistema automatizado de detección y clasificación vehicular basado en redes neuronales de Aprendizaje Profundo que permitió realizar aforos vehiculares en una intersección vial a través del uso de videograbaciones.
2. Se desarrolló un algoritmo basado en Visión Artificial, que en conjunto con los modelos de redes neuronales, permitió la detección y clasificación vehicular a través del procesamiento de imágenes y videos digitales.
3. Se procesaron los modelos de Redes Neuronales en el algoritmo de Visión Artificial para detectar y clasificar vehículos en videograbaciones de tránsito, lo que permitió determinar el aforo vehicular en una intersección vial.
4. Se analizaron y compraron los resultados de los aforos vehiculares de una intersección vial, obtenidos mediante los métodos de: i) red neuronal YoloV3, ii) red neuronal Detectron y iii) aforo manual.
5. Se determinaron los rendimientos y precisión de cada una de las redes neuronales que fueron utilizadas para la obtención de aforos vehiculares mediante el procesamiento de videos.
6. Tomando en consideración la precisión y tiempo de procesamiento de cada red neuronal analizada, se concluye que, para el procesamiento de videos, la red YoloV3 es muy efectiva y precisa en calidad de resultados y reducción de tiempo de procesamiento en comparación a la red Detectron2. Los resultados de Yolo son muy confiables al realizar un aforo vehicular.
7. La presente tesis ha logrado implementar y aplicar técnicas de Inteligencia Artificial en la automatización del proceso de detección y clasificación vehicular a través del área de visión computacional.
8. En este trabajo se presenta una alternativa accesible, rentable y económica para realizar un aforo vehicular de una vía y/o intersección mediante el uso de la inteligencia artificial.
9. El sistema propuesto puede ser implementado como parte de un Sistema Inteligente de Transporte (ITS), ya que podemos tener actualizado modelos de tráfico de manera constante haciendo uso de videocamaras, las cuales en mucho de los casos se encuentran ya instaladas (como el de las municipalidades). Con información actualizándose constantemente se podrán tomar medidas más oportunas y eficientes para la gestión del tráfico y/o mejoras en la infraestructura vial.

RECOMENDACIONES

1. El presente trabajo de investigación puede ampliarse y aplicarse con otros modelos de redes neuronales que surjan y se desarrollen por otras fuentes, utilizando la misma metodología propuesta en la presente tesis.

2. Con la evolución de los procesadores y la creación de computadoras más rápidas y autónomas será posible la reducción de los tiempos de procesamiento, derivando a un análisis mucho más preciso y veloz. Se recomienda continuar con el presente trabajo de investigación haciendo uso de equipos con mejores capacidades.

3. Se recomienda continuar con la investigación en el campo de la inteligencia artificial para la automatización de procesos de la inteligencia humana, que permitan disminuir los tiempos de trabajo del hombre y que estos sean aprovechados en otras actividades.

4. Finalmente, se recomienda buscar otras alternativas que sumen esfuerzos para mejorar el proceso de recolección de datos de manera eficiente y con menores costos financieros y logísticos.

REFERENCIAS BIBLIOGRÁFICAS

Bagnato, J. I. (2020). Aprende Machine Learning. *Leanpub*.

Becerra, R. (2014). *Modelo Neuronal de Demanda de Transporte en Redes Viales Urbanas* (Tesis Posgrado). Universidad Nacional de Ingeniería.

Benítez, R., Escudero, G., Kanaan, S., y Masip Rodó, D. (2014). *Inteligencia Artificial Avanzada*. Barcelona, España: Editorial UOC.

Boden, M. (2017). *Inteligencia artificial.* Madrid, España: Turner.

Cea, E., Pincheira, M., y Figueroa, J. (2018). *Estudio y aplicación de la librería OpenCV sobre la arquitectura ARM, para el control de agentes robóticos móviles usando visión artificial.* (Tesis Doctoral no publicada). Universidad del Bío-Bío.

IBM Cloud Education. (2020). AI vs. Machine Learning vs. Deep Learning vs. Neuronal Networks: What's the Difference? *IBM*.

Lasse, R. (2018). *Inteligencia artificial: 101 cosas que debes saber hoy sobre nuestro futuro.* Barcelona, España: Alienta Editorial.

Montenegro, W. (2018). *Automatización del monitoreo y diagnóstico del comportamiento vehicular y optimización del tiempo de recorrido en una red vial.* (Tesis Doctoral no publicada). Universidad Nacional de Ingeniería. Lima, Perú.

Núñez, F. (2016). *Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional* (Tesis Doctoral no publicada). Universitat Oberta de Catalunya.

Peña, J. (2017). *Sistema de detección y conteo de vehículos utilizando visión artificial.* (Tesis Doctoral no publicada). Universidad Nacional de Piura. Piura, Perú.

Ramachandran, P., Zoph, B., y Le, Q. (2017). Searching for activation functions. *arXiv preprint*. Descargado de https://arxiv.org/abs/1710.05941

Redmon, J., y Farhadi, A. (2018, apr). YOLOv3: An Incremental Improvement. Descargado de https://arxiv.org/abs/1804.02767

Vector ITC. (2018). Inteligencia Artificial. Pasado, presente y futuro. *Vector ITC*(Digital&Innovation).

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., y Girshick, R. (2019). *detectron2.* Descargado de https://github.com/facebookresearch/detectron2

ANEXOS

ANEXO A: CÓDIGOS EN PYTHON

A continuación se presentan los códigos desarrollados en *Python* utilizados en la presente tesis. El Código A.1 muestra el algoritmo usado para la detección y clasificación vehicular en imagenes utilizando las redes neuronales Faster R-CNN y RetinaNet. El Código A.2 muestra el algoritmo usado para la detección y clasificación vehicular en imagenes utilizando la red neuronal YoloV3. Y el Código A.3 muestra el algoritmo usado para la detección y clasificación vehicular en videos.

Algoritmo de detección y clasificación vehicular en imágenes, para las redes neuronales Faster R-CNN y RetinaNet

```
# Importamos liberías
from torchvision.models import detection
import numpy as np
import argparse
import torch
import time
import cv2
import os
import pandas as pd
from pandas import ExcelWriter
from collections import Counter

# Definimos los argumentos
ap = argparse. ArgumentParser()
ap.add_argument("-i", "--image", type=str, required=True,
  help="Ruta de la imagen de entrada")
ap.add_argument("-o", "--output", type=str, required=True,
  help="Ruta de la imagen de salida")
ap.add_argument("-m", "--model", type=str, default="frcnn-resnet",
  choices=["frcnn-resnet", "retinanet"],
  help="Modelo de red neuronal que utilizará")
ap.add_argument("-l", "--labels", required=True,
  help="Ruta de la lista de categorías COCO dataset")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
  help="Confianza mínima para filtrar detecciones débiles")
args = vars(ap.parse_args())

# Elegimos el procesador a utilizar
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu"
    )

```

```
# Cargamos la lista de etiquetas (C O C O dataset) y generamos un color aleatorio para cada etiqueta.
labelsPath = os.path.sep.join([args["labels"], "coco.names"])
CLASSES = open(labelsPath).read().strip().split("\n")
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# Definimos una lista para almacenar los objetos detectados
Lista_Detectada=[]

# Especificamos las funciones PyTorch que llaman a los modelos de red
MODELS = {
  "frcnn-resnet": detection.fasterrcnn_resnet50_fpn,
  "retinanet": detection.retinanet_resnet50_fpn}

# Cargamos el modelo de red que se utilizará y lo configuramos en modo evaluación
model = MODELS[args["model"]](pretrained=True, progress=True, pretrained_backbone=True).to(DEVICE)
model.eval()

# Cargamos la imagen de entrada y creamos una copia
image = cv2.imread(args["image"])
orig = image.copy()

# Convertimos la imágen BGR a RGB y ordenamos los canales
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image = image.transpose((2, 0, 1))

# Agregamos una dimensión a la imagen, escalamos sus pixeles al rango de [0,1] y la convertimos a un tensor de tipo punto flotante
image = np.expand_dims(image, axis=0)
image = image / 255.0
image = torch.FloatTensor(image)

# Enviamos la imagen al procesador y la pasamos por la red para obtener las detecciones
image = image.to(DEVICE)
start = time.time()
detections = model(image)[0]
end = time.time()

# Imprimimos el tiempo que dura el proceso
print("[INFO] El proceso duró {:.6f} segundos".format(end - start))

# Creamos un bucle para recorrer cada detección
for i in range(0, len(detections["boxes"])):

```

```
73    # Extraemos la confianza asociada con la detección
74    confidence = detections["scores"][i]
75
76    # Filtramos la detección asegurándonos que la confianza sea mayor
        que la confianza mínima
77    if confidence > args["confidence"]:
78
79      # Extraemos la etiqueta del objeto detectado y las coordenadas
        de su cuadro delimitador (x, y)
80      idx = int(detections["labels"][i])
81      box = detections["boxes"][i].detach().cpu().numpy()
82      (startX, startY, endX, endY) = box.astype("int")
83
84        # Imprimimos la predicción en nuestro terminal
85      label = "{}: {:.2f}%".format(CLASSES[idx], confidence * 100)
86      print("[DETECCION] {}".format(label))
87
88      # Agregamos el objeto detectado a nuestra lista de detecciones
89      Lista_Detectada.append(CLASSES[idx])
90
91        # Dibujamos el cuadro delimitador y su etiqueta
92      cv2.rectangle(orig, (startX, startY), (endX, endY),
93        COLORS[idx], 2)
94      y = startY - 15 if startY - 15 > 15 else startY + 15
95      cv2.putText(orig, label, (startX, y),
96        cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
97
98  #Contamos los objetos detectados y los clasificamos de acuerdo a sus
        etiquetas
99  ConteoEtiquetas=Counter(Lista_Detectada)
100 Data = pd.DataFrame({"[TOTALES]":ConteoEtiquetas})
101 print(Data)
102
103 # Visualizamos y guardamos la imagen procesada (imagen de salida)
104 cv2.imshow("output", orig)
105 cv2.imwrite(args["output"], orig)
106
107 #Creamos un excel y guardamos la informacion
108 ubicacionexcel = args["output"].replace(".jpg",".xlsx")
109 escritor=pd.ExcelWriter(ubicacionexcel, engine="xlsxwriter")
110 Data.to_excel(escritor)
111 escritor.save()
112
113 print("¡PROCESO EXITOSO!")
114 cv2.waitKey(0)
```

Código A.1: Algoritmo de detección y clasificación vehicular en imágenes, para las redes neuronales Faster R-CNN y RetinaNet

Algoritmo de detección y clasificación vehicular en imágenes, para la red neuronal YoloV3

```
# Importamos librerías
import numpy as np
import argparse
import time
import cv2
import os
import pandas as pd
from pandas import ExcelWriter
from collections import Counter

# Definimos los argumentos
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
  help="Ruta de la imagen de entrada")
ap.add_argument("-o", "--output", required=True,
  help="Ruta de la imagen de salida")
ap.add_argument("-y", "--yolo", required=True,
  help="Ruta del directorio YOLO")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
  help="Confianza mínima para filtrar detecciones débiles")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
  help="Umbral para aplicar supresión no-máxima")
args = vars(ap.parse_args())

# Cargamos la lista de etiquetas COCO dataset
labelsPath = os.path.sep.join([args["yolo"], "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# Generamos un color aleatorio para cada etiqueta.
COLORS = np.random.uniform(0, 255, size=(len(LABELS), 3))

# Definimos una lista para almacenar los objetos detectados
Lista_Detectada=[]

# Leemos los pesos de la red (YOLO wights) y la configuración del
    modelo
weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])

# Cargamos el detector de objetos YOLO
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

# Cargamos la imagen de entrada
image = cv2.imread(args["image"])
(H, W) = image.shape[:2]

```

```
46 # Determinamos los nombres de las capas de YOLO
47 ln = net.getLayerNames()
48 ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
49
50 # Pasamos la imagen de entrada por la red YOLO
51 # Obtenemos los cuadros delimitadores y las probabilidades asociadas
52 # Calculamos el tiempo del proceso
53 blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416),
54   swapRB=True, crop=False)
55 net.setInput(blob)
56 start = time.time()
57 layerOutputs = net.forward(ln)
58 end = time.time()
59
60 # Imprimimos el tiempo del proceso
61 print("[INFO] El proceso YOLO duró {:.6f} segundos".format(end -
       start))
62
63 # Inicializamos la lista de cuadros delimitadores, confianza y clase
       de ID
64 boxes = []
65 confidences = []
66 classIDs = []
67
68 # Bucle sobre cada una de las capas de salida
69 for output in layerOutputs:
70   # Bucle sobre cada una de las detecciones
71   for detection in output:
72     # Extramos la clase ID y la confianza
73     scores = detection[5:]
74     classID = np.argmax(scores)
75     confidence = scores[classID]
76
77     # Filtramos la detección asegurándonos que la confianza
78     # sea mayor que la confianza mínima
79     if confidence > args["confidence"]:
80       # Escalamos las coordenadas del cuadro delimitador con
81       # relación a la imagen de entrada.
82       # YOLO identifica las coordenadas centrales (x, y) del
83       # cuadro delimitador y su ancho y alto.
84       box = detection[0:4] * np.array([W, H, W, H])
85       (centerX, centerY, width, height) = box.astype("int")
86
87       # Utilizamos las coordenadas del centro (x, y), ancho y alto
88       # para determinar las coordenadas de la esquina superior
       izquierda.
89       x = int(centerX - (width / 2))
90       y = int(centerY - (height / 2))
91
```

```
92          # Actualizamos la lista de coordenadas, confianzas y clase ID
93          boxes.append([x, y, int(width), int(height)])
94          confidences.append(float(confidence))
95          classIDs.append(classID)
96
97  # Suprimimos cuadros delimitadores débiles y superpuestos
98  idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],
99    args["threshold"])
100
101 # Nos aseguramos que exista al menos una deteccion
102 if len(idxs) > 0:
103   # bucle sobre los indices obtenidos
104   for i in idxs.flatten():
105
106     # Extraemos las coordenadas del cuadro delimitador
107     (x, y) = (boxes[i][0], boxes[i][1])
108     (w, h) = (boxes[i][2], boxes[i][3])
109
110     # Dibujamos el cuadro delimitador y su etiqueta
111     color = [int(c) for c in COLORS[classIDs[i]]]
112     cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
113     text = "{}: {:.2f}%".format(LABELS[classIDs[i]], confidences[i
        ]*100)
114     print("[DETECCION] {}".format(text))
115     cv2.putText(image, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
116       0.5, color, 2)
117
118     # Agregamos el objeto detectado a nuestra lista de detecciones
119     Lista_Detectada.append(LABELS[classIDs[i]])
120
121 #Contamos los objetos detectados y los clasificamos de acuerdo a sus
        etiquetas
122 ConteoEtiquetas=Counter(Lista_Detectada)
123 Data = pd.DataFrame({"[TOTALES]":ConteoEtiquetas})
124 print(Data)
125
126 # Visualizamos y guardamos la imagen procesada (imagen de salida)
127 cv2.imshow("Image", image)
128 cv2.imwrite(args["output"], image)
129
130 print("¡PROCESO EXITOSO!")
131 cv2.waitKey(0)
```

Código A.2: Algoritmo de detección y clasificación vehicular en imágenes, para la red neuronal YoloV3

Algoritmo de detección y clasificación vehicular en videos

```
'''
Ivy-master (main.py)

Autor: Nicholas Kajoh
Aportes: David Nizama
'''
import sys
import time
import cv2

from dotenv import load_dotenv
load_dotenv()

import settings
from util.logger import init_logger
from util.image import take_screenshot
from util.logger import get_logger
from util.debugger import mouse_callback
from ObjectCounter import ObjectCounter
from util.excel import save_excel

init_logger()
logger = get_logger()


def run():
    '''
    Inicialice el contador de objetos y ejecute el bucle de conteo.
    '''

    video = settings.VIDEO
    cap = cv2.VideoCapture(video)
    if not cap.isOpened():
        logger.error('Invalid video source %s', video, extra={
            'meta': {'label': 'INVALID_VIDEO_SOURCE'},
        })
        sys.exit()
    retval, frame = cap.read()
    f_height, f_width, _ = frame.shape
    detection_interval = settings.DI
    mcdf = settings.MCDF
    mctf = settings.MCTF
    detector = settings.DETECTOR
    tracker = settings.TRACKER
    use_droi = settings.USE_DROI
    # create detection region of interest polygon
    droi = settings.DROI \
```

```
48              if use_droi \
49              else [(0, 0), (f_width, 0), (f_width, f_height), (0,
    f_height)]
50      show_droi = settings.SHOW_DROI
51      counting_lines = settings.COUNTING_LINES
52      show_counts = settings.SHOW_COUNTS
53      hud_color = settings.HUD_COLOR
54
55      object_counter = ObjectCounter(frame, detector, tracker, droi,
    show_droi, mcdf, mctf,
56                                     detection_interval,
    counting_lines, show_counts, hud_color)
57
58      record = settings.RECORD
59      if record:
60          # initialize video object to record counting
61          output_video = cv2.VideoWriter(settings.OUTPUT_VIDEO_PATH, \
62                                         cv2.VideoWriter_fourcc(*'
    MJPG'), \
63                                         30, \
64                                         (f_width, f_height))
65
66      print ('========== INICIALIZANDO...\ ==========')
67      logger.info('========== ¡PROCESO INICIALIZADO! ==========')
68      start=time.time()
69
70      headless = settings.HEADLESS
71      if not headless:
72          # capture mouse events in the debug window
73          cv2.namedWindow('Debug')
74          cv2.setMouseCallback('Debug', mouse_callback, {'frame_width'
    : f_width, 'frame_height': f_height})
75
76      is_paused = False
77      output_frame = None
78      frames_count = round(cap.get(cv2.CAP_PROP_FRAME_COUNT))
79      frames_processed = 0
80
81      try:
82          # main loop
83          while retval:
84              k = cv2.waitKey(1) & 0xFF
85              if k == ord('p'): # pause/play loop if 'p' key is
    pressed
86                  is_paused = False if is_paused else True
87                  logger.info('PAUSAR/REPRODUCIR', extra={'meta': {'EN
     PAUSA': is_paused,}})
88              if k == ord('s') and output_frame is not None: # save
    frame if 's' key is pressed
```

```
89                  take_screenshot(output_frame)
90              if k == ord('q'): # end video loop if 'q' key is pressed
91                  #logger.info('FINALIZAR', extra={'meta': {'PROCESO':
    'FINALIZADO'}})
92                  break
93
94              if is_paused:
95                  time.sleep(0.5)
96                  continue
97
98              _timer = cv2.getTickCount() # set timer to calculate
    processing frame rate
99
100             object_counter.count(frame)
101             output_frame = object_counter.visualize()
102
103             if record:
104                 output_video.write(output_frame)
105
106             if not headless:
107                 debug_window_size = settings.DEBUG_WINDOW_SIZE
108                 resized_frame = cv2.resize(output_frame,
    debug_window_size)
109                 cv2.imshow('Debug', resized_frame)
110
111             processing_frame_rate = round(cv2.getTickFrequency() / (
    cv2.getTickCount() - _timer), 2)
112             frames_processed += 1
113             logger.debug('Frame processed.', extra={
114                 'meta': {
115                     'label': 'FRAME_PROCESS',
116                     'frames_processed': frames_processed,
117                     'frame_rate': processing_frame_rate,
118                     'frames_left': frames_count - frames_processed,
119                     'percentage_processed': round((frames_processed
    / frames_count) * 100, 2),
120                 },
121             })
122
123             retval, frame = cap.read()
124     finally:
125         # end capture, close window, close log file and video object
     if any
126         cap.release()
127         if not headless:
128             cv2.destroyAllWindows()
129         if record:
130             output_video.release()
131         logger.info('========== ¡PROCESO FINALIZADO! ==========')
```

```
132             logger.info('========== RESULTADOS ==========', extra={
133                 'meta': {'CONTEO FINAL': object_counter.get_counts(),
134 
135                 },
136             })
137             end=time.time()
138             print("========== INFO     : El proceso duró {:.4f} segundos".
        format(end - start))
139 
140             save_excel(object_counter.get_counts())
141 
142 if __name__ == '__main__':
143     run()
```

Código A.3: Algoritmo de detección y clasificación vehicular en videos