UNIVERSIDAD DEL AZUAY

FACULTAD DE CIENCIAS DE LA ADMINISTRACIÓN
ESCUELA DE INGENIERÍA DE SISTEMAS Y TELEMÁTICA

Rendimiento de Raspberry Pi 3B+ con Intel NCS ejecutando métodos de aprendizaje profundo neuronal en clasificación de tipo de vehículos en tiempo continuo.

TESIS PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS Y TELEMÁTICA

Autor: David Andrés Tapia Moscoso

Director: Juan Gabriel Barros Gavilanes

Cuenca, 2020

ECUADOR

DEDICATORIA

para mi familia y aquellos que me han apoyado incondicionalmente en el transcurso de mis estudios universitarios.

AGRADECIMIENTO

Gracias, padres por todo el tiempo que me han dedicado y los valores que me han compartido, su ejemplo me ha enseñado a ser una persona ética y responsable. Gracias familia, sin su apoyo incondicional no podría haber llegado a ser la persona que soy ahora. Un agradecimiento especial para mis revisores, Pao Vázquez y Gabriel Barros; sin su apoyo este proyecto no habría sido posible. Un agradecimiento grande a mis compañeros, profesores y a la Universidad del Azuay. Adicionalmente, estoy muy agradecido con el Ing. Pablo Esquivel, Ing. Oswaldo Silva e Ing. Fernando Balarezo y todo el departamento de T.I. por la motivación constante para culminar con el trabajo de titulación.

ÍNDICE

RESUMEN

La clasificación de tipo de vehículos y su conteo en las calles de una ciudad, ayuda a tomar decisiones respecto a la construcción y ampliación de vías, cuantificación del tráfico y necesidades de ciclo vías. Sin embargo, se requiere de un sistema automático que realice esta clasificación de forma continua en un flujo de video. El objetivo de esta investigación es reportar el rendimiento de dos métodos de detección y clasificación de actores de movilidad basados en *Deep Neural Networks (DNN)* ejecutándose en un Raspberry Pi 3B+, permitiendo identificar el modelo que brinda mayor precisión con menor consumo de recursos. Se comparó *You Only Look Once v3* (*YOLO v3*) *y Single Shot MultiBox Detector (SSD).* Los resultados demuestran que *SSD* es la mejor alternativa siendo hasta trece veces más veloz que *YOLO v3*. Durante la ejecución del sistema utilizando el *Intel Neural Compute Stick 2* con *SSD* se tiene un consumo promedio del CPU del Raspberry Pi 3B+ del 10% y 15% de memoria RAM. El valor de referencia de mAP para *SSD* es del 72.7%.

Palabras clave: Medición de rendimiento, Raspberry Pi 3B+, *You Only Look Once v3*, *Single Shot MultiBox Detector*, *Deep Neural Networks*, *Neural Compute Stick*, Actores de movilidad, Conteo.

Abstract

The classification and counting of the type of vehicles in the streets of a city helps to make decisions regarding the construction and expansion of roads, quantification of traffic, and needs of bicycle lanes. However, there is a need for an automatic system to perform this classification continuously on a video flow. The objective of this research is to report the performance of two Deep Neural Networks (DNN) based methods running on a Raspberry Pi 3B+ to classify and detect mobility actors. This allows the identification of the model that provides greater precision with less resource consumption. The following methods were compared: You Only Look Once v3 (YOLO v3) and Single Shot MultiBox Detector (SSD). The results showed that SSD is the best option in up to thirteen times faster than YOLO v3. During the testing with the Intel Neural Compute Stick 2 and SSD there is a CPU usage of 10% and 15% of RAM memory usage from the Raspberry Pi 3B+. The reference mAP value for the SSD model used is 72.7%.

Keywords: Performance measure, Raspberry Pi 3B+, You Only Look Once v3, Single Shot MultiBox Detector, Deep Neural Networks, Neural Compute Stick, Mobility Actors, Counting.

UNIVERSIDAD DEL AZUAY
Dpto. Idiomas

Translated by

David Tapia

Capítulo I

PLANTEAMIENTO DE LA INVESTIGACIÓN

INTRODUCCIÓN

1.1 Motivación

La motivación de la investigación nace con el deseo de conocer qué método de redes neuronales profundas (DNN) es más eficiente en el consumo de memoria y procesamiento cuando se ejecuta en un minicomputador para realizar la clasificación de vehículos. Esto es importante porque si el rendimiento de los métodos propuestos representa una mejora frente al proceso que realiza un ser humano, entonces se puede implementar en las calles de una ciudad para la clasificación y conteo de vehículos a un costo bajo. La clasificación y conteo permitirá obtener estadísticas sobre los niveles de tráfico, tipo de vehículos utilizados y uso de bicicletas; promoviendo y mejorando la toma de decisiones respecto a la construcción y ampliación de vías, cuantificación del tráfico y requerimiento de ciclovías [1]. Otra razón importante que conforma la motivación según [2] es la esperanza de obtener mayor precisión en comparación a los métodos que usan el componente humano. Una ventaja adicional con respecto al método sugerido que el costo es económico en comparación a otros que incluyen sensores costosos, como los que incluyen tecnologías láser [3] y calor térmico de las vías [2].

1.2 Problemática y justificación

Los métodos para clasificación de objetos basados en *Deep Neural Networks (DNN)* consumen una cantidad elevada de recursos y se ejecutan en servidores o clústeres [4]. Sin embargo, se carece de suficiente información sobre el análisis del rendimiento al implementarlos en minicomputadores [5], [2] y [6]. Debido a las limitaciones del *hardware* del equipo candidato (Raspberry Pi 3B+), la capacidad de procesamiento es limitada, por lo cual resulta complicado incorporar procesos que consumen alta cantidad de recursos, como lo hacen los *DNNs*. Por otro lado, existe evidencia que al adicionar el *Intel Neural Compute Stick*, se disminuye notablemente la carga de recursos en el equipo embebido [6]. Generalmente cuando se selecciona un método se sacrifica velocidad por precisión y viceversa [7]. Sin embargo, se desconocen las configuraciones para una implementación en la cual se pueda alcanzar a realizar el procesamiento en tiempo continuo de un flujo de video en base a los recursos disponibles. Existen sistemas disponibles para el reconocimiento, pero son de código propietario, por lo que

resulta una complicación acceder a la información de éstos [2]. En la práctica usualmente una persona es quien realiza el conteo, con la limitación que sólo es un muestreo de datos y con elevados costos de personal. Esto se contrasta con el uso de un dispositivo embarcado, cuyo costo es bajo y funciona de manera continua todos los días del año [6].

El valor económico de realizar el conteo y clasificación con personal contratado y utilizando otros métodos actuales resulta más costoso que realizarlo en un dispositivo embarcado. Por ejemplo, en Ecuador el valor de un salario básico de una persona mensualmente es de 394.00$ [8] , sin contar con los beneficios adicionales de la ley. Esto permitiría realizar un conteo de objetos durante un lapso de 8 horas al día; para realizar el conteo de objetos de un flujo de video de las 24 horas se requeriría haber grabado ese lapso y contar con 4 personas para realizar este trabajo. Sin embargo, es posible disminuir los costos notablemente si es que se realiza la implementación propuesta, es decir utilizar un Raspberry Pi 3B+ juntamente con el Intel *NCS 2*. La inteligencia artificial (IA) permite automatizar tareas repetitivas de los humanos.

Este trabajo es para conocer sobre las limitaciones de la IA (específicamente *DNN*) en dispositivos embarcados. Existen numerosas fuentes de video, pero el análisis y extracción de información es costoso por el factor humano, con la exploración de esta tecnología se busca disminuir el mismo. Paralelamente se quiere indagar y apoyar a la investigación de proyectos en el ámbito que respecta a la sociedad/comunidad, mediante esta investigación se podrían obtener datos que ayudan a visualizar el estado del tráfico en las vías de una ciudad y nuevos posibles requerimientos, como la construcción de vías públicas.

1.3 Objetivos de la investigación

1.3.1 General

Reportar el rendimiento, en términos de medidas (precisión, exhaustividad, exactitud y consumo de memoria RAM y procesador), de Raspberry Pi 3B+ con Intel *NCS 2* ejecutando dos métodos basados en *DNN* para el conteo y clasificación de actores de movilidad.

1.3.2 Específicos

1.3.2.1 Revisar la literatura sobre técnicas de rastreo de objetos, modelos *DNN*, rendimiento de estos en dispositivos embarcados, posibles medidas para evaluar el rendimiento y limitaciones de hardware en Raspberry Pi 3B+ e Intel *NCS 2*.

1.3.2.2 Seleccionar plataformas que permitan emplear *DNN* y experimentos que permitan obtener medidas para evaluar el rendimiento. Seleccionar dos modelos de *DNN* y utilizarlos con el método del centroide para el rastreo de objetos. Por último, se debe seleccionar la fuente de video para aplicar la clasificación y conteo.

1.3.2.3 Preparar los modelos de *DNN* a utilizar para ejecución en dispositivos embarcados con el dispositivo Intel *Neural Compute Stick 2*. Preparar el sistema del dispositivo embarcado para ejecutar los métodos propuestos.

1.3.2.4 Probar los modelos con y sin el dispositivo Intel *NCS 2*, documentar los resultados obtenidos en términos de medidas de consumo de CPU y Memoria RAM en porcentaje y tiempo. Comparar los resultados y proponer temas de discusión y mejoras para el futuro.

Capítulo II

ESTADO DEL ARTE

2.1 Marco conceptual

Existen varios métodos para el conteo y clasificación de objetos. Gran cantidad de métodos de conteo y clasificación de vehículos que se basan en la visión por computador. Una minoría se basa en otros conceptos como –por ejemplo– aquellas basadas en tubos neumáticos, en la cual se identifica el tráfico en base a mapas de calor [2]. Para aquellos basados en visión por computador existen múltiples alternativas para la detección, clasificación y el rastreo de los objetos: los métodos tradicionales basados en reglas como combinaciones de *Support Vector Machine (SVM)* y *Pyramid Histogram Oriented Gradients* [9], así como los nuevos métodos basados en redes neuronales profundas (*DNN*). En este ámbito, las redes más populares son las convolucionales (*CNN: Convolutional Neural Network*) por los resultados de alta precisión y exhaustividad que han presentado. Estos han alcanzado el mejor rendimiento en el ámbito de detección y clasificación de objetos y su uso es altamente recomendado [4] y [6]. La investigación presente se enfoca en los métodos derivados de *CNN*.

Para las medidas de evaluación de los resultados, lo más importante en este proyecto es evaluar el rendimiento de cada método en el dispositivo embarcado, sin embargo, como se comenta en [7] “Las métricas estándar de exactitud como por ejemplo *mean average precision* (mAP) no son suficientes para decidir una arquitectura en base a una aplicación”, por este motivo para hacer un mejor criterio de evaluación se consideran dos métricas adicionales a las existentes de cada método: el porcentaje de uso de memoria RAM y CPU. Las métricas por defecto con las que cuenta cada método al momento de ser elaborados pueden variar dependiendo del entorno y el tamaño de objeto presentado al clasificador, también pueden variar en base al funcionamiento del sistema como tal. Para constatarlo se obtendrán las siguientes métricas de las fuentes de videos cortos propuestos: precisión, exhaustividad y exactitud. Las tres últimas métricas mencionadas se obtienen de la matriz de confusión agregada como lo hace [9] con respecto a los siguientes criterios: detecciones verdaderas negativas, verdaderas positivas, falsas negativas y, por último, detecciones falsas positivas.

Predicted classes \ Ground truth classes	C_1	C_2	$\cdots$	C_N	FP
C_1	$c_{1,1}$	$c_{1,2}$	$\cdots$	$c_{1,N}$	$c_{1,N+1}$
C_2	$c_{2,1}$	$c_{2,2}$	$\cdots$	$c_{2,N}$	$c_{2,N+1}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
C_N	$c_{N,1}$	$c_{N,2}$	$\cdots$	$c_{N,N}$	$c_{N,N+1}$
FN	$c_{N+1,1}$	$c_{N+1,2}$	$\cdots$	$c_{N+1,N}$	$c_{N+1,,N+1}$

Figura 1. Ejemplo de matriz de confusión agregada. Imagen obtenida de [9].

2.2 Métodos modernos

La figura 2 presenta los resultados de la medida estándar de exactitud en este tipo de métodos: *Mean Average Precision(mAP).* Estos resultados se obtienen al ejecutar distintas alternativas que incorporan *CNN* en un computador con las siguientes características: 32GB RAM, Intel Xeon E5-1650 v2 y tarjeta gráfica NVIDIA GeForce GTX Titan X. Esta tarjeta contiene 3072 CUDA Cores y 12 GB de memoria RAM (modelo GDDR5) dedicada. Los tiempos son reportados en GPU para un tamaño de lote equivalente a 1, los tamaños de las imágenes de prueba son de 300x300px y de 600x600px:

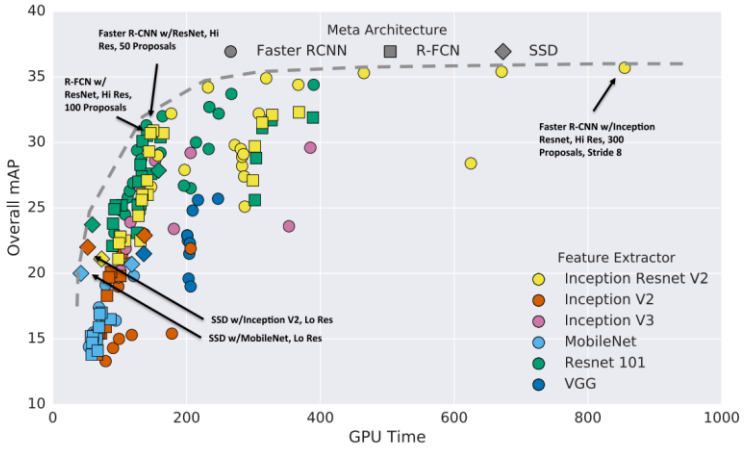


Figura 2: Precisión en mAP vs Tiempo de *GPU* en milisegundos. Grafica tomada de [7].

Los métodos de conteo usualmente cuentan con cinco etapas: captura, detección, clasificación, rastreo y opcionalmente procesamiento de resultados [2], [9]. Los métodos modernos basados en *CNN* reducen la sección de detección y clasificación a un paso. Algunos métodos conocidos actualmente son *YOLO* [10], *YOLO v2* [11], *YOLO v3* [12], *SSD* [13], *R-CNN* y *Fast R-CNN* [14], *Faster R-CNN* [15].

YOLO representa un nuevo enfoque para detectar objetos. Además, este modelo unificado tiene varios beneficios sobre los métodos tradicionales de detección de objetos, como velocidad, y capacidad de inferencia para clasificar los tipos de objetos en la imagen cuando hace predicciones; esto gracias a que en su fase de entrenamiento aprende representaciones generalizables de objetos. Sin embargo, existen mejores métodos en cuanto a lo que la precisión refiere en sistemas de detección de última generación [7].

Motivado por la mejora de *Faster R-CNN*, Redmon y Farhadi [11] propusieron un *YOLO* mejorado (denominado *YOLO v2*) donde se utilizan cuadros de anclaje para predecir los cuadros delimitadores. Para lograrlo tuvieron que remover las capas totalmente conectadas del método anterior *(YOLO)*. Para facilitar el procedimiento de entrenamiento para la sección de cuadros de anclaje, se ejecutó la agrupación *k-means* en el conjunto de entrenamiento de cuadros delimitadores para seleccionar automáticamente los buenos antecedentes y así mejorar la precisión del modelado, usando además una nueva arquitectura de red de *CNN*.

YOLO divide la imagen de entrada en una cuadrícula de tamaño SxS y predice B cuadros delimitadores para cada celda de cuadrícula (para evaluar *YOLO* en Pascal VOC se utilizó S=7 y B=2). Simultáneamente, cada celda de la cuadrícula predice un conjunto de probabilidades de clase (C) (en el caso de Pascal VOC fue equivalente a 20 clases), independientemente del número de casillas. *YOLO* puede procesar imágenes a 45 marcos por segundo (sus siglas en inglés: FPS) con un GPU NVIDIA Titan X y llega a obtener la media de precisión promedio *(mAP)* de 63.4% en el *dataset* Pascal VOC 2007 [10]. El *Single Shot Multi Box Detector* (*SSD*) es el actual sistema de detección de objetos de última generación, que logra un mAP de hasta 81.6% según la prueba ejecutada en el *dataset* Pascal VOC 2007 con imágenes de 512x512px [13]. Es interesante observar en la tabla 1 que alcanzó las siguientes precisiones promedio (*AP*) para las clases que nos interesan.

	Carro	Moto	Bus	Persona
AP	88.9%	88.3%	88.6%	84.6%

Tabla 1: Precisión media por clase de objeto para SSD, obtenida de [13].

Para enfrentar los inconvenientes presentados en la técnica anterior, propusieron un nuevo método llamado *Fast YOLO*, que acelera 3.3 veces la detección más el método *YOLO v2* para la detección de objetos en video en dispositivos embarcados a través de dos estrategias clave (una arquitectura optimizada y un método de inferencia adaptativo al movimiento) [16]. La combinación de estas dos estrategias da como resultado un marco de detección de objetos que mejora la velocidad de detección y clasificación para el video, haciéndolo una buena alternativa para dispositivos embarcados con requisitos de computación, memoria y energía restringidos. La siguiente mejora dispuesta en 2018 es *YOLO v3*. El método incluye mejoras de rendimiento, un mejor clasificador, entre otros [12].

Volviendo a los métodos de redes neuronales convolucionales: según [15] en la introducción se menciona que *Fast R-CNN* puede lograr velocidades casi en tiempo real (en nuestro caso 30 FPS, varía dependiendo de la cámara utilizada) utilizando redes profundas cuando se ignora el tiempo gastado en las propuestas de región. (Generalmente esto se basa en un procesador gráfico NVIDIA GeForce GTX Titan X en las pruebas indicadas en [7]). Según los mismos autores las propuestas de región son el cuello de botella computacional en los sistemas de detección de última generación. *Faster R-CNN* es un método particularmente exitoso para la detección general de objetos. Consta de dos componentes: el *Region Proposal Network* (conocida por sus siglas *RPN*) totalmente convolucional para proponer las regiones candidatas y un clasificador que también funciona con *CNN*. A pesar de tener excelentes resultados en la detección y clasificación de objetos de forma general, *Faster R-CNN* tiene limitantes al momento de detectar peatones, esto causado sorprendentemente por la degradación de resultados por el clasificador que se emplea; se estima que existen dos motivos: el primero falta de resolución en mapas de características para manejo de instancias pequeñas, y el segundo falta de estrategia de arranque para extraer ejemplos negativos complicados [17].

Estos métodos descritos en los párrafos anteriores pueden ser implementados mediante distintas plataformas, como por ejemplo *Caffe* [18], *Tensorflow* [19] y otros. Es importante aclarar que todos los métodos mencionados realizan detección y clasificación de objetos directamente dentro

de su estructura. Los *datasets* que se emplean para evaluar y en algunos casos entrenar estos modelos por lo general incluyen a los siguientes: COCO y PASCAL VOC.

Resumen de métodos modernos	**Datasets**	COCO	PASCAL VOC	
	Métodos de clasificación y detección	*R-CNN, Fast R-CNN y Faster R-CNN*	*YOLO, YOLO v2, Fast YOLO, YOLO v3*	*SSD*
	Equipos generales con los que se ha probado	NVIDIA GeForce GTX Titan X	NVIDIA Tesla M40	

Tabla 2: Resumen de métodos modernos, datasets, modelos y equipos.

Existen múltiples sistemas de clasificación, pero los métodos de *Deep Learning* no han sido aplicados aún a minicomputadoras que incluyan un Intel *NCS 2*, un dispositivo que integra una unidad de procesamiento visual (*VPU)*. Según la lectura bibliográfica revisada hasta octubre del 2019 tampoco se encuentra que se ha evaluado el rendimiento en este ámbito en la clasificación de vehículos. Es necesario probar los métodos *YOLO v3* y *SSD,* verificar el rendimiento y si es que se alcanza a clasificar los vehículos en tiempo continuo de un flujo de video. Es importante evaluar las distintas posibilidades, la tendencia del futuro es hacia la portabilidad y la reducción de consumo de recursos [1] y [6].

2.3 Métodos utilizados

2.3.1 Modelos de clasificación y detección.

Los métodos que usaremos se basan *Convolutional Neural Networks (CNN)* usadas por primera vez en 1989 [20]. De los mismos se escogió *YOLO v3* y *SSD*, debido a que según las evidencias encontradas en la lectura consumen menos recursos y tienen un valor de mAP similares a otros métodos actuales descritos en la sección de métodos modernos, siendo así estos los más indicados para ejecutarse en dispositivos embarcados como *(Raspberry Pi 3B+ y Smartphones);* también representan buenas alternativas dada la amplia documentación disponible. Además, tras un entrenamiento pueden ser modificados para customizar las clases de clasificación. El primer método, *Single Shot Detector (SSD)* es un rápido detector para varias categorías con resultados hasta del 74.3% en *Mean Average Precision (mAP)* a 59 *Frames Per Second (FPS)* en el dataset VOC2007 con un procesador grafico NVIDIA Titan X con una entrada de 512x512px. Este método es más rápido que *Faster Recurrent CNN* (*Faster R-CNN*) [13]. El segundo método seleccionado es *You Only Look Once (YOLO)* en su versión 3 publicado en el año 2018 con mejoras significativas en la precisión igualando a *SSD*, pero superándolo en velocidad como se puede observar en la figura 3, en donde se comparan múltiples métodos de detección de objetos. El extractor de características que se usa es *Darknet-53* llamado así porque tiene 53 capas convolucionales [12]. La velocidad de marcos por segundo en *YOLO* versión base es de 45 con un GPU Titan X [10] y en Darknet-53 se alcanza 78 FPS con GPU Titan X en 256x256px. Los tiempos que se observan en la gráfica pertenecen a un GPU NVIDIA Tesla M40 o NVIDIA Titan X. Es importante recordar que los resultados mencionados pueden variar dependiendo del dataset utilizado para las pruebas y las plataformas en las que se elaboraron. En algunos casos los algoritmos son optimizados para ejecutarse rápidamente bajo ciertos ambientes, por este motivo ha sido importante probar con el Raspberry Pi 3B+ e Intel *NCS 2*.

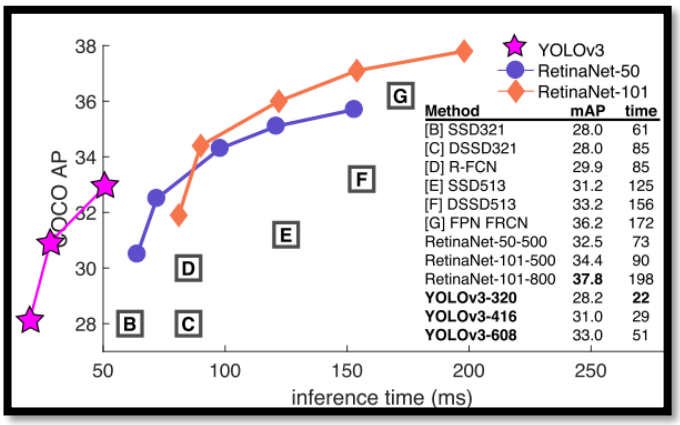


Figura 3: Comparación de métodos en términos de precisión y tiempo. Tomada de [12].

Se presentan más detalles técnicos sobre cada modelo juntamente con el dataset (por defecto) con el que se ha entrenado cada uno.

- MobileNet *SSD*
  i. Plataforma: Caffe
  ii. Dataset: MS-COCO para entrenamiento y VOC0712 para afinamiento.
  iii. mAP: 0.727
  iv. Fuente: [21]
  v. Transformación para *NCS* con *mvNCCompile* [22]
  vi. Clases disponibles: (*background, aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tvmonitor*).
- Frozen *YOLO v3*:
  i. Plataforma: Tensorflow
  ii. Dataset: MS-COCO
  iii. mAP: 0.6028 [23]
  iv. Fuente: [24]
  v. Transformador: OpenVINO
  vi. Clases disponibles: (*person, bicycle, car, motorbike, aeroplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball, bat, baseball glove, skateboard, surfboard, tennis*

racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, sofa, pottedplant, bed, diningtable, toilet, tvmonitor, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush).

- Frozen *Tiny YOLO v3*
  i. Plataforma: Tensorflow
  ii. Dataset: MS-COCO
  iii. mAP: 33.1 [25]
  iv. Fuente: [24]
  v. Transformador: OpenVINO
  vi. Clases disponibles: (*person, bicycle, car, motorbike, aeroplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, sofa, pottedplant, bed, diningtable, toilet, tvmonitor, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush*).

2.3.2 Diferencias entre *YOLO v3* y *SSD*:

Existen múltiples diferencias entre ambos detectores. Para esto es necesario entender el principio de operación de cada uno. En *YOLO v3* los conceptos más importantes son los siguientes: detectar cajas de anclaje, predicción de clase, predicción de clase en múltiples escalas con muestreo incremental(*upsampling*), extraer características (Darknet-53) y finalmente presentar los resultados. El número total de capas que contiene *YOLO v3* a través de todo el modelo es de 106 convolucionales. Esta es la causa por la que *YOLO v3* es más lento que *YOLO v2*, sin embargo, es más preciso. Se puede obtener una explicación exacta del funcionamiento de estas en el documento original de *YOLO v3* [12]. *SSD* por otro lado procura hacer la detección y clasificación en un intento, a diferencia de otros métodos que usan dos pasos: uno determinar las redes de propuestas de región y otro para detectar los objetos de cada región propuesta. Los conceptos más importantes de *SSD* son los siguientes: Se compone de un detector multicaja que identifica las regiones de posibles objetos, intenta extraer mapas de características y finalmente se aplican filtros

convolucionales para detectar los objetos [26]. Se pueden obtener detalles del funcionamiento de *SSD* en [13].

En la figura 4 se puede visualizar la Arquitectura de *YOLO v3*:

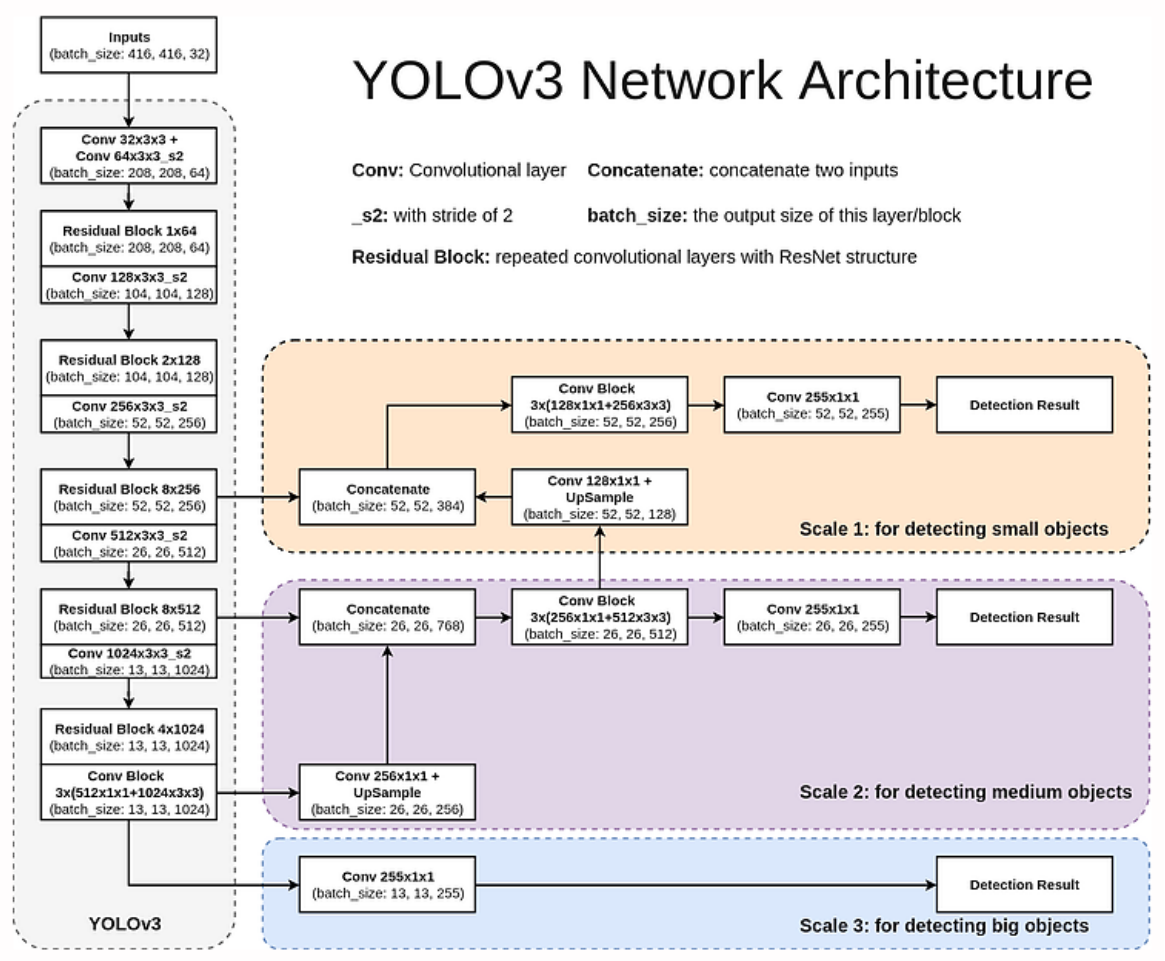


Figura 4: Arquitectura de *YOLO v3*. Imagen obtenida de: [27].

La figura 5 presenta la arquitectura de *SSD*:

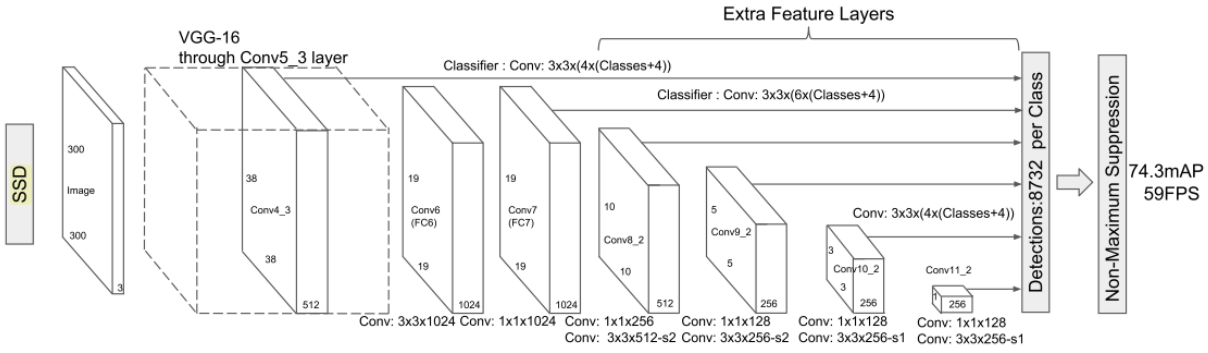


Figura 5: Arquitectura de *SSD*. Imagen obtenida de: [13].

Tiny YOLO v3: *Tiny YOLO v3* es prácticamente un modelo informal derivado de *YOLO v3* que ha pasado por algunas optimizaciones para hacer que su tamaño sea menor, por lo que resulta más veloz que *YOLO v3*. Se puede obtener más información sobre este método en el siguiente sitio: [25]

2.3.3 Datos sobre los *datasets*:

Como se mencionó previamente los principales *datasets* con los que fueron entrenados los métodos propuestos son: MS-COCO y en el caso de *SSD* para el afinamiento VOC0712. MS-COCO en ingles representa *Microsoft-Common Objects in Context* es un conjunto de datos de detección, segmentación y subtitulación de objetos a gran escala [28]. Las características se especifican en la tabla 3. VOC hace referencia a *Visual Object Classes*, inició como un proyecto de desafío de clases de objetos visuales, ahora se ha convertido en un punto de referencia en el reconocimiento y detección de categorías de objetos visuales, esto debido a su evaluación estándar de imágenes con sus respectivas anotaciones [29]. Más características se pueden visualizar en la tabla 3.

Dataset	**Numero de imágenes**	**Instancias de objetos**	**Categorías/Clases de objetos**
MS-COCO	330.000	1.5 millones	80
PASCAL VOC 2012	11.530	27.450	20

Tabla 3: Detalles sobre los datasets, datos obtenidos de [28] y [30].

2.3.4 Métodos de rastreo y conteo:

El funcionamiento general del sistema establecido se basa en la captura, detección, clasificación, el seguimiento y conteo de objetos. Para la etapa de detección y clasificación se aplican métodos de *deep learning* como *SSD* y *YOLO v3*, pero según lo leído se pueden requerir más recursos de los provistos por dispositivos embarcados como el Raspberry Pi 3B+. Por este motivo para hacer este procesamiento en tiempo continuo se buscó alternativas, una de ellas es ejecutar la detección y clasificación cada “C” marcos. El funcionamiento ideal sería poder ejecutar la detección y clasificación en cada marco, sin embargo, como existe un flujo generalmente de 30 marcos por segundo no afecta significativamente el saltarse 5 o menos marcos entre detecciones y clasificaciones. Después de la detección y clasificación el siguiente paso es realizar el seguimiento de los objetos. Para esto también existen múltiples técnicas, por ejemplo el método del centroide

[31], Filtro de Kalman [2], método de correlación y otros [32]. La técnica del centroide se puede realizar cada que se presenta una detección y clasificación; a diferencia del método de correlación, que tiene independencia. Es decir, el mismo puede hacer el seguimiento del objeto (incluso si cambia de escalas entre marcos) con excelentes resultados. Incluyendo los marcos intermedios donde que no se efectúan detecciones y clasificaciones. De los algoritmos investigados el más simple es el método del centroide, el más complicado y robusto es el método de correlación. Por este motivo se ha decidido probar con estas dos alternativas. El centroide se ejecuta en cada marco en el que existen detecciones y el método de correlación en todos los marcos.

- Centroide:

El método del centroide que se ha empleado en este proyecto fue obtenido de un ejemplo de [31]. El funcionamiento es simple y permite hacer el seguimiento y conteo de objetos determinados basándose en los centroides. En resumen, la operación del algoritmo analiza la ubicación del cuadro delimitador del objeto y calcula el centroide, en caso de ser el primer marco, registra los nuevos centroides con un ID correspondiente a un valor auto incremental que inicia en 1. Después de detectar objetos en el primer marco se toman en cuenta dos parámetros para agregar nuevos centroides, eliminar los antiguos y actualizar la ubicación de los existentes. Es decir, en el segundo paso a partir de los marcos subsiguientes al primero, en caso de haber nuevos y antiguos centroides se calcula la distancia Euclidiana entre los mismos. Esto nos permite identificar o asociar los centroides nuevos con los antiguos. Si existen centroides adicionales a los previos permite agregar un identificador nuevo a los objetos que aparecieron. Para la asociación de los identificadores se considera el primer parámetro, la distancia máxima que puede haber entre la última detección del centroide, si es que se supera la distancia máxima entre los nuevos centroides y los antiguos se asigna un nuevo identificador, debe ser un nuevo objeto. Cuando no supera la distancia máxima entonces se asocia el centroide antiguo con el nuevo que tenga la distancia euclidiana más corta al mismo. El segundo parámetro para la asociación es la máxima desaparición permitida del objeto entre marcos antes de eliminarlo, es decir, puede que no sea detectado entre marcos por la iluminación u otros factores, en este caso se espera un número máximo de marcos que se determina por ciertos factores que se explicaran para verificar que ese objeto ha salido de la vista de la cámara. El penúltimo paso es actualizar las ubicaciones de los centroides a las más recientes. En la figura 6 se puede observar el comportamiento general del método mediante un gráfico. Los círculos de color rojo con identificadores 1 y 2 fueron los objetos detectados en el primer marco

del video. En el segundo marco se detectan 3 objetos cuyos centroides tienen color amarillo en círculos rellenos. El algoritmo en el segundo marco realiza el cálculo de distancias euclidianas y asocia los círculos con los identificadores más cercanos, y al más distante le asigna un identificador nuevo.

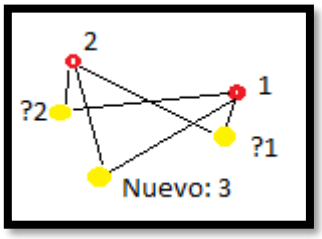


Figura 6: Ejemplo del comportamiento al aplicar el método del centroide.

Para entender los valores establecidos para cada parámetro es importante primero comprender el valor "C" seleccionado, es decir cada cuantos marcos se realiza una detección. Esta configuración y explicación se detallará en el capítulo de pruebas y resultados. El último paso consta en eliminar los centroides antiguos que no tienen asociación a un centroide nuevo y que ya ha cumplido el parámetro de máxima desaparición permitida. En este caso para no perder el conteo de los objetos históricos se maneja un atributo de valor booleano que le permite al algoritmo saber que ese objeto ya no está activo para las futuras acciones. De esta manera se puede llevar una cuenta de cuantos objetos hubo durante el transcurso del video.

- Método de correlación:

DLIB es una librería que contiene métodos que pueden emplear *deep learning* y otros tipos de métodos tradicionales para realizar análisis sobre imágenes/videos [33]. La alternativa de *DLIB* utilizada para el proyecto fue la del algoritmo de seguimiento mediante correlación que implementa la solución especificada en [32]. Esta es una solución robusta que hace el rastreo del objeto en cada marco y es adaptable a cambios de escala en el objeto rastreado entre marcos.

- Problemas relacionados con el rastreo:

En [32] establece: "A pesar del progreso significativo…el problema sigue siendo difícil debido a factores como oclusión parcial, deformación, desenfoque de movimiento, movimiento rápido, variación de iluminación, desorden de fondo y variaciones de escala" (véase figura 7). Estos problemas impiden que los métodos de rastreo existentes funcionen con un 100% de éxito. Sin

embargo, aunque existen múltiples problemas asociados con el rastreo de objetos, estos pueden ser resueltos mediante el hardware. Es decir, la resolución de la cámara, el enfoque, el ángulo de grabación, la iluminación, altura y distancia de los objetos, etc. También se puede observar que, en investigaciones similares, pero con métodos distintos se han presentado problemas como por ejemplo oclusión en [2]. Se realizaron pruebas para identificar cuáles de los problemas comentados se presentan en los métodos propuestos. Cada uno es robusto con respecto a ciertos factores; el método de correlación por ejemplo es robusto a cambios de escala. En el del centroide también se esperan resultados robustos con respecto a ciertos problemas, la detección del objeto lo realiza el modelo basado en redes neuronales. El mismo usualmente está preparado para trabajar con cambios de iluminación y enfoque. Sin embargo, el método del centroide es vulnerable a factores como la oclusión, dependiendo del ángulo del video.

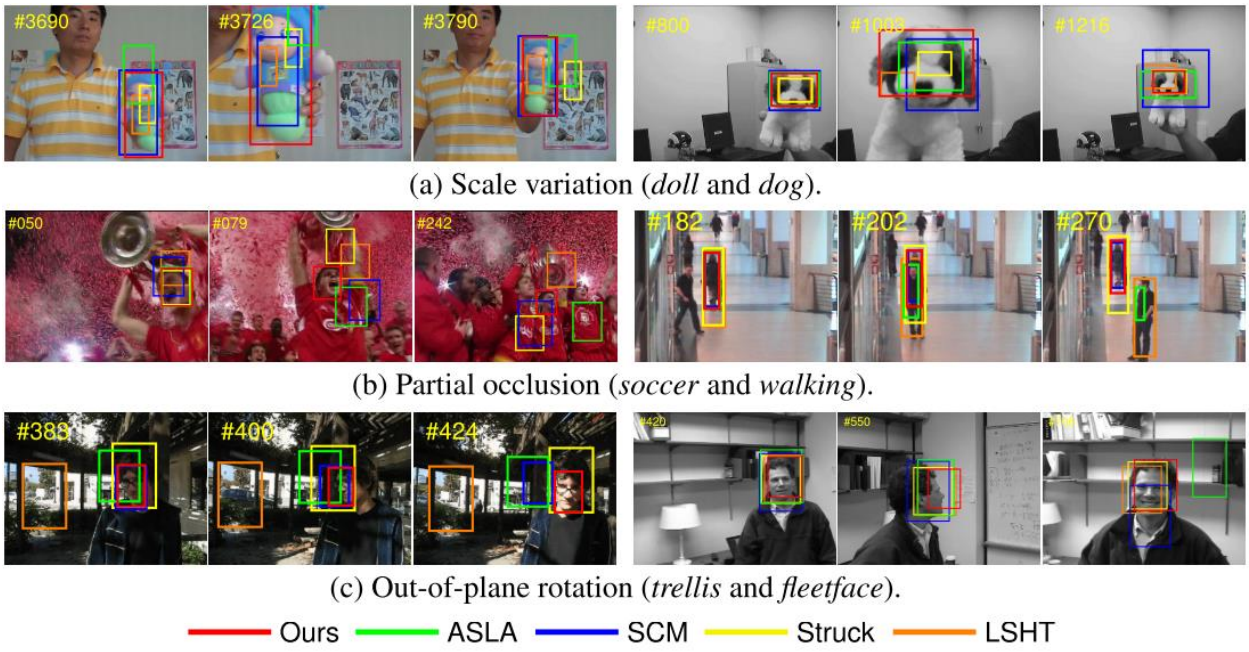


Figura 7: Una muestra de los problemas mencionados y comportamiento del método de correlación en cada caso. (a) variación de escala, (b) oclusión parcial, (c) rotación. Se presenta una comparación del método de correlación el cual tiene las cajas de color rojo, con otros métodos de rastreo. Para más detalles y la fuente de esta imagen revisar [32].

2.3.5 *OpenVINO* y *Neural Compute Stick 2*

OpenVINO es un set de herramientas de Intel que provee librerías optimizadas para la ejecución e inferencia de redes neuronales. Su título original en inglés es *Open Visual Inference and Network Optimization* (*OpenVINO*). Aquí podemos encontrar, entre otras, la librería optimizada de

OpenCV con la *API* de *OpenVX* para la visión por computador tradicional. También cuenta con una herramienta que permite la optimización de modelos para algunas redes neuronales donde que transforma los modelos en una representación intermedia optimizada para luego ser aplicado en la aplicación del usuario. También es importante mencionar que *OpenVINO* soporta por defecto Python y C++. Cuenta con múltiples ejemplos de uso, por ejemplo, el procesamiento de múltiples flujos de video, el procesamiento asíncrono de video y más.

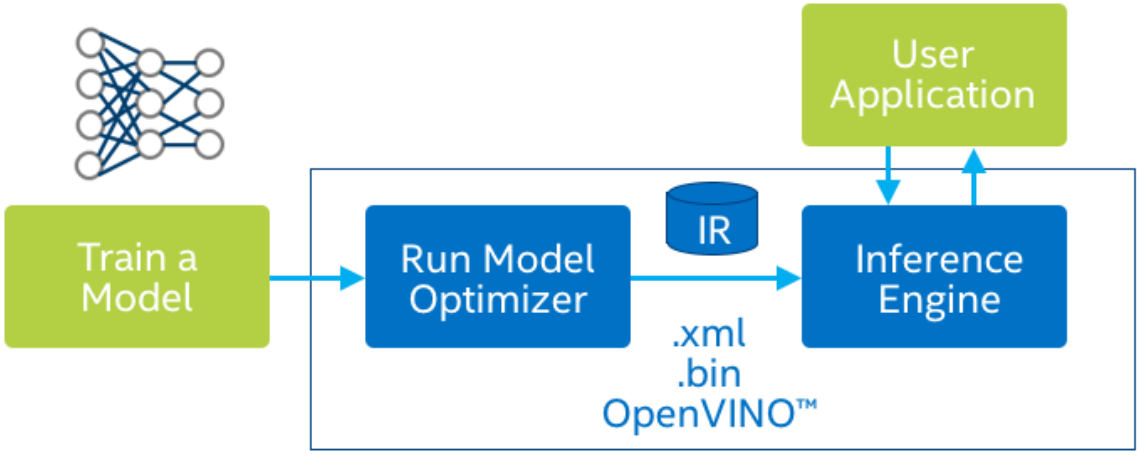


Figura 8: Flujo de optimización de modelos entrenados con redes neuronales mediante la representación intermedia de *OpenVINO*. Imagen obtenida de: [34].

Una función importante de *OpenVINO* es que permite la ejecución de programas en cualquier procesador de la marca Intel incluyendo al Intel *NCS 2* sin necesidad de cambiar el código para ser adaptado a las plataformas. Es por estos beneficios y optimizaciones ofertadas que se ha optado por hacer uso de *OpenVINO* para la implementación del sistema. El Intel *Neural Compute Stick 2*, es un dispositivo que permite la ejecución de redes neuronales orientadas hacia procesamiento visual en dispositivos embebidos. Esto gracias a que tiene un bajo consumo de energía y adicionalmente es portable al ser un dispositivo del tamaño de una memoria flash actual común que se interconecta mediante USB 3.0 o 2.0. El Intel *NCS 2*, cuenta con un procesador *Myriad X Visual Processing Unit* (*VPU*). Soporta *Tensorflow*, *Caffe*, *Apache MXNet*, y otros *frameworks* para redes neuronales. Gracias al uso de *OpenVINO* el *NCS 2* tiene compatibilidad con múltiples sistemas operativos por lo cual resulta atractivo para este tipo de proyectos. Es relevante mencionar que el uso de este dispositivo logra disminuir la carga del procesador del dispositivo embarcado en el cual se emplea, en nuestro caso el Raspberry Pi 3B+, permitiendo usarlo para otras funciones adicionales. Se puede encontrar más detalles técnicos en: [35].

2.3.6 Hardware y otros elementos:

- Hardware principal para solución

El Raspberry Pi 3B+ es un dispositivo embarcado popular en la actualidad por lo cual ha sido nuestra elección como dispositivo base de las pruebas que se realizaron. Adicionalmente se seleccionó la versión 3B+ porque fue el modelo más nuevo al momento de elaboración de este proyecto. El sistema operativo es basado en Linux facilitando así la instalación de los paquetes de software requeridos, para más especificaciones técnicas revisar [36]. Para la memoria se ha utilizado una MicroSSD SanDisk Ultra 32 GB y para el *VPU* un Intel *NCS 2*.

- Lenguaje de programación:

Python3: La razón o motivación para haber utilizado Python, un lenguaje de programación que permite trabajar de manera rápida e integrar sistemas efectivamente, es que tiene mucha acogida en la actualidad [37]. Adicionalmente cuenta con numerosas librerías que permiten la integración con las plataformas que se requieren para el proyecto, existe soporte continuo gracias a la amplia comunidad de usuarios de Python y también tiene compatibilidad con el Raspberry Pi 3B+. Sin embargo, es importante mencionar que si se utiliza un lenguaje de bajo nivel como C++ se obtendrían optimizaciones notables con respecto a la velocidad del procesamiento de los algoritmos.

Capítulo III

MÉTODOS Y EXPERIMENTOS

3.1 Detalles del sistema y resumen general del diagrama del método y objetivos del proyecto.

El procedimiento seguido para realizar la clasificación y conteo de actores de movilidad obtenidos del flujo de video se detalla en la figura 9. Existen cuatro etapas o fases que conforman el proceso; Inicialmente la Captura de datos en la cual se obtiene el flujo de video mediante marcos continuos en forma de imagen. En esta etapa se implementa un método para saltar ciertos marcos que no van a ser procesados por el sistema de detección para incrementar la velocidad del proceso, sacrificando la precisión en el rastreo. Se procede a la segunda fase de detección y clasificación, aquí se pasa el marco a la red neuronal y se obtiene la clase del objeto en caso de haber y las coordenadas de este; cabe recalcar que, aunque el sistema es robusto y soporta cualquier ángulo del video, para esta fase se experimentó con videos que contaban con dos ángulos (con el objetivo de validar si ayuda en la problemática de rastreo mencionada en el capítulo previo). Fue necesario modificar ciertos parámetros (como por ejemplo zona de detección y línea de detección) para mejorar los resultados de conteo dependiendo del ángulo del video. En la tercera fase, inicialmente se intenta realizar el rastreo en todos los marcos y la detección como se indicó previamente cada C marcos, sin embargo, esto no fue posible (el motivo se detallará en los resultados). El objetivo del rastreo es poder asignar un identificador a los objetos detectados. De esta forma cada que ingresa un nuevo marco se verifica que no se cuente dos veces el mismo objeto. La última fase se basa en acumular en memoria los resultados obtenidos, en este caso las clases y el número de objetos de cada una. De esta forma se puede realizar el conteo de actores de movilidad identificados durante el transcurso del tiempo.

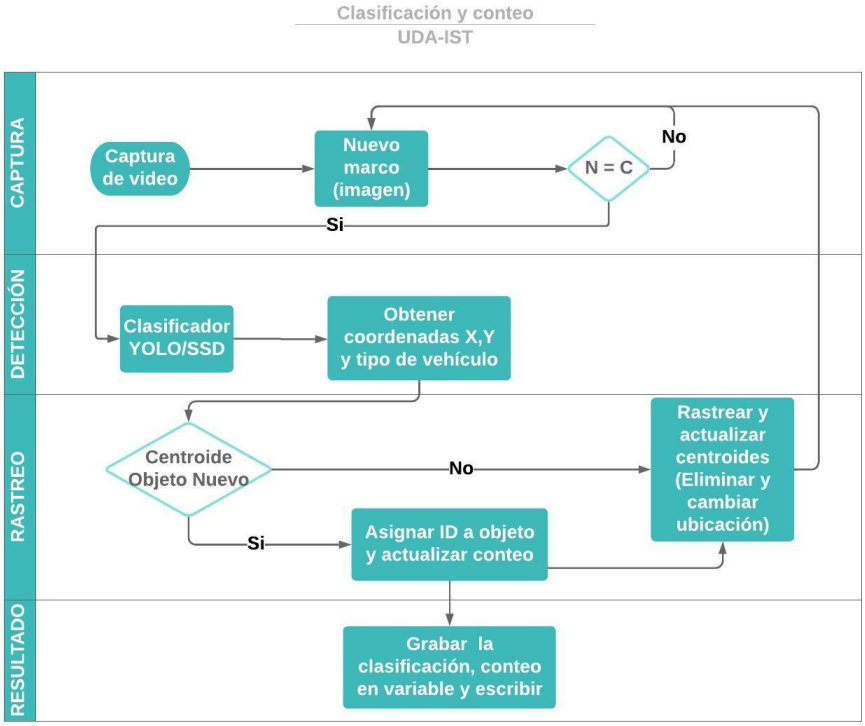


Figura 9: Diagrama de flujo del sistema propuesto.

- Etapa de Captura:

Se basa en obtener el flujo o la captura del video. Para este proyecto se han utilizado archivos de video mp4 obtenidos desde una cámara de un Huawei P Smart 2019. Es posible hacer uso de flujos de videos en tiempo real mediante una conexión USB (desde cualquier cámara que tenga esta capacidad para USB) hacia el Raspberry Pi 3B+. La proyección es que se utilice un flujo de una cámara en tiempo real. La ventaja de utilizar el flujo de video en tiempo real es que se pueden contar los actores de movilidad sin el requerimiento de almacenar en memoria la grabación de video.

Existen múltiples módulos implementados para obtener y procesar el flujo de video. Entre estos por ejemplo la librería de OpenCV tiene un módulo de lectura de video llamado "*VideoCapture*". En este no se alcanza el máximo nivel de eficiencia. El funcionamiento del módulo se basa en procesar la lectura y decodificación secuencialmente; es decir el hilo principal de procesamiento se bloquea para procesar la lectura y luego la decodificación. Sería más eficiente liberar el hilo principal de procesamiento y correr dos hilos, uno que lea y otro que decodifique. Este funcionamiento fue propuesto en la librería de herramientas para manejo de video "*imutils*" [38].

Otra alternativa para la captura del video eficientemente es la lectura asíncrona de marcos, pero esto tiene sus propias complicaciones que se mencionaran en la sección de resultados.

Luego de seleccionar el método de lectura y decodificación se procede con el preprocesamiento requerido por los distintos métodos evaluados de detección y clasificación. Los métodos requieren de una entrada con ciertas características específicas. En general para *YOLO v3, Tiny YOLO v3* y *SSD* se realiza los siguientes pasos: redimensionar el tamaño del video y transponer internamente la matriz de cada imagen de HWC a CHW (por defecto la red requiere de un ingreso en CHW por lo cual se requiere realizar este cambio). Las siglas hacen referencia a: H=*height*(altura), W=*width*(ancho), C=*color channel*(canal de color que puede estar en RGB o BGR lo cual a su vez significa rojo(R), verde(G), azul(B). Otro cambio realizado para optimizar los resultados es la redimensión de los videos manualmente con distintos tamaños para conducir múltiples experimentos y observar cual combinación brinda mejores resultados. Para garantizar el funcionamiento óptimo de la red neuronal el marco de entrada del video debe ser proporcional y cuadrado. En los casos donde los videos no eran cuadrados hubo que agregar bordes para lograr que se encuentre con una proporción de 1:1. De esta manera se evita el problema de deformación de imagen. Esta redimensión se realizó con el programa "VLC", y se notó perdida de la calidad de imagen. Esto puede causar efectos adversos en la detección y clasificación debido a que los filos del vehículo ahora contienen un nivel de difuminación, caso que no existía en el video original. Después, se analizó como lograr procesar el video en tiempo continuo. El factor de salto de marco el cual se detalla en las pruebas y resultados realizados.

- Etapa de Detección y Clasificación.

En esta etapa se evalúa cada método en detalle. Se pasa cada marco obtenido de la etapa previa hacia la red neuronal, se obtiene la clase de los objetos detectados y las coordenadas de estos. En la sección de pruebas y resultados, es posible observar algunos ejemplos de los métodos probados. (véase tabla 7).

- Etapa de Rastreo.

La razón por la cual se tiene que realizar el rastreo es para lograr contar los actores de movilidad. Esto es parte importante del proyecto y permite contabilizar el número de vehículos con sus respectivas clases. Una vez decidida la metodología, es decir de realizar el conteo mediante el rastreo con el método del centroide o correlación se procede a analizar formas para mejorar y hacer

más robusto el método de conteo. Para esto se ha ingeniado agregar una zona de conteo o línea de conteo dependiendo del video, el funcionamiento de estas se detallará en la sección correspondiente de las pruebas y resultados obtenidos.

- Etapa de Resultado.

En esta fase se imprimen en pantalla los resultados de conteo por cada clase de los actores de movilidad. Es importante analizar los resultados desde múltiples perspectivas. Puede haber falsos positivos, falsos negativos, es posible que funcione mejor de un ángulo determinado y puede haber otros factores que deben ser analizados minuciosamente. Cada detalle puede influenciar en la eficiencia de conteo del flujo de entrada. Por este motivo se deben usar videos cortos para poder observar el comportamiento de los resultados ante los hechos reales considerados por un humano. Con los datos de conteo se procede a analizar los datos comparando con los hechos reales y se obtienen conclusiones, también se puede generar una matriz de confusión y finalmente observar el consumo de los recursos de los dispositivos utilizados.

3.2 Materiales:

3.2.1 Datos utilizados:

Los flujos de video serán mencionados como los datos utilizados para esta investigación. Los videos fueron tomados en la ciudad de Cuenca, Ecuador. La ubicación exacta pertenece a la Panamericana intersección Av. Felipe II. Estos videos tienen una duración de aproximadamente 1 minuto cada uno y se han contabilizado y clasificado por un agente humano para poder comparar el nivel de precisión obtenido con el conteo obtenido del sistema. La motivación de los videos cortos tiene el propósito de hacer un muestreo que facilite la comparación del conteo humano con los resultados obtenidos del sistema en búsqueda de identificar distintas fallas. Para la comparación es necesario revisar dos videos paralelamente. Este proceso lo debe hacer un humano (no se puede automatizar dicho proceso, la maquina no sabe si es que está bien o mal clasificado el resultado). En caso de no estar correcta los resultados se ven afectados y se debe documentar las falencias del sistema. Es necesario hacer énfasis en que esto es una muestra para tener una idea de cómo opera el sistema ante las distintas circunstancias. Los valores de las métricas como mAP ya se encuentran documentados por cada método utilizado.

Los videos tienen un valor promedio de 30.40 FPS. En la tabla 13 (Anexo 1) se identifican los valores exactos, aunque resulta un poco repetitiva con respecto a los FPS y duración porque todos tienen el mismo tiempo de duración. Para las pruebas iniciales se utilizará el video "FRENTE1" como base y en distintas resoluciones. En total el número de videos distintos que se probaron son 5 los cuales se pueden visualizar en la tabla 4 con el respectivo número de vehículos identificados en el conteo por un humano.

Video	Carros	Motos	Camiones	Buses	Personas	Total
FRENTE1	22	0	0	1	1	24
FRENTE2	20	1	0	0	0	21
FRENTE3	25	0	1	0	1	27
FRENTE4	9	0	1	0	0	10
LATERAL1	14	0	1	1	0	16

Tabla 4: Videos y conteo obtenido por un humano por cada video.

3.2.2 Métricas

Existe una variedad de métricas que se pueden considerar, por ejemplo precisión, exhaustividad y exactitud [9]. Las mismas ayudan a evaluar los modelos obtenidos conjuntamente dependiendo del dataset utilizado. Cada modelo que se ha presentado cuenta con sus métricas respectivas como por ejemplo la métrica de mAP. Por este motivo no se requiere hacer un análisis profundo así que se considera la precisión, exhaustividad y exactitud con respecto a los videos propuestos. Esto permite analizar cómo se comporta el sistema con las muestras existentes y si es que los resultados obtenidos sirven para poder hacer un conteo continuo. Adicionalmente se agregan dos métricas para el consumo de recursos del dispositivo embarcado: consumo en porcentaje de memoria RAM y CPU. Las fórmulas para cada métrica se presentan a continuación. El significado de las siglas es: Verdaderos positivos (T.P.), Falsos Positivos (F.P.), Verdaderos Negativos (T.N.) y Falsos Negativos (F.N.).

Precisión: $\frac{T.P.}{T.P.+F.P.}$

Exhaustividad: $\frac{T.P.}{T.P.+F.N.}$

Exactitud: $\frac{T.P.+T.N.}{T.P.+F.P.+T.N.+F.N.}$

RAM y procesador (del *Raspberry Pi 3B+*):

Consumo de RAM en porcentaje al ejecutar la clasificación y conteo de un video. Consumo de procesador en porcentaje al ejecutar la clasificación y conteo de un video.

3.3 Pruebas:

Para el proyecto se efectuaron 7 pruebas distintas. La motivación para las mismas se basa en los hechos mencionados en el estado del arte y marco teórico. Para las pruebas se sigue un orden similar al funcionamiento del sistema para una comprensión correcta de la razón de estas.

3.3.1 Captura:

3.3.1.1 Evaluar módulos de lectura:

En esta prueba se verifica cual es el mejor método para realizar la lectura de datos, es decir mayor velocidad de lectura. Las evaluaciones se medirán en FPS. Como se mencionó en el detalle resumen general del sistema se han considerado 3 métodos distintos para evaluar: la clase de *VideoCapture*, *imutils* y el método asíncrono. Para las pruebas se usa un video de 416x416px, el motivo se indicará en las pruebas de detección. Se realizaron dos procedimientos, uno ejecutar el procesamiento asíncrono de marcos con *YOLO v3* con el objetivo de observar si es que existe una diferencia entre procesar síncrona y asíncronamente. El segundo procedimiento fue evaluar *VideoCapture* y la librería de *imutils* con el modelo *SSD*. Los modelos utilizados son referenciales, con cualquier modelo se puede visualizar la influencia e impacto del método de lectura utilizado. Lo importante es analizar si es que existen diferencias entre los métodos de lectura. El funcionamiento del método asíncrono es el siguiente: Se lee 1 marco inicialmente el mismo que se le pasa al Intel *NCS 2* para ser procesado por *YOLO v3* y sin esperar la respuesta del *NCS 2* se empieza la lectura del próximo marco asíncronamente; una vez que se hace la lectura del segundo marco se envía al dispositivo *NCS 2* y se revisa si es que el marco inicial u algún otro previo en la cola se encuentra listo con resultados. Al encontrarse listo se imprime la respuesta y se procede a la lectura del siguiente marco; cuando no está listo se procede a la lectura del siguiente marco y el proceso se repite. El dispositivo *NCS 2* puede tener un buffer donde que se encolan las peticiones y respuestas solicitadas.

3.3.2 Detección y clasificación:

3.3.2.1 Evaluación de modelos *SSD, YOLO v3* y *Tiny YOLO v3* con imágenes*:*

El propósito de esta prueba es analizar el comportamiento de los modelos ante distintos escenarios. Se toma en cuenta varios factores: resolución, detección de objetos en distintos ángulos (frontal y lateral) y clasificación de objetos en varias imágenes.

3.3.2.2 Evaluación de velocidad de procesamiento de los modelos en video:

Esta prueba se realiza para verificar cuanto tiempo tarda cada método en procesarse en el dispositivo embarcado e identificar la factibilidad de cada uno. Se evalúa 1 video con distintas resoluciones y se verifica la velocidad de procesamiento de cada método.

3.3.3 Rastreo:

3.3.3.1 Evaluación de los parámetros de rastreo:

El objetivo de este método es identificar cuáles son los parámetros de rastreo que brindan mejores resultados para hacer el seguimiento con el método del centroide. Los parámetros para evaluar son: máxima desaparición y distancia. Esta evaluación permitirá hacer uso de los mejores parámetros para hacer una comparación con el método de correlación.

3.3.3.2 Evaluación de modelos de rastreo:

Como se mencionó en el estado de arte, se evalúan dos modelos concretamente, el de correlación mediante la implementación de *DLIB* y el del centroide mediante la implementación de *imutils*. El objetivo es identificar cual modelo brinda los mejores resultados para usarse en el sistema. Se consideran dos valores importantes para esta evaluación: el conteo de objetos y el valor de FPS alcanzado con cada método.

3.3.3.3 Evaluación de zona y línea de detección:

Para mejorar el conteo y eliminar ciertos factores que afectan en los videos como la oclusión y variación de escala. Se han propuesto dos metodologías que permiten hacer el conteo sobre los métodos existentes como por ejemplo el método del centroide. Esto se ha propuesto con la integración de una zona en la que se hace el conteo de los vehículos y una línea que verifica la

dirección del vehículo y si va en la dirección deseada realiza el conteo. Cada método (zona de conteo y línea de detección) se ha evaluado independientemente y en dos tipos de video frontal y lateral con la espera de verificar si es que existe una mejora en el conteo de los objetos. También se presentan los resultados de conteo con el sistema elaborado de los videos aplicando todas las recomendaciones obtenidas a lo largo de las pruebas realizadas.

3.3.3.4 Evaluación de uso de recursos:

En esta prueba se identifica el consumo de recursos específicamente memoria RAM y CPU del dispositivo en distintos escenarios: Con *NCS 1, NCS 2* y sin la ayuda de un dispositivo con *VPU*. El objetivo de esta evaluación es observar cuantos recursos del dispositivo se consumen con *SSD* e identificar si es que efectivamente se presenta una reducción en el uso de recursos con el dispositivo *NCS 2*.

Capítulo IV

RESULTADOS

Los resultados obtenidos estarán en el mismo orden de las pruebas realizadas:

4.1. Captura:

4.1.1 Evaluación de módulos de lectura:

El método asíncrono tiene una complicación, este no funciona de forma secuencial por lo cual imposibilita el conteo de los objetos. Los métodos propuestos para el rastreo requieren de un ingreso secuencial para poder hacer el seguimiento de un objeto. Sin embargo, se realizó una prueba con el método asíncrono para comprobar mejoras en la velocidad. Esta consistió en procesar videos de 416x416 con el modelo *YOLO v3* y observar si es que había diferencia en la velocidad del procesamiento. Como se visualiza en la tabla 5 existe una mejora notable al procesar el modelo de forma asíncrona. En esta prueba se demuestra que existió una mejora aproximada del 33% en el número de FPS. Este valor podría variar dependiendo del video utilizado, pero en todos los casos existe una mejora notoria al realizar el proceso asíncrono. La razón de variación se debe a que con una menor resolución el proceso de lectura es más veloz, al igual que el proceso de detección y clasificación en el dispositivo *NCS 2*.

(Videos de 416x416px)	
Método asíncrono con *YOLO v3* (Utilizando *imutils*)	Método síncrono con *YOLO v3* (Utilizando *imutils*)
Promedio de FPS: 1.19	Promedio de FPS: 0.79

Tabla 5: Comparación entre método asíncrono y síncrono con *YOLO v3*.

La segunda parte se basó en realizar una comparación entre *VideoCapture* y la opción de *imutils*. En este caso ambas opciones alimentan los marcos al dispositivo *NCS 2* de forma secuencial, el modelo cargado para esta prueba es *SSD*. La diferencia entre *VideoCapture* e *imutils* como se ha mencionado previamente es que *imutils* utiliza dos hilos mientras *VideoCapture* utiliza uno. En la tabla 6 se observa que el uso de *imutils* presenta una mejora notable en el número de FPS promedio de los videos que representa aproximadamente el 8% de incremento de velocidad en la lectura de los marcos.

(Videos de 416x416px)	
Uso de *VideoCapture* de *OpenCV* en *SSD* (Utilizando método síncrono)	Uso de *imutils* en *SSD* (Utilizando método síncrono)
Promedio de FPS: 8.5	Promedio de FPS: 9.23

Tabla 6: Comparación entre *VideoCapture* e *imutils*.

En conclusión, la opción que proporcionó los mejores resultados con respecto a alcanzar a procesar el mayor número de FPS fue la de procesamiento asíncrono con un incremento sustancial en la velocidad de procesamiento de los marcos, sin embargo, no se pudo utilizar, debido al problema del requerimiento del rastreo de entrada de marcos en forma secuencial. Por este motivo se utilizó el método del Dr. Adrian Rosebrock para realizar la lectura y decodificación del flujo de video. Sin embargo, es importante mencionar que durante los experimentos realizados con la librería "*imutils*" se obtuvo una cantidad de FPS variante en cada ejecución; a diferencia de "*VideoCapture*" con el cual se obtiene un numero constante de FPS en cada ejecución. Esto se debe a la naturaleza del método al ejecutar dos hilos separados para procesar el flujo del video.

4.2. Detección y clasificación:

4.2.1 Evaluación de modelos *SSD, YOLO v3* y *Tiny YOLO v3* con imágenes*:*

Para esta prueba se han analizado múltiples imágenes tomadas de los videos. Lo que evalúa esta prueba es el nivel de detección y comportamiento de los modelos en cuanto a distintos escenarios como por ejemplo cómo reaccionan ante diferentes tipos de vehículo. El comportamiento de los modelos varía y es relevante sostener que no todos tienen las mismas clases y tampoco son robustos a todos los escenarios como cuando se presentan cambios de escala de la cual fueron inicialmente entrenados. Uno de los ejemplos con respecto a las clases es que en el modelo *SSD* no existe la clase "Camión" o "*Truck*" por lo cual no se puede hacer una comparativa exacta con nuestro conteo manual. La decisión que se tomó con respecto a este escenario es hacer una estimación/asignar un margen de error +/- el número de vehículos de clase "Camión". Como se observa en las siguientes imágenes en algunos casos detecta el camión como "Carro" y en otros casos no lo detecta. En la figura 10 se observan dos imágenes del mismo camión en forma lateral; a la izquierda se visualiza la detección y clasificación de *YOLO v3* y a la derecha se observa que *SSD* no lo detecta, tampoco lo identifica como carro o bus.

En la figura 11 se puede visualizar nuevamente un camión pequeño que es detectado como carro por *SSD*.

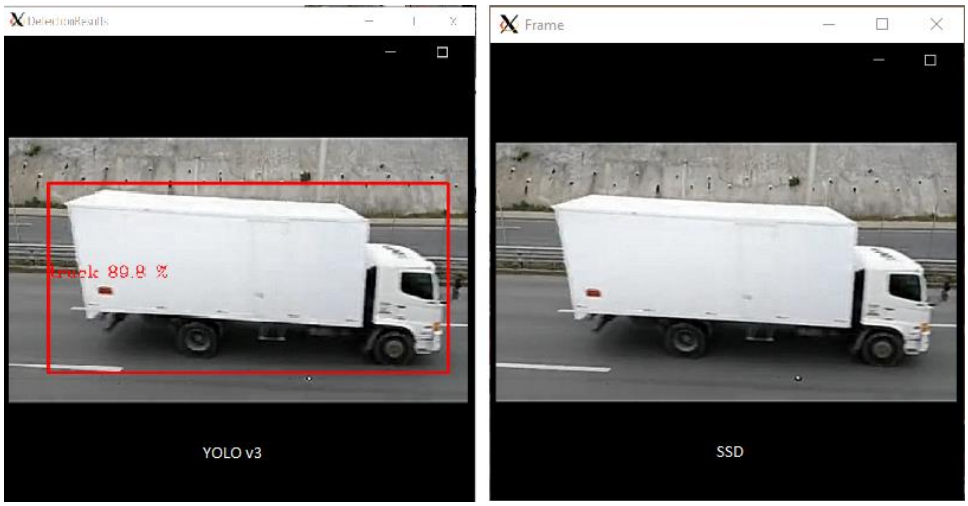


Figura 10: Imagen de la izquierda aplicada *YOLO v3*, en la derecha *SSD* sin detectar al vehículo.

Figura 11: Imagen aplicando *SSD* donde que se detecta al camión como un carro.

Con respecto a los cambios de escala en la tabla 7 se puede observar que *SSD* no detecta algunos objetos que son pequeños, a diferencia de *YOLO v3* que detecta los objetos grandes y pequeños. La explicación correspondiente es que en *YOLO v3* existe una estructura que le permite analizar tres escalas distintas de una imagen detectando y clasificando así objetos pequeños y grandes. Es fundamental mencionar que, aunque *YOLO v3* tenga registrado un valor de mAP inferior al de *SSD*, en este caso con nuestras clases presenta mejores resultados en la detección y clasificación. El valor menor de mAP se debe a que al momento de determinar el mAP de *YOLO v3* se analizan más clases de objetos que *SSD*.

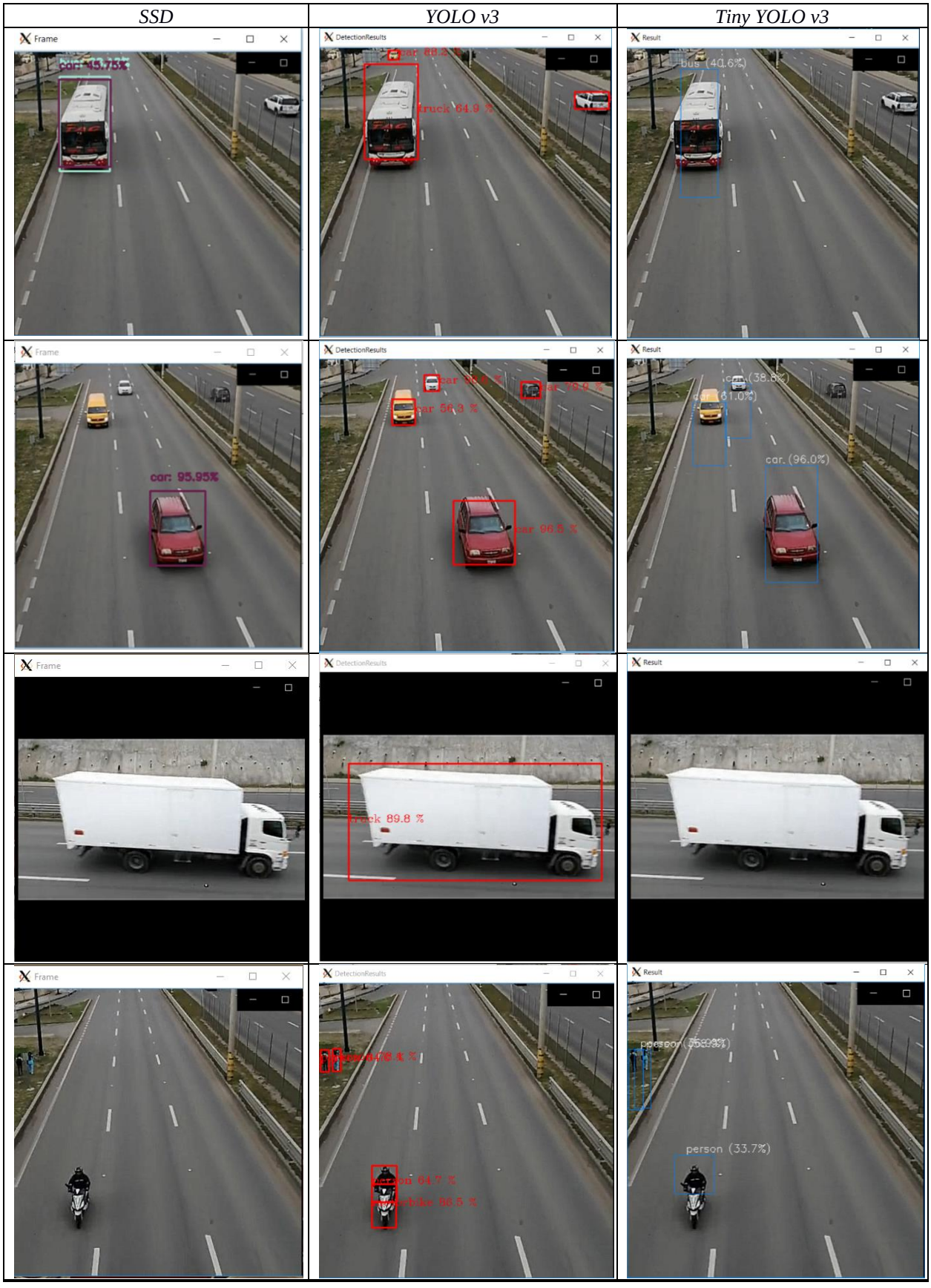


Tabla 7: Métodos neuronales aplicados en distintos escenarios.

En las pruebas realizadas con respecto a esta etapa se observa y concluye que *YOLO v3* detecta todos los objetos presentados en las pruebas, mientras que *SSD* no detecta objetos en múltiples escenarios, sin embargo, *YOLO v3* puede presentar un problema en cuanto al número de FPS lo cual se analizara en la próxima prueba.

4.2.2 Evaluación de velocidad de procesamiento de los modelos en video:

Para esta prueba se ha evaluado el tiempo de procesar de cada video con y los marcos por segundo alcanzados con los distintos modelos. El video que se utiliza es el FRENTE1 con distintas resoluciones lo cual también representa distintos valores de BITRATE. Para la prueba se evaluó el video con la lectura mediante *imutils* y luego procesando con el modelo neuronal respectivo. Los resultados se pueden visualizar en la tabla 8. También se identificó en cada caso cuantos marcos se deben saltar es decir el "*Skip frame*" (o variable "*C*": véase figura 9) mediante un cálculo. Si se observan los resultados del experimento, en la tabla 8, resulta imposible procesar los videos en tiempo real sin saltarse marcos. La fórmula para calcular el "*Skip frame*" es simple:

$$\frac{FPS\ del\ video}{FPS\ alcanzado}$$

Resolución	**Bitrate (KBPS)**	**Modelo**	**FPS procesar**	**Tiempo procesar(s)**	**"*Skip frame*" requerido para tiempo real**
620X620	592	SSD	8.260	0223.23	03.69
500X500	358	SSD	8.780	0209.26	03.47
400x400	232	SSD	9.400	0195.52	03.24
300x300	146	SSD	9.190	0200.01	03.31
200X200	077	SSD	9.630	0190.84	03.16
620X620	592	YOLO v3	0.636	2900.92	47.92
500X500	358	YOLO v3	0.670	2745.50	45.49
400x400	232	YOLO v3	0.716	2565.00	42.56
300x300	146	YOLO v3	0.750	2450.80	40.64
200X200	077	YOLO v3	0.779	2359.60	39.12
620X620	592	Tiny YOLO v3	1.537	1200.40	19.83
500X500	358	Tiny YOLO v3	1.829	1005.60	16.66
400x400	232	Tiny YOLO v3	2.240	0821.20	13.60
300x300	146	Tiny YOLO v3	2.586	0711.20	11.78
200X200	077	Tiny YOLO v3	3.050	0602.80	09.99

Tabla 8: FPS, tiempo de procesar y "*Skip frame"* requerido para tiempo real en videos de acuerdo a los modelos establecidos.

Por lo tanto, en la tabla 8 se muestra que en el caso de *SSD* la mayor cantidad de FPS alcanzadas es con el video de 400x400px y 200x200px alcanzando 9.4 y 9.63 FPS respectivamente; sin embargo, el momento de hacer una inspección visual de las detecciones en cada video, se observó, que existían errores en el video de 200px. La resolución de 400x400px se seleccionó como referencia para los siguientes experimentos. Es importante notar que también se consideró *YOLO v3* y *Tiny YOLO v3* para esta decisión, pero después de un análisis visual se determina que también los resultados de detección y clasificación en estos modelos es más preciso con el video de resolución 400x400px. En base a esta resolución se obtiene los valores para cada modelo del parámetro de "*Skip frame*" requerido. Para *YOLO v3* se obtiene un valor de salto de 42.56, considerando que el video cuenta con un flujo de 30FPS, el procesamiento real es menor a 1 marco por segundo. Por este motivo resulta imposible utilizar *YOLO v3*; con dicho número de salto de marco no se puede realizar el rastreo del objeto en tiempo continuo. El mismo caso sucede con *Tiny YOLO v3*, con un valor "*Skip frame*" de 13.6. El saltarse 14 marcos entre detecciones introduce errores en el rastreo debido a que un vehículo puede pasar en el tramo de marcos saltados y no se detecta y tampoco se cuenta. En el caso de *SSD*, el valor es 3.69 por lo cual el parámetro de salto de marco colocado es 4, permitiendo procesar un flujo de video con estas características en tiempo real. (Para implementar el salto de marco en el programa se agregó una condición para ejecutar la detección y esto se alcanzó mediante: un contador simple "*skipfrm*" que cuenta cada marco leído del flujo del video y otra variable "*paramSkip*" que es el valor "*Skip frame*". Para verificar cuando se cumple la condición se utilizó la función de modulo; cada que se obtiene un módulo de 0 entre el número de marcos (*paramSkip*) y el valor "*Skip frame*" se hace una detección: "*skipfrm* % *paramSkip* == 0"). En la figura 12 se puede observar la diferencia gráfica en tiempo de procesamiento en segundos según cada modelo.

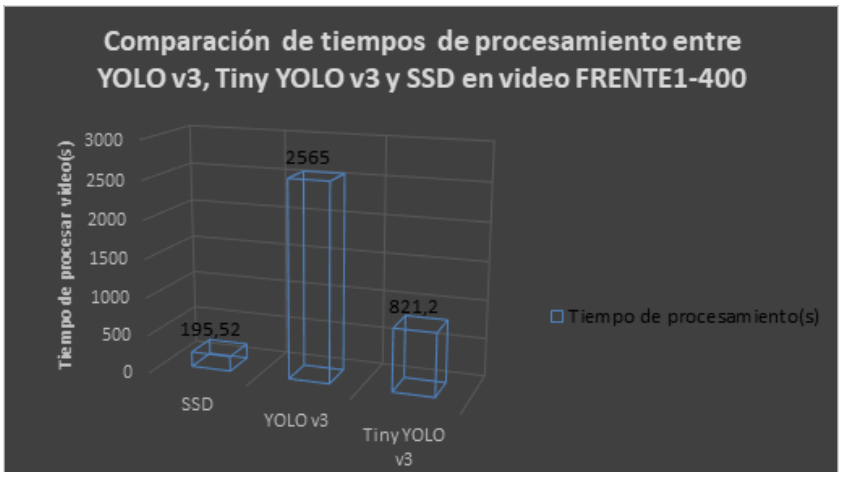


Figura 12: Diferencias de tiempo de procesamiento entre *SSD*, *YOLO v3* y *Tiny YOLO v3*.

Es importante hacer énfasis en que después de haber realizado las pruebas en la fase de detección el único método que resulta factible para poder realizar todas las etapas es *SSD*. La razón es que, aunque *YOLO v3* brinda mejor precisión, consume más recursos elevando el tiempo de detección y clasificación impidiendo el proceso en tiempo real o continuo. Por este motivo *YOLO v3* y *Tiny YOLO v3* se utilizó en la etapa de detección y clasificación, pero no en la etapa de rastreo, dado que resulta inadecuado para el objetivo del proyecto.

4.3. Rastreo

4.3.1 Evaluación de los parámetros de rastreo:

En esta prueba se utilizan e influyen datos que ya se han identificado en pruebas previas. Entre los datos importantes está el valor de "*Skip frame*" o salto de marco equivalente a 4 para *SSD*. Para la prueba se aplicaron distintos valores en los parámetros que se explicaran y se analizó los resultados obtenidos. El parámetro de máxima desaparición, que no debe confundirse con el valor de salto de marco, se determinó por lo siguiente:

La influencia del "*Skip frame*": Al realizar la detección cada 4 marcos todos los marcos intermedios son ignorados, es decir no contienen ningún tipo de objeto o centroide. Por este motivo el valor de desaparición máxima de un objeto debe ser un múltiplo de 4. Luego se realizaron las pruebas con los videos para verificar el valor optimo, los valores que se usaron para las pruebas fueron de 4 y 8. Con 12 o más el objeto siendo rastreado se pierde del video y genera detecciones incorrectas confundiendo nuevos vehículos con los anteriores.

El parámetro de máxima distancia entre objetos se determinó por el siguiente motivo:

Los valores se seleccionaron en base a prueba y error. Para obtener los valores base se observó el recorrido de un vehículo en un trayecto de 4 marcos. Se estimó en pixeles un valor aproximado que resultó en 80px, también se visualizó el tamaño de los vehículos que fue de un valor aproximado de 60px. Por esto las pruebas se hicieron sumando intervalos de 20px a partir de 80px hasta un máximo de 160px. Una vez ejecutadas las pruebas los valores con mejores resultados fueron de 80px y 100px dependiendo de la ubicación de la cámara. Esto hace que este parámetro sea dependiente del video y de la resolución. Por lo cual si se introduce un nuevo video es necesario analizar los valores óptimos en ese escenario para este parámetro.

Los resultados obtenidos se pueden observar en la tabla 10. Es evidente que las mejores configuraciones para los parámetros fueron de 80px a 100px en la máxima distancia y 8 en la

máxima desaparición permitida. Aunque hubo resultados compatibles con el conteo real, en algunos casos se presenció un problema como se mencionó en el capítulo 2, los identificadores se intercambiaron por la distancia establecida, esto se puede visualizar en la figura 13:

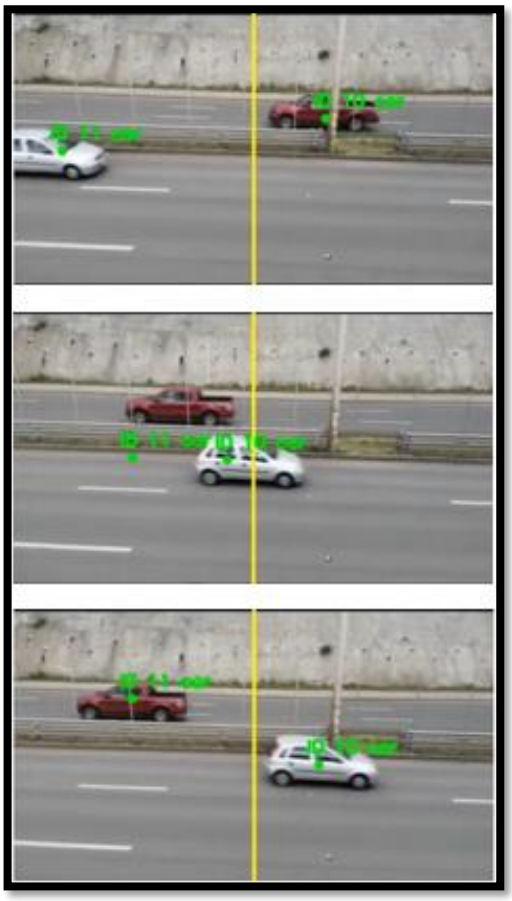


Figura 13: Ejemplo de intercambio de identificadores.

Figura14: Ejemplo de oclusión parcial en el video lateral.

También hubo un problema de oclusión parcial como se observa en la figura 14. Estos problemas no se presentaron en los videos de tipo frontal. Los problemas de oclusión tampoco se encuentran en los videos frontales, por este motivo se optó por analizar 4 videos frontales y uno en forma lateral.

4.3.2 Evaluación de modelos de rastreo:

Para esta prueba se analiza el método de correlación mediante la librería que lo implementa *DLIB* y se compara con el método del centroide que se encuentra en la librería de *imutils*. La forma en la que se realizó es mediante la evaluación del conteo obtenido y el valor de FPS alcanzado al implementar cada método. El método de correlación realizaba un seguimiento del objeto en todos los marcos a diferencia del centroide que se ejecuta y busca otros objetos de acuerdo con el salto de marco establecido el cual es 4. Esto se hace con el procesamiento de distintos videos, sin embargo, se toma como referencia el video lateral el cual es en el que se encuentran distintas complicaciones como oclusión, cambios de escala e intercambio de identificadores. Después de realizar las pruebas se obtuvieron las siguientes conclusiones:

- Al aplicar el seguimiento mediante correlación y sin el Intel *NCS 2* se observa que el seguimiento entre marcos es más veloz que la propia detección. Es decir, ejecutar únicamente con el procesador del Raspberry Pi 3B+ el seguimiento y el modelo neuronal se evidencia que el seguimiento se ejecuta más rápidamente que ejecutar el modelo de red neuronal.
- Cuando se aplica el algoritmo de detección con el *NCS 2* el tiempo de seguimiento entre marcos es igual o superior a realizar la detección en cada marco; esto se debe a que el *NCS 2* ayuda con el procesamiento de la detección siendo un dispositivo especializado para procesar imágenes con modelos de redes neuronales.
- En la figura 15 se observa la comparación de marcos por segundo alcanzados con el método de correlación y el método del centroide. Esto permite observar el comportamiento de ambos métodos en distintos escenarios de salto de marcos "*Skip frame*". Se observa que mientras más saltos se realizan, más se utiliza la correlación con la esperanza de rastrear los objetos en todos los marcos intermedios. Sin embargo, esto reduce la cantidad de FPS.
- El nivel de precisión en el conteo en el caso de aplicar el centroide es mejor que utilizar la correlación por lo cual se optó por aplicar el método del centroide; como se presenta en la gráfica el método de correlación es más lento en todos los escenarios. Según los datos obtenidos también es menos preciso que el centroide en cada prueba realizada. Es importante observar que con el centroide con un "*Skip frame"* de 0, es decir, al realizar detección en todos los marcos (el cual es el escenario óptimo de poder realizar) se obtiene un mejor rendimiento que al aplicar la correlación.

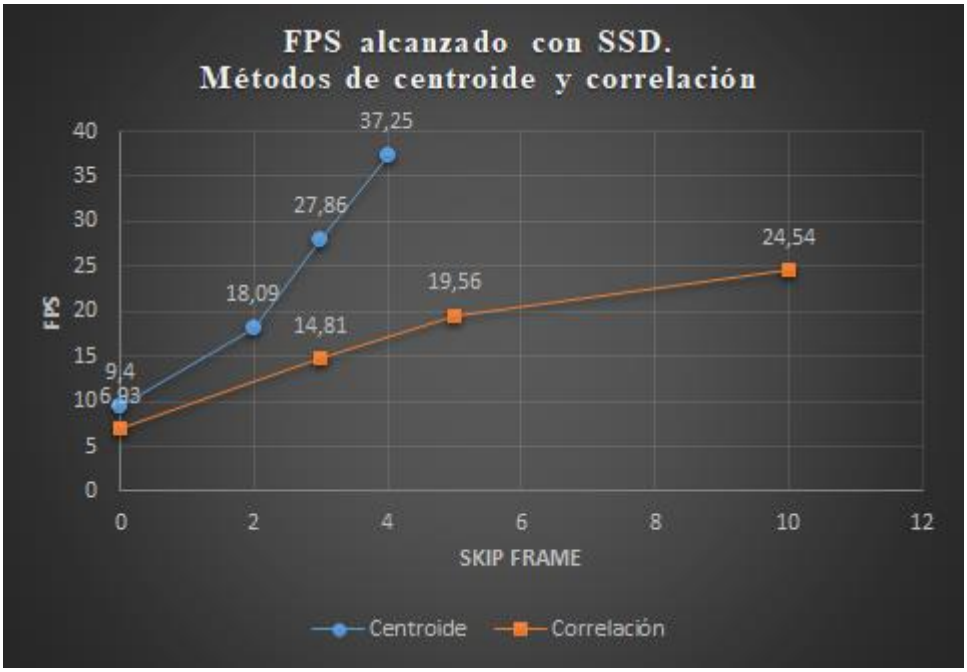


Figura 15: Comparación FPS alcanzado con el método del centroide (mediante *imutils*) y correlación (mediante *DLIB*). Se muestran los resultados obtenidos utilizando el Intel *NCS 2;* estos son resultados obtenidos con el video "Lateral1" en 400px.

4.3.3 Evaluación de zona y línea de detección:

En esta prueba final se obtiene el resultado de conteo de vehículos. Se probaron dos técnicas distintas con el propósito de controlar la oclusión y cambios de escala causados por vehículos distantes al punto de enfoque de la cámara. Previo a mencionar las dos técnicas consideradas es necesario mencionar que una cámara en una autopista apunta a múltiples carriles y dos direcciones existentes. Para el propósito del proyecto una forma de eliminar la variación en el cambio de escala es mediante el análisis de una sola dirección por cámara. Esto permite analizar las secciones correctamente enfocadas mostrando una imagen clara. Para lograrlo se debe determinar o segmentar la direccionalidad de los vehículos. La técnica de zona de conteo permite segmentar/aislar una zona para análisis, la línea de conteo permite determinar la direccionalidad de un vehículo. La propuesta para la línea de conteo es colocar una línea virtual en el centro de la imagen, cuando un vehículo cruza la línea de un lado hacia el otro se determina la dirección. La zona y línea de conteo se explicará detalladamente, se puede visualizar ejemplos en la tabla 9.

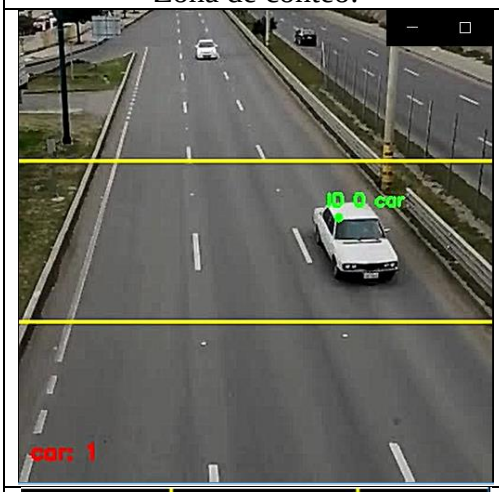
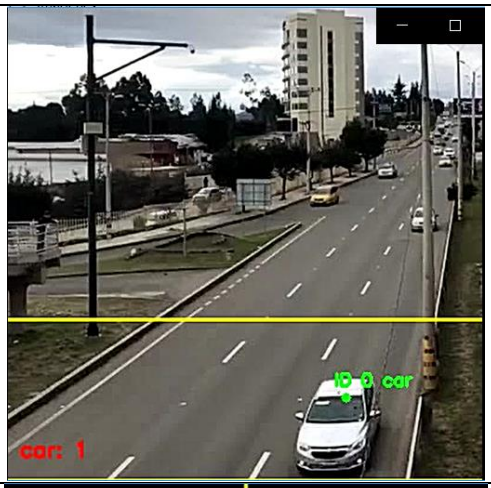
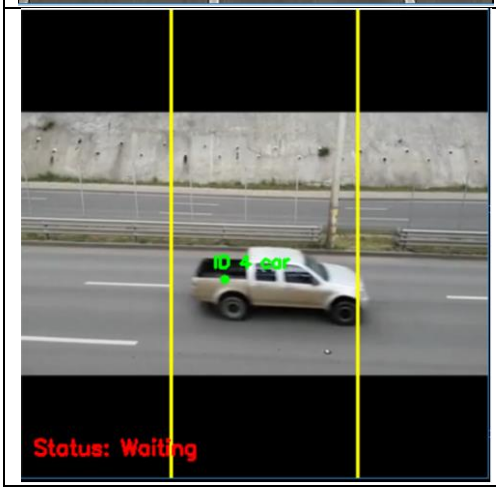
Zona de conteo:	Línea de conteo:
	
	

Tabla 9: Visualización de línea y zona de conteo en videos laterales y frontales.

Para el cruce de línea el algoritmo funciona de forma simple si es que el vehículo cruza de un lado hacia el otro se obtiene la dirección, si es la dirección que esperamos se agrega el vehículo con la clase correspondiente al conteo caso contrario no se agrega. Para la zona de conteo el funcionamiento es más complicado, una vez obtenidas las coordenadas del centroide se evalúa si es que esta dentro de las dos líneas establecidas. Para esto se aplica un método matemático, primero se obtienen dos ecuaciones de la recta y=mx+b. Luego se reemplaza el valor de x del centroide en cada ecuación, siempre se encentra un valor Y2 y Y1, con los cuales se verifica que esté entre las dos líneas. La forma de determinar esto para los videos frontales es con la condición si Y2>Y obtenida y Y obtenida>Y1. En el caso de videos laterales se verifica que el valor de X del centroide esté dentro del rango determinado. Según las pruebas realizadas en la tabla 10 se observa que, en los videos de tipo lateral, la línea de conteo funciona mejor que la zona de conteo. La razón es que con la zona de conteo no se encontró una forma de determinar la dirección, por lo cual se cuentan los carros de los dos carriles. Incluso si se agregase una tercera línea cortando por el centro de la

autopista para diferenciar los carriles con diferentes direcciones esto representaría otro problema: cuando un vehículo es alto en el caso del camión, y si está ubicado cerca del parterre central el centroide aparecería como si estuviese en el lado de la dirección opuesta provocando que no se cuente. Por esto no se puede agregar una tercera línea que divida el centro de la calle para diferenciar la direccionalidad. La línea de conteo por otro lado permite justamente determinar la direccionalidad y por lo cual es apta para este escenario. En los videos frontales utilizar la zona de conteo brinda mejores resultados que la línea de conteo.

FRONTAL1-400		**Incluyendo Zona de Conteo**					
Max. Dist.	**Max. Disap.**	**carros**	**motos**	**personas**	**camiones**	**buses**	**TOTAL**
80	4	28	0	3	0	1	32
80	***8***	***24***	***0***	***1***	***0***	***1***	***26***
100	4	28	0	3	0	1	32
100	8	24	0	1	0	1	26
FRONTAL1-400		**Con línea de conteo**					
Max. Dist.	**Max. Disap.**	**carros**	**motos**	**personas**	**camiones**	**buses**	**TOTAL**
80	4	18	0	0	0	1	19
80	8	18	0	0	0	1	19
100	4	18	0	0	0	1	19
100	8	19	0	0	0	1	20
LATERAL1-400		**Con zona de conteo**					
Max. Dist.	**Max. Disap.**	**carros**	**motos**	**personas**	**camiones**	**buses**	**TOTAL**
80	4	18	0	1	0	3	22
80	8	18	0	0	0	2	20
100	4	18	0	1	0	3	22
100	8	18	0	0	0	2	20
120	4	18	0	1	0	3	22
120	8	18	0	0	0	2	20
LATERAL1-400		**Con línea de conteo**					
Max. Dist.	**Max. Disap.**	**carros**	**motos**	**personas**	**camiones**	**buses**	**TOTAL**
80	4	10	0	0	0	0	10
80	8	6	0	0	0	0	6
100	4	13	0	0	0	0	13
100	***8***	***14***	***0***	***0***	***0***	***0***	***14***
120	4	14	0	0	0	0	14
120	8	14	0	0	0	0	14
160	4	13	0	0	0	0	13

Tabla 10: En la tabla se puede encontrar los resultados de pruebas realizadas para establecer la distancia y desaparición máxima.

Con los datos previos se procesaron los videos y se obtuvieron los resultados de la tabla 11. Para los videos frontales se utilizó la zona de conteo, para los laterales se utilizó línea de conteo. También se aplicó centroide para el rastreo y los parámetros de máxima distancia se colocó en 80

y máxima desaparición en 8. Se utilizó *SSD* porque como se determinó previamente *YOLO v3* no resulta factible. También se utiliza el Intel *NCS 2* y para la lectura del video *imutils*:

DATOS OBTENIDOS DE LAS PRUEBAS						
Video	**Carros**	**Motos**	**Personas**	**Camiones**	**Buses**	**Total**
LATERAL1-400	14	00	00	00	00	14
FRONTAL1-400	24	00	01	00	01	26
FRONTAL2-400	20	00	00	00	00	20
FRONTAL3-400	27	00	00	00	00	27
FRONTAL4-400	09	00	00	00	01	10
DATOS REALES						
Video	**Carros**	**Motos**	**Personas**	**Camiones**	**Buses**	**Total**
LATERAL1-400	14	00	00	01	01	16
FRONTAL1-400	22	00	01	00	01	24
FRONTAL2-400	20	01	00	00	00	21
FRONTAL3-400	25	00	01	01	00	27
FRONTAL4-400	9	00	00	01	00	10

Tabla 11: Datos de conteo de los videos utilizados obteniendo el conteo con el factor humano (DATOS REALES) y con el sistema presentado (DATOS OBTENIDOS DE LAS PRUEBAS).

Estos datos se almacenan en un diccionario de Python el cual tiene como llave el tipo de vehículo y como dato el número de vehículos de esta clase que han pasado por el camino. Se presentan los resultados en una matriz de confusión en la siguiente figura 16:

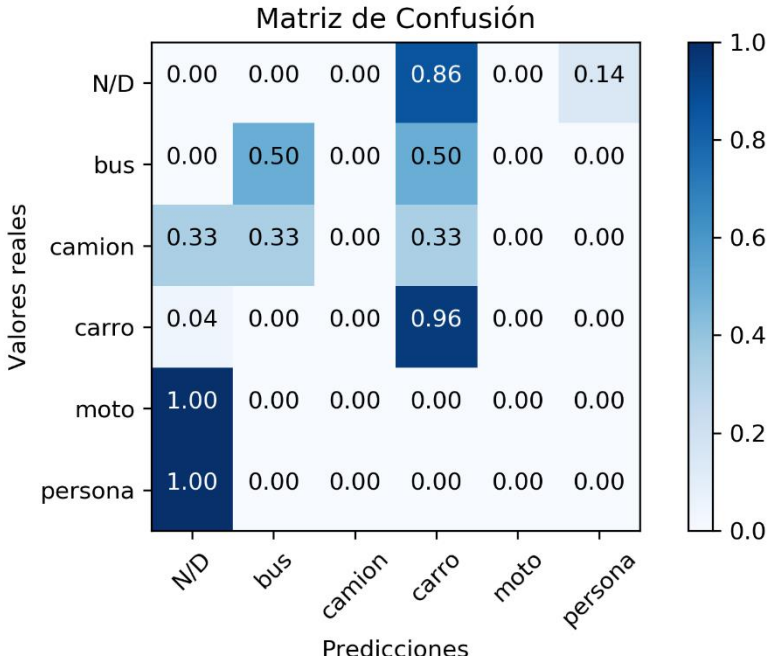


Figura 16: Matriz de confusión generada a partir de los resultados del conteo humano.

N/D hace referencia a que el algoritmo hace detecciones que no existen (véase figura 17) o cuando se presentan dobles detecciones (véase figura 18).

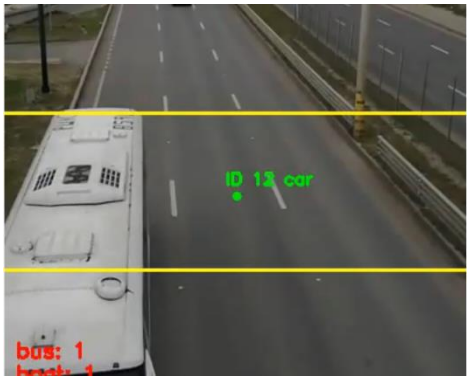


Figura 17: Ejemplo de detección incorrecta, se observa que el algoritmo identifica a toda la imagen como un objeto determinado; en este caso un vehículo.

Figura 18: Ejemplo de conteo doble. En algunos casos como el que se observa en la figura los vehículos con parrillas especiales son detectados como dos vehículos.

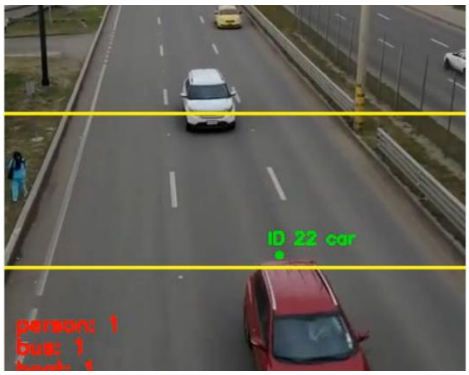


Figura 19: Ejemplo de detección inexistente de la persona a la izquierda.

También se usa para el otro escenario cuando existe por ejemplo una persona (véase figura 19), pero el algoritmo no la identifica entonces se coloca el resultado en no detecta (N/D). Como se puede observar, el algoritmo tiene un nivel de exactitud de detección de carros del 89% en los videos propuestos. En el caso del bus lo detecta siempre, pero existen dobles detecciones en donde lo identifica como carro. En el caso de N/D el modelo presenta dobles detecciones o detecciones falsas (falsos positivos) con la clase de carro y en un bajo porcentaje de falsos positivos con la clase persona. Con respecto a las otras medidas que se obtuvieron a partir de la matriz de confusión se obtienen los siguientes resultados presentados en la tabla 12:

	N/D	**bus**	**camión**	**carro**	**moto**	**persona**
Exhaustividad	0%	50%	0%	96%	0%	0%
Especificidad	92%	99%	100%	47%	100%	99%
Precisión	0%	50%	0%	91%	0%	0%
Exactitud	***86%***	***98%***	***97%***	***89%***	***99%***	***97%***

Tabla 12: Resultados obtenidos de la matriz de confusión presentados en términos de exhaustividad, especificidad y precisión.

Las medidas más importantes son exhaustividad, exactitud, precisión y especificidad. La exhaustividad hace referencia a lo siguiente: Por ejemplo, considerando la clase de carros, cuántos de ellos fueron correctamente detectados en porcentaje de los que sí eran carros. La exactitud nos indica la proporción de clasificaciones correctas de todos los casos posibles. La precisión es la proporción del número de positivos correctos para los falsos positivos y verdaderos positivos. Finalmente, la especificidad es el porcentaje de verdaderos negativos obtenidos.

4.3.4 Evaluación de uso de recursos:

Los resultados de esta prueba se obtuvieron al muestrear cuantos recursos con respecto al CPU y memoria RAM consume el proceso del sistema elaborado. Con respecto al rendimiento, se obtuvieron mejoras notables al incorporar el *Intel NCS 2* (véase figuras 20 y 21), el cual permite aumentar la velocidad de reconocimiento de objetos. Como se mencionó en el estado de arte el *NCS 2* es un dispositivo que está preparado para el procesamiento visual. Para lograr cargar el modelo en el *NCS 2* en el caso de *YOLO v3* y *Tiny YOLO v3* se utilizaron los modelos transformados con *OpenVINO* (Los modelos de representación intermedia IR). Para *SSD* se utilizaron los modelos normales que vienen en formato “caffemodel” y “prototxt”, pero con la opción de ser procesado por el *Inference Engine* como *backend*. Esto quiere decir que, aunque no

se obtengan los modelos intermediarios la librería los transforma durante la ejecución de la mejor manera y es procesado por el motor de inferencia el cual se realiza en el dispositivo *Intel NCS 2*.

Para calcular el consumo de recursos se ejecutó un script que tomaba muestras del Raspberry Pi 3B+ cada 500ms. Esta muestra contiene el consumo de CPU y RAM del dispositivo para cada proceso. En este caso se hizo el enfoque en el proceso de Python que es donde que se ejecuta el sistema propuesto.

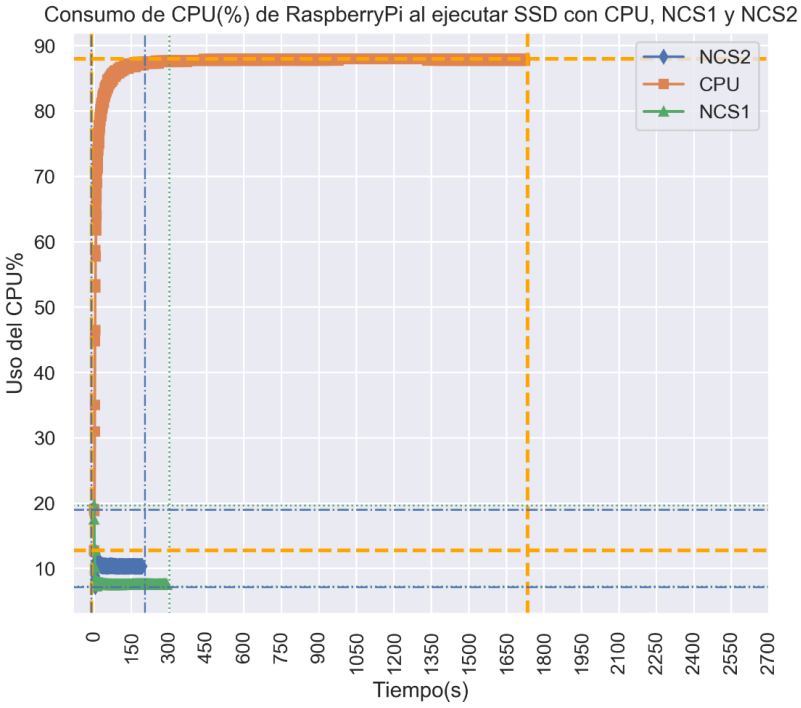


Figura 20: Consumo de CPU en porcentaje ejecutando *SSD*.

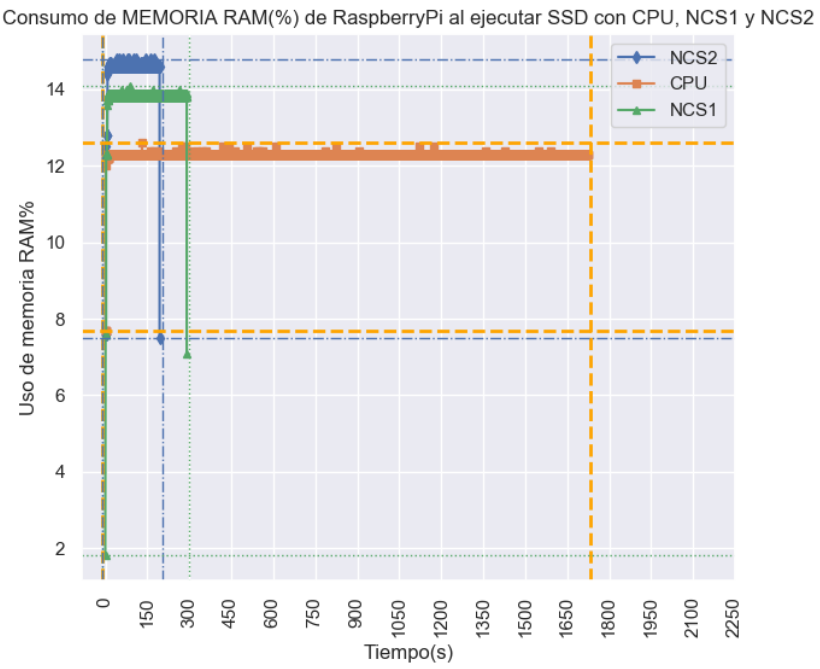


Figura 21: Consumo de memoria RAM en porcentaje ejecutando *SSD*.

Al analizar las gráficas se observa que aún quedan recursos libres para utilizar, según la cantidad de recursos libres evidentes se podrían agregar hasta 4 dispositivos *NCS 2* más y esto daría la capacidad de procesar hasta aproximadamente 4 flujos de video simultáneamente (cada flujo ejecutándose en un hilo distinto para cada *NCS 2*). En cada gráfica se visualizan tres líneas, la primera con cuadros hace referencia a la ejecución de todo el proceso de un video sin componente de *VPU*, la segunda con triángulos indica el consumo al utilizar el *NCS 1* (la primera generación del dispositivo *NCS*). La tercera línea indica el consumo de recursos con *NCS 2*, como se observa en la figura 20 el consumo de CPU es más alto con el *NCS 2* que con el *NCS 1*, sin embargo, el tiempo de ejecución es más corto. Esto se debe a que el *NCS 2* procesa más rápido los marcos, por lo tanto, exige que el procesador trabaje más rápido para entregarle los marcos. La situación mencionada a la vez resulta en un consumo más alto de memoria RAM del Raspberry Pi 3B+. Es interesante observar que al aplicar cualquier dispositivo *NCS* el consumo de CPU se reduce notablemente comparado con el caso de no utilizar un dispositivo con *VPU*. El consumo obtenido del CPU con el *NCS 2* es de 10% en promedio. Es relevante señalar que el número promedio de FPS con el *NCS 2* en los experimentos y con el consumo de recursos indicados es de 37.

Este proyecto es un inicio, en el futuro se espera mejorar las medidas de precisión, exhaustividad y exactitud. La mayoría de los proyectos previos como [2], [9] no hacen uso de redes neuronales profundas y los que sí lo hacen utilizan equipos con mayor capacidad como servidores o computadores de escritorio.

Capítulo V

DISCUSIÓN

Existen múltiples mejoras y otros experimentos que se pueden aplicar con el objetivo de mejorar el rendimiento de este proyecto. Una de estas es generar el modelo IR de *SSD*, posiblemente representando una mejora en la velocidad e incluso disminuyendo el valor de salto de marco. Mejorando así el conteo y disminuyendo dobles detecciones. Otra prueba que se podría realizar a futuro es implementar el proyecto compilado con C++, este es un lenguaje de bajo nivel que generalmente representa un rendimiento optimizado. Adicionalmente, es posible mejorar el nivel de detección entrenando un modelo con un dataset propio, para esto es necesario tener un dataset con imágenes locales y etiquetar cada objeto con la clase requerida; es un proceso que toma tiempo, recursos de computación y paciencia. En este caso las optimizaciones propuestas fueron al realizar el modelo IR y hacer uso de motor de inferencia de OpenVINO.

Otra recomendación es hacer pruebas más extensas con videos prolongados o haciéndolo directo de una cámara con un flujo en tiempo real. Para determinar si es que está detectando correctamente se podrían almacenar las fotos con la etiqueta propuesta el momento que detecta y clasifica un vehículo. Luego se puede revisar cuales están clasificadas incorrectamente. El problema sería que no se tiene un conteo del video de los carros; si es que ha habido casos en los que pasa un vehículo y no es detectado.

Con respecto al método del centroide se puede mejorar notablemente al reducir el número de saltos de marcos, a la vez reduciendo los parámetros de distancia y desaparición máximas. Esto haría que resulte más preciso el algoritmo y al minimizar la distancia máxima se elimina el problema de cambio de identificadores de vehículos por la confusión de la distancia euclidiana. Sin embargo, esto representaría otro problema cuando un vehículo pase de forma veloz porque la distancia de aparición entre cada marco sería muy alta, aunque este sería un caso extraordinario. También se puede poner más etiquetas en base a otros factores por ejemplo color o forma y que se haga el seguimiento en base a todas las etiquetas identificadas.

Capítulo VI

CONCLUSIÓN

En este proyecto se realizó un sistema de conteo de actores de movilidad mediante un Raspberry Pi 3B+ juntamente con un Intel *NCS 2*. La mejor alternativa con respecto al modelo de redes neuronales para la detección y clasificación fue *SSD*. Se descartó *YOLO v3* en las pruebas de detección y clasificación por el tiempo que tomó para realizar el procesamiento. Se logró realizar todo el proceso en tiempo real alcanzando a procesar un video de 30FPS y 400x400px. Esto fue posible alcanzar por el uso del *NCS 2* y la implementación del parámetro "*Skip frame*". El consumo de memoria RAM y CPU se mantuvo estable con un 14.5% y 10% respectivamente. Esto significa que se pueden agregar más flujos de video juntamente con dispositivos Intel *NCS 2* para realizar el conteo de múltiples cámaras. También se podría utilizar el dispositivo embarcado para realizar múltiples operaciones adicionales con los recursos restantes.

Los pasos generales que se realizaron para alcanzar el objetivo fueron investigar el funcionamiento general de un sistema de detección de objetos tradicional y con redes neuronales mediante revisión literaria. Una vez concluida esta etapa se procedió a armar el experimento, obtener los videos a partir de distintos ángulos y seleccionar e instalar las plataformas, programas y requerimientos para la ejecución de los modelos *SSD* y *YOLO v3*. Entre la selección de software realizada esta *OpenVINO*, *Python*, *imutils*, *DLIB* y el sistema operativo del Raspberry Pi 3B+ con *Raspbian stretch*. El siguiente paso fue iniciar la fase de experimentación. En la etapa de experimentación se logró determinar parámetros como el de máxima distancia y desaparición en el caso del centroide, el valor de "*Skip frame*" para la lectura en tiempo continuo y las variables requeridas para el flujo de video como la resolución, el tipo de conteo que se debe aplicar como zona de conteo o línea de conteo con el objetivo de obtener los mejores resultados. La detección y clasificación de carros con los videos probados presenta un escenario satisfactorio alcanzando una precisión del 91%. Esta clase es la más importante porque representa la mayor cantidad de agentes de movilidad.

Se logró reportar el rendimiento y consumo de recursos del Raspberry Pi 3B+ con y sin el *NCS 2* en términos de consumo de CPU y RAM en porcentaje. El conteo se logró realizar implementando el método del centroide y descartando el de correlación entre marcos porque el tiempo que tomaba realizar el análisis con el de correlación es mayor al tiempo que tomaba ejecutar la detección y clasificación de la red neuronal con el *NCS 2*.

Es importante recordar que este proyecto puede mejorar con las optimizaciones propuestas en la sección de discusión y recomendaciones.

REFERENCIAS

[1] L. Figueiredo, I. Jesus, J. A. T. Machado, J. R. Ferreira, and J. L. Martins de Carvalho, "Towards the development of intelligent transportation systems," in *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No.01TH8585)*, 2001, pp. 1206–1211.

[2] F. Torres, G. Barros, and M. J. Barros, "Computer vision classifier and platform for automatic counting: More than cars," *2017 IEEE 2nd Ecuador Tech. Chapters Meet. ETCM 2017*, pp. 1–6, 2017.

[3] E. Soroush, A. Mirzaei, and S. Kamkar, "Near Real-Time Vehicle Detection and Tracking in Highways," no. March 2017, p. 11, 2016.

[4] A.-O. Fulop and L. Tamas, "Lessons learned from lightweight CNN based object recognition for mobile robots," 2018.

[5] P. Abrahamsson *et al.*, "Affordable and energy-efficient cloud computing clusters: The Bolzano Raspberry Pi cloud cluster experiment," *Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom*, vol. 2, pp. 170–175, 2013.

[6] D. Pena, A. Forembski, X. Xu, and D. Moloney, "Benchmarking of CNNs for Low-Cost, Low-Power Robotics Applications," 2017.

[7] J. Huang *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 3296–3305, 2017.

[8] Ministerio del Trabajo, "Incremento del Salario Básico Unificado 2019.," 2018. [Online]. Available: http://www.trabajo.gob.ec/incremento-del-salario-basico-unificado-2019/. [Accessed: 10-Sep-2019].

[9] Z. Chen, T. Ellis, and S. A. Velastin, "Vehicle detection, tracking and classification in urban traffic," *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, pp. 951–956, 2012.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016.

[11] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6517–6525, 2017.

[12] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018.

[13] W. Liu *et al.*, "SSD: Single shot multibox detector," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016.

[14] R. Girshick, "Fast R-CNN," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015.

[15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017.

[16] M. J. Shaifee, B. Chywl, F. Li, and A. Wong, "Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video," *J. Comput. Vis. Imaging Syst.*, vol. 3, no. 1, 2017.

[17] L. Zhang, L. Lin, X. Liang, and K. He, "Is faster R-CNN doing well for pedestrian detection?," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9906 LNCS, pp. 443–457, 2016.

[18] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," *MM 2014 - Proc. 2014 ACM Conf. Multimed.*, vol. abs/1506.0, pp. 675–678, 2014.

[19] P. B. Martín Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," *Methods Enzymol.*, 2016.

[20] Y. LeCun *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[21] Chuanqi305, "MobileNet-SSD," 2019. [Online]. Available: https://github.com/chuanqi305/MobileNet-SSD. [Accessed: 10-Sep-2019].

[22] A. Rosebrock, "Real-time object detection on the Raspberry Pi with the Movidius NCS," 2018. [Online]. Available: https://www.pyimagesearch.com/2018/02/19/real-time-object-detection-on-the-raspberry-pi-with-the-movidius-ncs/. [Accessed: 10-Sep-2019].

[23] Taehoonlee, "High level network definitions with pre-trained weights in TensorFlow," 2019. [Online]. Available: https://github.com/taehoonlee/tensornets. [Accessed: 10-Sep-2019].

[24] K. Hyodo, "YoloV3," 2019. [Online]. Available: https://github.com/PINTO0309. [Accessed: 10-Sep-2019].

[25] J. Redmon and A. Farhadi, "Tiny YOLOv3," 2018. [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed: 10-Sep-2019].

[26] J. Hui, "SSD object detection: Single Shot MultiBox Detector for real-time processing," 2018. [Online]. Available: https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06. [Accessed: 10-Sep-2019].

[27] CyberAILab, "A Closer Look at YOLOv3," 2018. [Online]. Available: https://www.cyberailab.com/home/a-closer-look-at-yolov3. [Accessed: 10-Sep-2019].

[28] T. Y. Lin *et al.*, "Microsoft COCO: Common objects in context," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8693 LNCS, no. PART 5, pp. 740–755, 2014.

[29] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010.

[30] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012)," 2012. [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/. [Accessed: 10-Sep-2019].

[31] A. Rosebrock, "Simple object tracking with OpenCV," 2018. [Online]. Available: https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/. [Accessed: 10-Sep-2019].

[32] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg, "Accurate scale estimation for robust visual tracking," *BMVC 2014 - Proc. Br. Mach. Vis. Conf. 2014*, 2014.

[33] D. King, "Dlib C++ Library Python API," 2019. [Online]. Available: http://dlib.net/python/index.html. [Accessed: 10-Oct-2019].

[34] Intel, "Deep Learning For Computer Vision," 2019. [Online]. Available: https://software.intel.com/en-us/openvino-toolkit/deep-learning-cv. [Accessed: 10-Oct-2019].

[35] Intel, "Intel® Neural Compute Stick 2," 2019. [Online]. Available: https://software.intel.com/en-us/neural-compute-stick. [Accessed: 10-Oct-2019].

[36] Raspberry Pi, "Raspberry Pi 3 Model B+," 2018. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. [Accessed: 10-Oct-2019].

[37] Python, "About Python," 2019. [Online]. Available: https://www.python.org/about/. [Accessed: 10-Oct-2019].

[38] A. Rosebrock, "Faster video file FPS with cv2.VideoCapture and OpenCV," 2017. [Online]. Available: https://www.pyimagesearch.com/2017/02/06/faster-video-file-fps-with-cv2-videocapture-and-opencv/. [Accessed: 10-Oct-2019].

Anexos

Anexo 1:

Video	Número de video	FPS	Resolución	Duración (min)
FRENTE1	1	30.48	620x620	1
FRENTE1-500	2	30.48	500x500	1
FRENTE1-400	3	30.48	400x400	1
FRENTE1-300	4	30.48	300x300	1
FRENTE1-200	5	30.48	200x200	1
FRENTE2-400	6	30.46	400x400	1
FRENTE3-400	7	30.11	400x400	1
FRENTE4-400	8	30.25	400x400	1
LATERAL1-400	9	30.46	400X400	1

Tabla 13: Los contenidos de datos de video

DECANATO DE LA FACULTAD DE CIENCIAS DE LA ADMINISTRACIÓN.- Cuenca, 11 de julio de 2019.-En atención a la solicitud que antecede, presentada por el estudiante David Andrés Tapia Moscoso (cód. 74981), para que se le conceda prórroga a la presentación del trabajo de titulación denominado ""Rendimiento de Raspberry Pi con Intel_NCS ejecutando métodos de aprendizaje profundo neuronal en clasificación de tipo de vehículos en tiempo continuo", , previo a la obtención del título de Ingeniero de Sistemas y Telemática, cuyo plazo de presentación está previsto para el 23 de julio de 2019; considerando el informe favorable del Director del trabajo, se resuelve aprobar la solicitud y conceder una prórroga de seis meses, debiendo el estudiante presentar su trabajo concluido hasta el 23 de enero de 2020.

Ing. Oswaldo Merchán Manzano
DECANO DE LA FACULTAD DE
CIENCIAS DE LA ADMINISTRACIÓN

UNIVERSIDAD DEL AZUAY

Doctora María Elena Ramírez Aguilar, Secretaria de la Facultad de Ciencias de la Administración de la Universidad del Azuay

CERTIFICA:

Que, el Consejo de Facultad en sesión del 23 de enero de 2019, conoció y aprobó la solicitud para realización del trabajo de titulación, presentada por:

Estudiante:	David Andrés Tapia Moscoso (cód. 74981)
Tema:	**"Rendimiento de RaspberryPi con Intel_NCS ejecutando métodos de aprendizaje profundo neuronal en clasificación de tipo de vehículos en tiempo continuo".** Previo a la obtención del título de Ingeniero de Sistemas y Telemática
Director:	Ing. Juan Gabriel Barros Gavilanez
Tribunal:	Ing. Marcos Orellana Cordero e Ing. Lenin Erazo Garzón

Plazo de presentación del trabajo de titulación: Se fijó como plazo para la entrega del trabajo de titulación, conforme a la Disposición Tercera del Reglamento de Régimen Académico, un período académico contado desde la fecha de aprobación del diseño del trabajo, esto es hasta el 23 de julio de 2019.

E INFORMA:

Que, en aplicación de la Disposición General Cuarta del Reglamento de Régimen Académico vigente, en caso de que el estudiante no culmine y apruebe el trabajo de titulación luego de dos períodos académicos contados a partir de su fecha de culminación de estudios, deberá realizar la actualización de conocimientos previa a su titulación.

Cuenca, 24 de enero de 2019

Dra. María Elena Ramírez Aguilar
Secretaria de la Facultad de
Ciencias de la Administración

SECRETARÍA