



**ESPE**  
UNIVERSIDAD DE LAS FUERZAS ARMADAS  
INNOVACIÓN PARA LA EXCELENCIA

**DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y  
TELECOMUNICACIONES**

**CARRERA DE INGENIERÍA EN ELECTRÓNICA Y  
TELECOMUNICACIONES**

**TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE  
INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES**

**TEMA: “IMPLEMENTACIÓN DE UN PROTOTIPO DE SISTEMA  
AUTÓNOMO DE VISIÓN ARTIFICIAL PARA LA DETECCIÓN DE OBJETOS EN  
VIDEO UTILIZANDO TÉCNICAS DE APRENDIZAJE PROFUNDO”**

**AUTOR: SALAZAR GUERRERO, JONATHAN EUGENIO**

**DIRECTOR: ING. SILVA TAPIA, RODRIGO**

**SANGOLQUÍ**

**2019**

## CERTIFICACIÓN



DEPARTAMENTO DE ELÉCTRICA,  
ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERIA EN ELECTRONICA Y TELECOMUNICACIONES

### CERTIFICACIÓN

Certifico que el trabajo de titulación, *“IMPLEMENTACIÓN DE UN PROTOTIPO DE SISTEMA AUTÓNOMO DE VISIÓN ARTIFICIAL PARA LA DETECCIÓN DE OBJETOS EN VIDEO UTILIZANDO TÉCNICAS DE APRENDIZAJE PROFUNDO”* fue realizado por el señor *Salazar Guerrero Jonathan Eugenio* el mismo que ha sido revisado en su totalidad, analizado por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Sangolquí, 01 de Julio de 2019

Ing. Silva Tapia Rodrigo  
C.C.: 0602199523

## AUTORÍA DE RESPONSABILIDAD



DEPARTAMENTO DE ELÉCTRICA,  
ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERIA EN ELECTRONICA Y TELECOMUNICACIONES

### *AUTORÍA DE RESPONSABILIDAD*

*Yo, Salazar Guerrero Jonathan Eugenio, declaro que el contenido, ideas y criterios del trabajo de titulación: “IMPLEMENTACIÓN DE UN PROTOTIPO DE SISTEMA AUTÓNOMO DE VISIÓN ARTIFICIAL PARA LA DETECCIÓN DE OBJETOS EN VIDEO UTILIZANDO TÉCNICAS DE APRENDIZAJE PROFUNDO” es de mi autoría y responsabilidad, cumpliendo con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.*

*Consecuentemente el contenido de la investigación mencionada es veraz.*

Sangolquí, 01 de Julio de 2019

Salazar Guerrero Jonathan Eugenio  
C.C.: 1003682174

## AUTORIZACIÓN



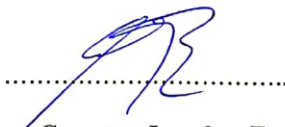
DEPARTAMENTO DE ELÉCTRICA,  
ELECTRÓNICA Y TELECOMUNICACIONES

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

### AUTORIZACIÓN

*Yo, Salazar Guerrero Jonathan Eugenio autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: “IMPLEMENTACIÓN DE UN PROTOTIPO DE SISTEMA AUTÓNOMO DE VISIÓN ARTIFICIAL PARA LA DETECCIÓN DE OBJETOS EN VIDEO UTILIZANDO TÉCNICAS DE APRENDIZAJE PROFUNDO” en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.*

Sangolquí, 01 de Julio de 2019



Salazar Guerrero Jonathan Eugenio  
C.C.: 1003682174

## **DEDICATORIA**

A mi familia por ayudarme en este largo camino de la carrera de ingeniería electrónica, por el apoyo recibido a lo largo de los años, por las palabras de aliento y superación que son muy importantes en este camino de la vida universitaria. A mi madre por todo el esfuerzo realizado ayudando a su hijo con todas las facilidades para que pueda cumplir su objetivo. A mi padre por todos los consejos que día a día sirvieron para superar estos duros años de estudio. A mis hermanos por siempre demostrarme cariño y confianza. A Dios por regalarme la vida y la salud y haber dado esta maravillosa familia.

Jonathan Eugenio Salazar Guerrero

## AGRADECIMINETO

A mis padres Giovanna, Marcelo y Eugenio por darme todo el cariño y apoyo para cumplir con esta meta propuesta, y enseñarme el significado del respeto y la valoración de todo su esfuerzo. Agradezco todos los consejos y palabras de aliento para salir adelante frente a duras situaciones presentadas en mi carrera universitaria. A mis hermanos Oscar y Joseph por ser los mejores hermanos cuando más los necesite. A mis amigos universitarios que me demostraron que con mucho esfuerzo y trabajo en equipo se pueden conseguir todas las metas planteadas, por ser unas grandes personas con las que compartí los mejores años de mi vida. A mi director Ing. Rodrigo Silva Tapia por su fundamental ayuda, paciencia y tiempo dedicado en la realización de este proyecto.

Jonathan Eugenio Salazar Guerrero

**ÍNDICE GENERAL**

<b>CERTIFICACIÓN.....</b>	<b>i</b>
<b>AUTORÍA DE RESPONSABILIDAD.....</b>	<b>ii</b>
<b>AUTORIZACIÓN .....</b>	<b>iii</b>
<b>DEDICATORIA.....</b>	<b>iv</b>
<b>AGRADECIMINETO .....</b>	<b>v</b>
<b>ÍNDICE DE TABLAS .....</b>	<b>ix</b>
<b>ÍNDICE DE FIGURAS .....</b>	<b>x</b>
<b>RESUMEN .....</b>	<b>xiii</b>
<b>ABSTRACT.....</b>	<b>xiv</b>
<b>CAPITULO 1 .....</b>	<b>1</b>
<b>1. INTROCUCCIÓN .....</b>	<b>1</b>
1.1 Antecedentes .....	1
1.2 Justificación e Importancia .....	3
1.3 Alcance del Proyecto .....	5
1.4 Objetivos .....	6
1.4.1 General .....	6
1.4.2 Específicos .....	6
<b>CAPITULO 2 .....</b>	<b>7</b>

<b>2. FUNDAMENTO TEÓRICO.....</b>	<b>7</b>
2.1 Redes neuronales.....	7
2.1.1 Características de las redes neuronales. ....	9
2.2 Aprendizaje Profundo (Deep Learning).....	9
2.2.1 Estructura Aprendizaje Profundo.....	11
2.3 Redes Neuronales Convolucionales (CNN).....	13
2.3.1 Capa Convolutional.....	14
2.3.2 Capa de reducción o pooling.....	15
2.3.3 Capa clasificadora totalmente conectada .....	16
2.4 Algoritmos de detección de objetos .....	17
2.4.1 R-CNN .....	18
2.4.2 R-CNN Rápido (Fast R-CNN).....	20
2.4.3 R-CNN más rápido (Faster R-CNN) .....	21
2.4.4 YOLO (You Only Look Once) .....	23
<b>CAPITULO 3 .....</b>	<b>27</b>
<b>3. MATERIALES Y MÉTODOS .....</b>	<b>27</b>
3.1 Introducción .....	27
3.2 Hardware Nvidia Jetson Tx2.....	27
3.2.1 Especificaciones técnicas del módulo.....	28



3.2.2 Paquete de desarrollo Jetpack .....	29
3.3 YoloV3.....	29
3.4.1 Detección a tres escalas. ....	30
<b>CAPITULO 4 .....</b>	<b>35</b>
<b>4. IMPLEMENTACIÓN DEL PROTOTIPO DE SISTEMA AUTÓNOMO</b>	
<b>DE VISIÓN ARTIFICIAL.....</b>	<b>35</b>
4.1 Instalación de JetPack 3.2 en la tarjeta Jetson Tx2.....	35
4.2 Descarga de la red neuronal YOLO en la tarjeta Jetson TX2.....	40
<b>CAPITULO 5 .....</b>	<b>47</b>
<b>5. PRUEBAS Y ANÁLISIS DE RESULTADOS .....</b>	<b>47</b>
5.1 Parámetros de evaluación. ....	47
5.2 Pruebas con YoloV3 .....	49
5.3 Pruebas con YoloV3 Tiny.....	60
<b>CAPÍTULO 6 .....</b>	<b>72</b>
<b>6. CONCLUSIONES Y RECOMENDACIONES.....</b>	<b>72</b>
6.2 Conclusiones .....	72
6.3 Recomendaciones.....	73
<b>REFERENCIAS BIBLIOGRÁFICAS.....</b>	<b>74</b>

## ÍNDICE DE TABLAS

<b>Tabla 1.</b> <i>Características del hardware Jetson Tx2.</i> .....	28
<b>Tabla 2.</b> <i>Relación entre valores positivos y negativos</i> .....	47
<b>Tabla 3.</b> <i>Frames capturados de video para la prueba 1, con el algoritmo YoloV3.</i> .....	50
<b>Tabla 4.</b> <i>Frames capturados de video para la prueba 2, con el algoritmo YoloV3.</i> .....	51
<b>Tabla 5.</b> <i>Frames capturados de video para la prueba 3, con el algoritmo YoloV3.</i> .....	52
<b>Tabla 6.</b> <i>Frames capturados de video para la prueba 4, con el algoritmo YoloV3.</i> .....	53
<b>Tabla 7.</b> <i>Frames capturados de video para la prueba 5, con el algoritmo YoloV3.</i> .....	55
<b>Tabla 8.</b> <i>Frames capturados de video para la prueba 6, con el algoritmo YoloV3.</i> .....	56
<b>Tabla 9.</b> <i>Frames capturados de video para la prueba 7, con el algoritmo YoloV3.</i> .....	58
<b>Tabla 10.</b> <i>Frames capturados de video para la prueba 7, con el algoritmo YoloV3.</i> .....	60
<b>Tabla 11.</b> <i>Frames capturados de video para la prueba 1, con el algoritmo YoloV3-tiny.</i> ..	61
<b>Tabla 12.</b> <i>Frames capturados de video para la prueba 2, con el algoritmo YoloV3-tiny.</i> ..	62
<b>Tabla 13.</b> <i>Frames capturados de video para la prueba 3, con el algoritmo YoloV3-tiny.</i> ..	63
<b>Tabla 14.</b> <i>Frames capturados de video para la prueba 4, con el algoritmo YoloV3-tiny.</i> ..	64
<b>Tabla 15.</b> <i>Frames capturados de video para la prueba 5, con el algoritmo YoloV3-tiny.</i> ..	65
<b>Tabla 16.</b> <i>Frames capturados de video para la prueba 6, con el algoritmo YoloV3-tiny.</i> ..	67
<b>Tabla 17.</b> <i>Frames capturados de video para la prueba 7, con el algoritmo YoloV3-tiny.</i> ..	68
<b>Tabla 18.</b> <i>Frames capturados de video para la prueba 8, con el algoritmo YoloV3-tiny.</i> ..	69
<b>Tabla 19.</b> <i>Comparación de parámetros de evaluación YoloV3 y YoloV3-tiny</i> .....	70

## ÍNDICE DE FIGURAS

<i>Figura 1.</i> Estructura de una red neuronal artificial. ....	8
<i>Figura 2.</i> Historia del nacimiento de la Inteligencia Artificial .....	11
<i>Figura 3.</i> Estructura Deep Learning.....	12
<i>Figura 4.</i> Funcionamiento de una capa convolucional. ....	15
<i>Figura 5.</i> Max-pooling aplicado a una imagen .....	16
<i>Figura 6.</i> Método de búsqueda selectiva.....	18
<i>Figura 7.</i> Funcionamiento del algoritmo Fast R-CNN .....	20
<i>Figura 8.</i> Comparación de algoritmos R-CNN, Fast R-CNN.....	21
<i>Figura 9.</i> Algoritmo Faster R-CNN .....	22
<i>Figura 10.</i> Comparación de prueba de velocidad de algoritmos .....	23
<i>Figura 11.</i> Funcionamiento del algoritmo YOLO .....	24
<i>Figura 12.</i> Recuadro delimitador.....	25
<i>Figura 13.</i> Kit de desarrollo Jetson Tx2.....	28
<i>Figura 14.</i> Arquitectura de red Yolo V3 .....	30
<i>Figura 15.</i> Detección a escala 1 .....	33
<i>Figura 16.</i> Detección a escala 2 .....	33
<i>Figura 17.</i> Detección a escala 3. ....	34
<i>Figura 18.</i> Estructura física para la instalación.....	36
<i>Figura 19.</i> Jetpack versión 3.2. ....	36
<i>Figura 20.</i> Archivo. run .....	37
<i>Figura 21.</i> Comandos para ejecutar el instalador.....	37
<i>Figura 22.</i> Instalación de componentes en el host. ....	38

<i>Figura 23.</i> Network layout.....	38
<i>Figura 24.</i> Post Instalación. ....	39
<i>Figura 25.</i> Conexión cable micro usb.....	39
<i>Figura 26.</i> Combinación de botones en la tarjeta .....	40
<i>Figura 27.</i> Github Alexey/AB.....	42
<i>Figura 28.</i> Red yolo (darknet-master).....	42
<i>Figura 29.</i> Edición archivo Makefile.....	43
<i>Figura 30.</i> Archivo Makefile. ....	44
<i>Figura 31.</i> Ejecución de comando make en consola.....	44
<i>Figura 32.</i> Archivo coco.data.....	45
<i>Figura 33.</i> Archivo coco.names .....	45
<i>Figura 34.</i> Archivo demo.c .....	46
<i>Figura 35.</i> Línea de comando para ejecutar el algoritmo. ....	46
<i>Figura 36.</i> Detección del sistema tomado del video 120 de la base de datos. ....	49
<i>Figura 37.</i> Detección de sistema autónomo, video 122 de la base de datos. ....	50
<i>Figura 38.</i> Detección de sistema autónomo, video 122 de la base de datos. ....	52
<i>Figura 39.</i> Detección de sistema autónomo, video 2.....	53
<i>Figura 40.</i> Detección de sistema autónomo, video 3.....	54
<i>Figura 41.</i> Detección de sistema autónomo, video 4 de cámara de video vigilancia. ....	56
<i>Figura 42.</i> Detección de sistema autónomo, video capturado en tiempo real .....	57
<i>Figura 43.</i> Detección de sistema autónomo, video capturado en tiempo real. ....	59
<i>Figura 44.</i> Detección de sistema autónomo, video 120 tomado de la base de datos.....	61
<i>Figura 45.</i> Detección de sistema autónomo, video 121 tomado de la base de datos.....	62

<i>Figura 46.</i> Detección de sistema autónomo, video 143 tomado de la base de datos. ....	63
<i>Figura 47.</i> Detección de sistema autónomo, video 2 tomado de la base de datos. ....	64
<i>Figura 48.</i> Detección de sistema autónomo, video 3 tomado de la base de datos. ....	65
<i>Figura 49.</i> Detección de sistema autónomo, video 4 de cámara de video vigilancia. ....	66
<i>Figura 50.</i> Detección de sistema autónomo, video capturado entiendo real. ....	68
<i>Figura 51.</i> Detección de sistema autónomo, video capturado entiendo real. ....	69
<i>Figura 52.</i> Comparativa Miss Rate. ....	71
<i>Figura 53.</i> Comparativa Recall. ....	71

## RESUMEN

Hoy en día la visión por computador o visión artificial es un campo con diversas aplicaciones en áreas tales como la seguridad ciudadana, instrumentación médica, líneas de procesos industriales, actividades militares, etc. El uso de algoritmos y técnicas de visión artificial basadas en machine learning y deep learning están haciéndose cada vez más populares por cuanto ya existen a la mano computadores pequeños con gran capacidad de procesamiento tales como los arduinos, raspberry, nvidia y otros. El objetivo de este proyecto es implementar un pequeño prototipo de sistema autónomo de visión artificial capaz de detectar autos y personas en video capturado en tiempo real cuyo detector se basa en el algoritmo YOLOv3 en la versión AlexeyAB-Darknet corriendo sobre una placa de hardware de alto rendimiento NVIDIA Jetson TX2 y un framework basado en Linux-Python-OpencvTensor-Cuda. YOLOv3 es un algoritmo pre-entrenado para detectar ochenta objetos, sin embargo, para propósito de este trabajo, el detector ha sido configurado para mostrar personas y autos livianos. En las pruebas realizadas en el prototipo, se alcanzó una tasa de acierto superior al noventa por ciento con YOLOv3 en la versión completa, aunque la velocidad de procesamiento del detector resultó inferior comparada con la versión liviana del mismo algoritmo al probarse con videos pregrabados y también en videos capturados en tiempo real a través de una webcam conectada en el prototipo.

### **PALABRAS CLAVE:**

- **VISIÓN ARTIFICIAL**
- **ALEXEY AB-DARKNET**
- **YOLO V3**
- **JETSON TX2**

## **ABSTRACT**

Nowadays computer vision or artificial vision is a field with diverse applications in areas such as citizen security, medical instrumentation, industrial process lines, military activities, etc. The use of algorithms and machine vision techniques based on machine learning and deep learning are becoming increasingly popular because small computers with large processing capacity such as arduinos, raspberry, nvidia and others already exist at hand. The objective of this project is to implement a small prototype of an autonomous system of artificial vision capable of detecting cars and people in video captured in real time whose detector is based on the YOLOv3 algorithm in the AlexeyAB-Darknet version running on a high hardware plate NVIDIA Jetson TX2 performance and a framework based on Linux-Python-OpencvTensor-Cuda. YOLOv3 is a pre-trained algorithm to detect eighty objects, however, for the purpose of this work, the detector has been configured to show light people and cars. In the tests performed in the prototype, a success rate of over ninety percent was achieved with YOLOv3 in the full version, although the processing speed of the detector was lower compared to the light version of the same algorithm when tested with prerecorded videos and also in videos captured in real time through a webcam connected in the prototype.

### **KEY WORDS:**

- **ARTIFICIAL VISION**
- **ALEXEY AB-DARKNET**
- **YOLO V3**
- **JETSON TX2**

## **CAPITULO 1**

### **1. INTRODUCCIÓN**

#### **1.1 Antecedentes**

Los sistemas autónomos de visión artificial (VA) han tenido un crecimiento exponencial en los últimos años, la combinación de hardware y software permite procesar, analizar, medir características y parámetros en tiempo real para la toma de decisiones. Dichos sistemas autónomos son muy populares en la actualidad debido a que pueden ser utilizadas en múltiples proyectos sea desde el ámbito industrial hasta el ámbito médico y el de la seguridad estos sistemas nacen con la finalidad de dar solución a diversos problemas que se presentan en la actualidad. Existen diversos algoritmos de visión artificial los cuales tienen una infinidad de aplicaciones entre los cuales está el reconocimiento de objetos en imágenes de video.

La detección de objetos hoy en día tiene muchas utilidades que van desde la toma de decisiones en la conducción de un auto hasta la prevención de ataques terroristas, de igual manera al determinar el rostro de las personas en video ayuda a disminuir la inseguridad y actos delictivos.



Los actuales algoritmos para la detección de objetos en imágenes de video requieren una gran capacidad computacional, en el mercado existen diversos sistemas embebidos que permiten utilizar dichos algoritmos. Uno de estos es el dispositivo llamado Jetson Tx2 del fabricante NVIDIA con el cual se puede probar un potente algoritmo para detección de objetos denominado YoloV3.

En (Peonza & Dominguez, 2017) se realiza una aplicación de reconocimiento de rostros mediante visión artificial con técnicas de redes neuronales artificiales (RNA). Para ello cuentan con dos etapas la primera en la que se usan técnicas de VA que contiene el procesamiento de imágenes y la detección de rostros, la segunda permite el reconocimiento del rostro a través de una RNA. Utilizan una base de datos de 50 imágenes con diferentes tipos de iluminación, gestos, perfiles, diferentes distancias, estas corresponden a 5 personas, las cuales son guardadas de forma matricial. Se utiliza un algoritmo de descenso de gradiente con rampa de aprendizaje adaptativa, debido a la capacidad de anticipación de sucesos pasados, el sistema tiene un umbral del 5% para el reconocimiento, bajo este valor arroja como desconocido, se realizó 15 pruebas en las cuales reconoció 12 imágenes con esto alcanza una confiabilidad del 80%.

En (Ambriz, Eleuterio , & González , 2017), se presenta un prototipo de un sistema autónomo basado en visión por computadora, cuyo fin es la clasificación y recolección de desechos. El sistema embebido utilizado para este fin es la tarjeta de desarrollo integrado Raspberry Pi B+, la cual está basado en sistema operativo Linux, sus características permiten un entorno de hardware esencial para ejecutar algoritmos de VA. El algoritmo

utilizado para detección de colores se basa en el análisis del modelo HSV, mientras que para la detección de formas se implementa el algoritmo Canny. Utilizando estos se programa el sistema autónomo para una exploración de su entorno para clasificar objetos.

## **1.2 Justificación e Importancia**

La visión artificial, aunque ha avanzado mucho en los últimos años, aún está lejos de igualar las capacidades del ojo humano cuando se trata de interpretar el contenido de una escena. Sin embargo, gracias a la miniaturización de la electrónica digital, han surgido sistemas embebidos capaces de procesar imágenes en tiempo real, y suficientemente pequeños para ajustarse a un pequeño prototipo de sistemas autónomos que buscan emular el proceso que realiza el cerebro humano para reconocer, memorizar, aprender ya sea imágenes, objetos, formas, colores, etc. basado en la experiencia de los sucesos transcurridos en la vida diaria y captado por el sentido de la vista (Terven, Salas, & Raducanu, 2013).

La técnica de aprendizaje profundo (deep learning) está basada en la implementación de redes neuronales convolucionales y presenta una gran ventaja frente a otras técnicas holísticas de detección y reconocimientos de objetos en imágenes ya que permite procesar toda la imagen completa sin la necesidad de extraer ciertas características relevantes para el procesamiento de objetos en específico. Este proceso de extracción, filtraje y

clasificación de imágenes lo hacen internamente las capas de la red neuronal para finalmente realizar la interpretación de los resultados obtenidos (Schmidhuber, 2015).

La implementación de algoritmos para visión artificial en sistemas autónomos permite realizar tareas específicas que necesiten procesamiento en tiempo real, los cuales llegan a obtener un costo computacional elevado además de un exceso de consumo energético si se lo implementa en un ordenador convencional.

En la actualidad hay diversos algoritmos computacionales para detección de objetos tales como R-CNN, FAST R-CNN y FASTER R-CNN (Xu, He, & Lan , 2016) que usan la imagen segmentada en miles de regiones para procesar la detección del objeto dentro de la imagen. YOLO (You Only Look Once) es un algoritmo de detección de objetos distinto a los algoritmos anteriores, que utiliza una sola red convolucional para predecir los marcos delimitadores (bounding boxes) y las probabilidades de clase para estos marcos, lo que le hace mucho más rápido y menos costoso que los algoritmos antecesores (Redmon & Farhadi, You Only Look Once: Unified, Real-Time Object Detection, 2016).

Existen diferentes sistemas embebidos entre los cuales podemos mencionar Raspberry Pi, LattePanda entre otros, en ellos podemos implementar diversas técnicas de visión artificial. Los actuales algoritmos para reconocimiento de objetos en imágenes de video en tiempo real requieren una gran capacidad computacional, para este proyecto se utiliza un sistema embebido del fabricante NVIDIA la cual posee un hardware adecuado

para probar algoritmos lo suficientemente rápidos y eficientes para detección y reconocimiento de objetos.

El proyecto nace con el fin de desarrollar un sistema autónomo de detección de objetos utilizando un algoritmo pre-entrenado muy eficiente basado en redes neuronales artificiales el cual se debe configurarlo para detectar objetos específicos en este caso objetos (personas y autos).

### **1.3 Alcance del Proyecto**

Todos los algoritmos antes mencionados requieren una gran capacidad computacional para lo cual es necesario contar con dispositivos electrónicos que cumplan con estos requisitos. El algoritmo a implementar es YoloV3 el cual tiene múltiples ventajas frente a otros algoritmos basados en redes neuronales convolucionales entre las cuales se encuentran mayor fiabilidad para predecir eventos, mayor rapidez y menor costo.

Aunque existen actualmente diversas plataformas de desarrollo en línea tales como los frameworks de Darknet y Google, el reto de este proyecto es implementar un framework propio para implementar un algoritmo de Darknet (YoloV3) en hardware de alto rendimiento (tarjeta Nvidia Jetson Tx2), con el objetivo de detectar personas y autos en video capturado en tiempo real. Los resultados de este proyecto permitirán disponer de un prototipo autónomo independiente de servidores que se utilizará para el posterior desarrollo

de aplicaciones basadas en sistemas de visión artificial tales como seguridad y video vigilancia, robótica, etc. Finalmente se realizarán pruebas para evaluar el desempeño del detector. En efecto, se determinarán tiempos de respuesta, errores de detección y los recursos utilizados en el hardware NVIDIA.

## **1.4 Objetivos**

### **1.4.1 General**

Implementar un prototipo para detectar objetos (autos y personas) en video capturado en tiempo real utilizando técnicas de aprendizaje profundo (deep learning).

### **1.4.2 Específicos**

- Investigar las bases teóricas y prácticas del aprendizaje profundo (Deep learning) aplicados para la detección de objetos en imágenes y video.
- Implementar un detector de objetos con redes neuronales convolucionales (CNN).
- Medir el desempeño del detector implementado en la librería publicada en AlexeyAB/github.

## CAPITULO 2

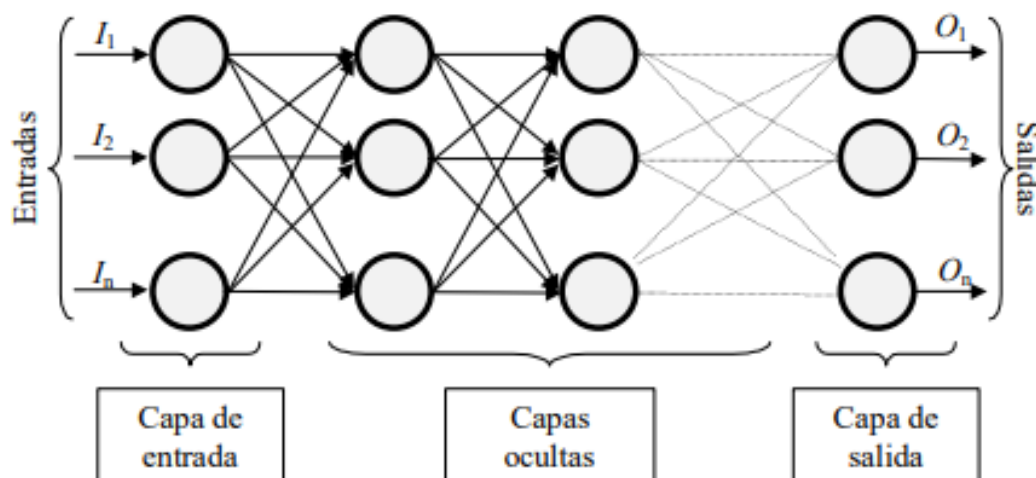
### 2. FUNDAMENTO TEÓRICO

#### 2.1 Redes neuronales

Las redes neuronales artificiales (RNA) son sistemas totalmente conectados que tratan de reproducir características propias del ser humano, como el memorizar, aprender, diferenciar y asociar los hechos actuales con los ocurridos en su aprendizaje, en si las redes neuronales son sistemas simplificados del cerebro humano que tiene la capacidad de adquirir conocimiento a través de la experiencia. Una red neuronal tiene como unidad básica de procesamiento un nodo denominado neurona. (Matich, 2001)

La neurona dentro de un sistema realiza un tipo de actividad simple. Esta consiste en sumar los valores de entrada (inputs) que ingresan en ella, comparar estas cantidades con un valor umbral de prueba, y si lo iguala o lo supera envía un mensaje de activación en la salida (output). Los valores de entrada a la neurona como los valores de salida dependen de los pesos a los que está asociado las conexiones entre neuronas (Montaño Moreno, 2002).

Para entrenar un sistema de RNA en una determinada clasificación de objetos, el mismo debe realizar dos tipos de operaciones. En primer lugar, se selecciona una muestra que represente a dicha clasificación, de todas sus posibles entradas y sus salidas correspondientes. Segundo, un algoritmo es el encargado de ajustar los pesos (valores sujetos a modificación) entre las conexiones de un proceso iterativo de presentación de entradas, observación de salidas y modificación de conexiones (Montaño Moreno, 2002). A continuación, se presenta un esquema de una red neuronal donde  $(I_1, I_2, I_n)$  son valores de entrada a la red los cuales ingresan a través de la capa oculta y salen por la capa de salida hasta finalmente obtener los valores de salida  $(O_1, O_2, O_n)$ .



**Figura 1.** Estructura de una red neuronal artificial.

Fuente: (Matich, 2001)

### 2.1.1 Características de las redes neuronales.

**Aprenden de la experiencia:** Las RNA en base a su entorno es capaz de modificar su comportamiento. Mediante un conjunto d determinadas entradas, las RNA modifican sus valores entre conexiones para producir respuestas compactas. Existe variedad de algoritmos de entrenamiento de redes neuronales en los cuales cada uno presenta ventajas y desventajas para el reconocimiento de objetos.

**Generalizan de ejemplos anteriores a los ejemplos nuevos:** finalizado el entrenamiento de una RNA, la red responde hasta cierto punto de forma imperceptible frente a pequeñas variaciones en las entradas, por lo tanto, es adecuado en el reconocimiento de objetos.

**Abstracción de la esencia de las entradas:** A la entrada de la red neuronal se le introduce un conjunto de imágenes distorsionadas como por ejemplo una letra, una vez finalizado el entrenamiento la RNA es capaz de reconocer correctamente el patrón de la letra en la imagen de entrada, con lo cual la red es capaz de reconocer patrones que jamás ha visto.

## 2.2 Aprendizaje Profundo (Deep Learning)

En los últimos años la tecnología ha dado un gran salto debido a la creación de la inteligencia artificial conocida por sus siglas en inglés AI. Desde el año de 1950 ha tenido un crecimiento exponencial.

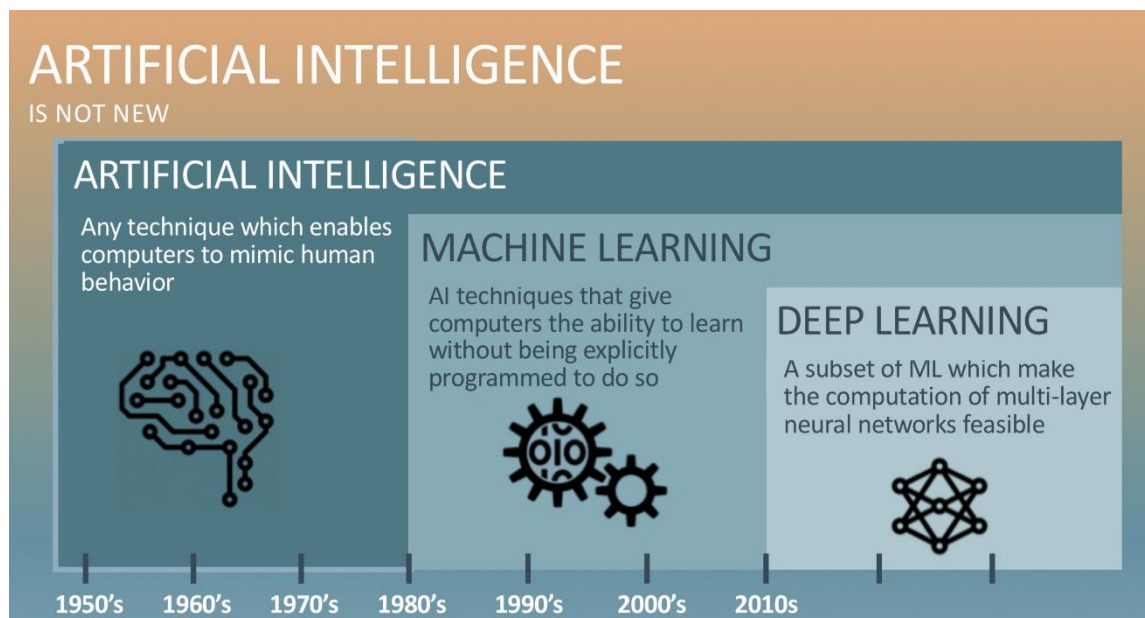


Esta surge de una idea en la cual se plantea que las computadoras puedan pensar y tomar decisiones por si mismas sin necesidad que el humano este supervisando este proceso informático.

Con el nacimiento de la denominada Inteligencia Artificial varios investigadores se pusieron a trabajar en diversos métodos de aprendizaje para computadores, para que estos puedan realizar por si solos los cálculos matemáticos necesarios para realizar tareas específicas sin la supervisión de un humano.

Uno de estos métodos desarrollados es el muy conocido Aprendizaje Automático (Machine Learning) que posee diversas técnicas informáticas las cuales ayudan a los computadores a tener un aprendizaje sin necesidad de que una persona este programándolas.

Del desarrollo del Aprendizaje automático surgió la creación de diversos algoritmos entre los cuales se encuentran algoritmos genéticos, arboles de decisión, redes neuronales, aprendizaje por refuerzo y uno de los más utilizados en la actualidad el aprendizaje profundo conocido como “Deep Learning”.



*Figura 2.* Historia del nacimiento de la Inteligencia Artificial

Fuente: (Oracle, 2018)

En la figura 2 se observa que el aprendizaje profundo es una parte del aprendizaje automático y este a su vez de la inteligencia artificial.

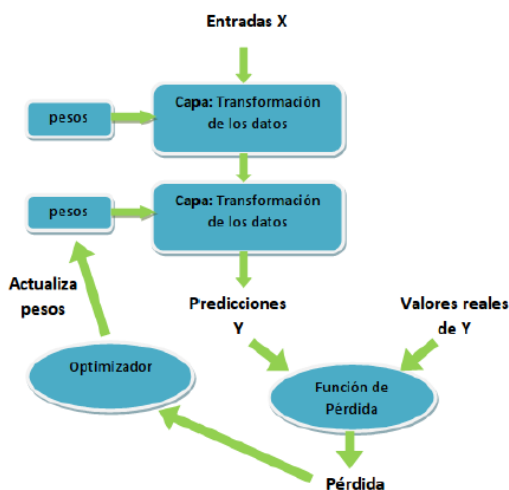
### 2.2.1 Estructura Aprendizaje Profundo

El aprendizaje profundo realiza un mapeo de datos al objeto de interés con ayuda de una red neuronal artificial. Su estructura es jerarquizada es decir subdividida en capas.

La primera capa extrae datos y aprende procesos básicos y envía esta información a una capa superior. Esta capa toma la información básica la combina con operaciones más complejas, una vez finalizado este procedimiento envía esta información a una capa superior a esta. Y así

sucesivamente hasta que la información llegue a las capas de salida, de esta manera la red aprende con la información recopilada en cada capa anterior (Lopez, 2017).

La información recibida por cada capa es almacenada en los pesos de las mismas, básicamente son números. En términos técnicos la transformación de datos que ocurre en las capas es parametrizada por sus pesos. Se necesita los pesos de todas las capas de la red para que se pueda realizar un mapeo entre las entradas y las salidas, de esta manera es como la red aprende. La red al poseer varios parámetros no está exenta de errores debido a que si un valor falla este repercute en el funcionamiento de toda la red. Para solucionar este problema se debe controlar la salida de la red verificando si es la información que se desea obtener.



**Figura 3.** Estructura Deep Learning  
Fuente: (Lopez, 2017)

En la figura 3 se muestra el proceso que realiza Deep Learning para aprender. En el presente trabajo se trabajará con una red neuronal para el aprendizaje en específico se utilizará una red neuronal convolucional (CNN).

### **2.3 Redes Neuronales Convolucionales (CNN).**

Las redes neuronales convolucionales son muy similares a las redes neuronales ordinarias. Las neuronas tienen pesos, valores propios y característicos estos reciben una entrada con la que realiza un producto escalar y en ella se le aplica una función de activación, además estas redes poseen una función de pérdida en la última capa. Sin embargo, su principal uso es para resolver el gran problema que tienen las redes neuronales ordinarias: el tratamiento de imágenes. Pese a que con las redes neuronales estándar es posible trabajar las imágenes, en cuanto las estas aumentan su tamaño, resolución y calidad esta se vuelve imposible de tratar. Por ser redes de neuronas totalmente conectadas, se conseguiría un desperdicio de recursos, así como un rápido sobreajuste (Cortés, 2017).

Las redes neuronales convolucionales su función es dividir y modelar la información en partes más pequeñas, y combinando esta información en las capas más profundas de la red. Por ejemplo, para el tratamiento de una imagen, las primeras capas buscarían detectar los bordes de las figuras. Las siguientes capas tratan de combinar los patrones de detección de bordes con lo cual consiguen formas más simples, para la aplicación de patrones de posición de objetos,

iluminación. Para finalizar, en las últimas capas se pretende que la imagen coincida con todos los patrones antes descubiertos, para conseguir con ella una predicción final de la suma de todas estas. De esta forma las redes neuronales convolucionales tienen la capacidad de modelar muchos datos, utilizando la división del problema en partes pequeñas con el fin de conseguir predicciones más sencillas y precisas (Cortés, 2017).

Las redes neuronales convolucionales consta de 3 capas las cuales se detallan a continuación.

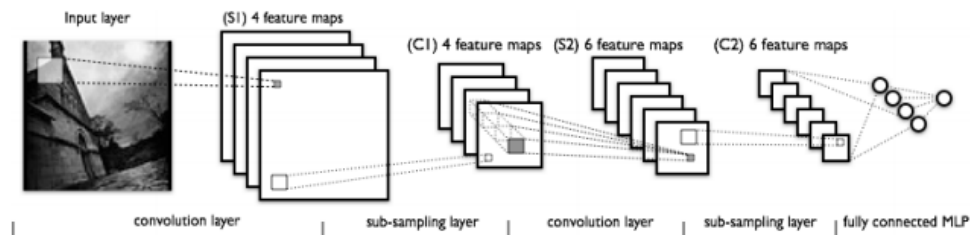
### **2.3.1 Capa Convolutiva**

La capa convolutiva a diferencia de las redes estándares que utiliza la multiplicación de matrices, esta utiliza una operación llamada convolución. El funcionamiento consiste en lo siguiente; recibe una imagen como entrada, sobre esta se aplica un filtro el cual devuelve un mapa de características el cual reduce el tamaño de los parámetros. La convolución realiza tres operaciones que ayudan notablemente a mejorar cualquier sistema sobre el que se aplique aprendizaje automático. (Cortés, 2017).

- **Interacciones dispersas:** Al aplicar un filtro de menor tamaño sobre la entrada, reducimos bastante el número de parámetros y cálculos.
- **Parámetros compartidos:** Compartir parámetros entre los distintos tipos de filtros ayuda a mejorar la eficiencia del sistema.

- Representaciones equivalentes: Si las entradas cambian, las salidas cambian de forma similar.

En la figura 4 se muestra el funcionamiento de la capa convolucional.



**Figura 4.** Funcionamiento de una capa convolucional.

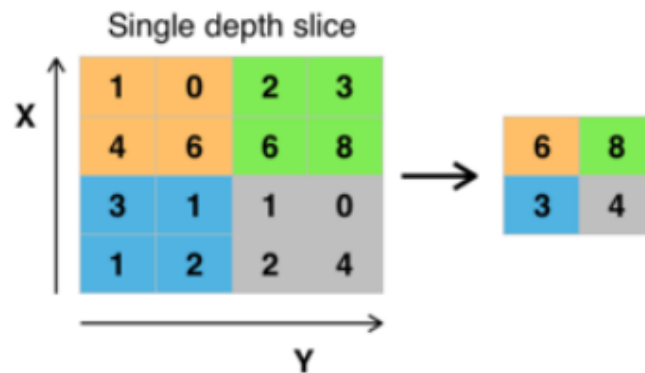
Fuente: (Cortés, 2017)

Además, la convolución tiene herramientas que le permite trabajar con entradas de tamaño variable, debido a que las imágenes tienen distinto número de píxeles es muy útil al trabajar con imágenes.

### 2.3.2 Capa de reducción o pooling

Esta capa es aquella que se ubica después de la capa convolucional. Y es la que permite reducir el número de parámetros que se va a analizar, esto se lo logra achicando las dimensiones espaciales (ancho x alto), para tener las características más comunes. La operación que se realiza tiene el nombre de reducción de muestreo o sub muestreo, la disminución de tamaño conlleva una reducción de información. Sin embargo, para una red neuronal, esta reducción resulta ser beneficioso debido a: La disminución del tamaño produce una menor sobrecarga de cálculo en

las capas que están próximas a la red. Reduce habitualmente el sobreajuste. La operación que generalmente se aplica en esta capa se llama "max-pooling", que se encarga de dividir la imagen de entrada en una serie de rectángulos, y respecto a cada uno de ellos, se queda con el valor más alto (Cortés, 2017).



**Figura 5.** Max-pooling aplicado a una imagen  
Fuente: (Cortés, 2017)

### 2.3.3 Capa clasificadora totalmente conectada

Cuando la imagen ha trascendido por las capas convolucionales y las de pooling, se ha podido separar sus características más destacadas, los datos llegan a la fase de clasificación. Para este objetivo, las redes convolucionales utilizan generalmente capas completamente conectadas en las que cada píxel se trata como una neurona independiente. Las neuronas en esta fase funcionan como las de un perceptrón multicapa, en las cuales la salida de cada neurona es calculada mediante la multiplicación de la salida de la capa anterior con el peso de la conexión, y aplicando a este resultado una función de activación (Cortés, 2017).

## 2.4 Algoritmos de detección de objetos

Los algoritmos detectores de objetos presentan la cualidad de intentar dibujar un marco delimitador alrededor del objeto que se pretende ubicar dentro de un frame o cuadro de imagen. Además, existe la posibilidad de que no dibuje un solo marco delimitador para una detección de objetos, a su vez podría tener muchos marcos delimitadores que representan varios objetos de interés en la imagen procesada y no sabría cuántos de antemano.

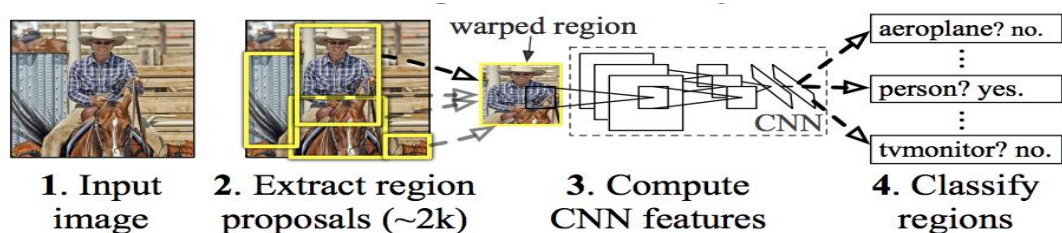
El principal inconveniente al realizar una red neuronal convolucional estándar seguida de una capa totalmente conectada radica en que la longitud de la capa de salida es variable, esto es principalmente a que la cantidad de ocurrencias de los objetos de interés es no arreglada. Una posible solución para dar solución a este problema sería tomar diferentes regiones de interés dentro de la imagen y utilizar una CNN para clasificar la presencia del objeto dentro de esa región. El problema con esta solución es que los objetos de interés están ubicados en diferentes regiones de la imagen y diferentes relaciones de aspecto. Por lo tanto, tendría que seleccionar una gran cantidad de regiones y esto desbordaría computacionalmente. Por lo tanto, los algoritmos como R-CNN, YOLO, etc., se han desarrollado para encontrar estas ocurrencias rápidamente.



### 2.4.1 R-CNN

Para no presentar el problema de seleccionar una gran cantidad de regiones, Ross Girshick, realizo un método para que la búsqueda selectiva solo pueda extraer 2000 regiones de la imagen y las denomino propuestas de región. De esta manera, no se necesita clasificar muchas regiones, ya que con la ayuda de esta solución únicamente se trabaja con 2000 regiones. Estas propuestas de 2000 regiones son las que se utilizan en el algoritmo de búsqueda selectiva (Girshick, Donahue, & Darrell, Rich feature hierarchies for accurate object detection and semantic segmentation, 2014).

En la figura 6 se muestra el funcionamiento del algoritmo de búsqueda selectiva, en primer lugar, se realiza una subsegmentación, con lo cual se genera diversas regiones candidatas, posterior en segundo paso se utiliza un algoritmo para combinar recursivamente regiones similares en otras más grandes y en tercer lugar, usa las regiones generadas para crear las propuestas de la región candidata final.



**Figura 6.** Método de búsqueda selectiva

Fuente: (Girshick, Donahue, & Darrell, Rich feature hierarchies for accurate object detection and semantic segmentation, 2014)

Estas propuestas de la región candidata de 2000 se convierten en un cuadrado e ingresan a una red neuronal convolucional que produce un vector de características de 4096 dimensiones como salida. La CNN permite extraer características, mientras que la capa de salida, posee las características de la imagen que fueron extraídas, las cuales entran a un algoritmo Support Vector Machine (SVM) para la clasificación del objeto dentro de esa propuesta de región candidata. Además de predecir la existencia de un objeto dentro de las propuestas de región, este algoritmo indica una serie de valores de desplazamiento para aumentar la precisión del marco encargado de delimitar la imagen. Por ejemplo, si se tiene una propuesta de región, el algoritmo es capaz de predecir la existencia de una persona, a su vez el rostro de esa persona dentro de esa propuesta de región es probable que se reduzca a la mitad. Debido a esto, los valores de compensación permiten realizar un ajuste al marco delimitador de la propuesta de la región. (Girshick, Donahue, & Darrell, Rich feature hierarchies for accurate object detection and semantic segmentation, 2014)

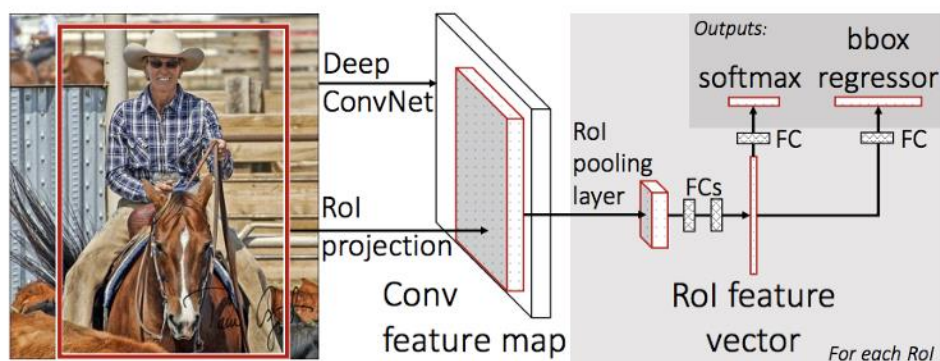
### **Problemas con R-CNN**

- Este algoritmo utiliza gran cantidad de tiempo para preparar a la red, ya que tiene que clasificar las propuestas de 2000 regiones por imagen.
- No puede ser implementada en tiempo real, debido a que esta necesita alrededor de 47 segundos para cada imagen de prueba.

- El algoritmo de búsqueda selectiva es un algoritmo fijo. Con lo cual, ningún aprendizaje está ocurriendo en esa etapa. Esto podría llevar a generar una serie de malas propuestas para la región candidata.

## 2.4.2 R-CNN Rápido (Fast R-CNN)

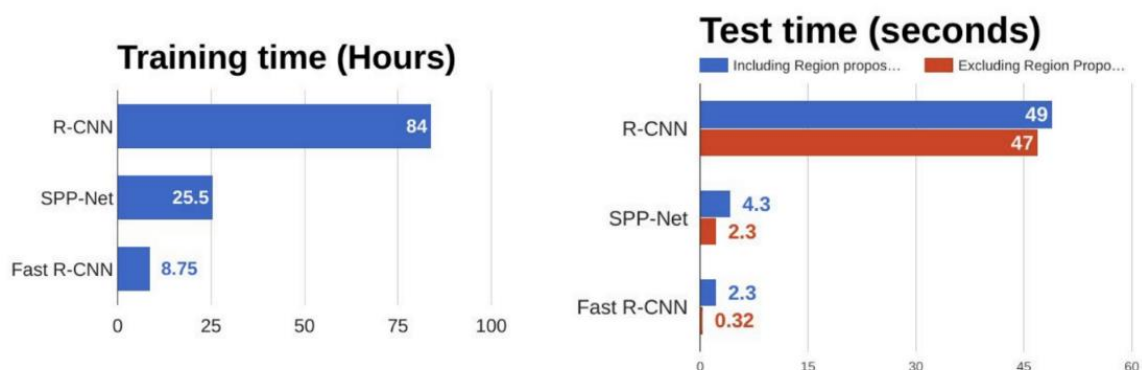
El algoritmo Fast R-CNN tiene un similar funcionamiento al R-CNN. Pero, en vez de enviar las propuestas de región a la CNN, este ingresa la imagen a la CNN para generar un mapa de características convolucionales. Con este mapa, se puede determinar la región de las propuestas las cuales se transforman en cuadrados y al aplicar una capa de agrupación RoI, las transformamos a un tamaño fijo para que pueda ingresar a una capa totalmente conectada. Partiendo del vector de características generado por la capa RoI, luego se emplea una capa softmax que ayuda en la predicción de la región propuesta como también los valores de desplazamiento para el marco delimitador (Girshick, Fast R-CNN, 2015).



**Figura 7.** Funcionamiento del algoritmo Fast R-CNN

Fuente: (Girshick, Fast R-CNN, 2015)

La particularidad del algoritmo "Fast R-CNN" por lo cual es más rápida que R-CNN es debido a que no envía de 2000 propuestas de regiones a la red neuronal convolucional en todo momento. Por lo tanto, la operación de convolución se efectúa solo una vez por imagen generando un mapa de características a partir de ella.



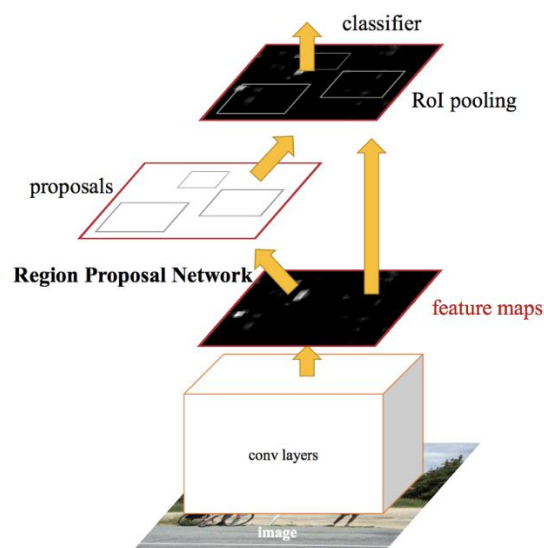
**Figura 8.** Comparación de algoritmos R-CNN, Fast R-CNN  
Fuente: (Girshick, Fast R-CNN, 2015)

En la figura 8 se observa que el algoritmo Fast R-CNN es significativamente más rápido que R-CNN dentro de las sesiones de entrenamiento como en los tiempos de prueba.

### 2.4.3 R-CNN más rápido (Faster R-CNN)

Mientras que los algoritmos (R-CNN y Fast R-CNN) utilizan la búsqueda selectiva para conocer las propuestas de la región. Esto es un proceso lento que se refleja en el rendimiento de la red. Por lo tanto, Shaoqing Ren realizó un algoritmo de detección de objetos que elimina la

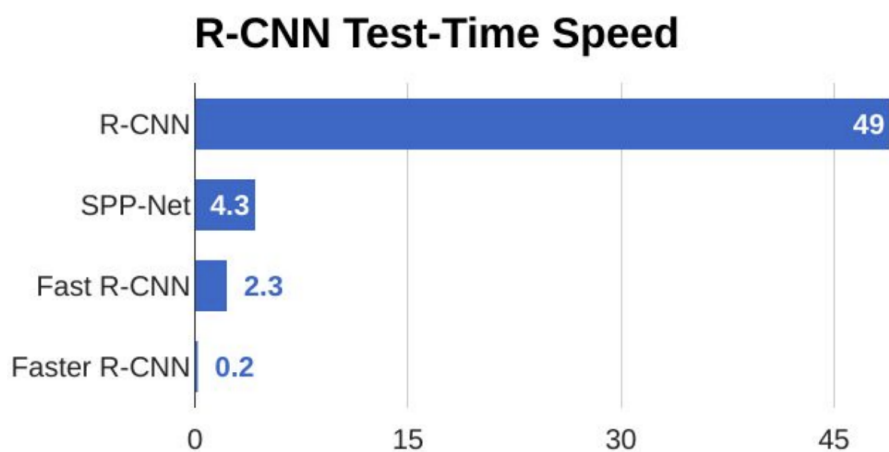
búsqueda selectiva y proporciona a la red un aprendizaje de las propuestas de la región (Ren, He, & Girshick, 2016).



**Figura 9.** Algoritmo Faster R-CNN

Fuente: (Ren, He, & Girshick, 2016)

En la figura 9 se indica cómo funciona el algoritmo Faster R-CNN, la imagen entra en una red convolucional la cual suministra un mapa de características convolucionales, y no utiliza la búsqueda selectiva en el mapa de características en el reconocimiento de las propuestas de la región, este ocupa una red separada y así realiza la predicción las propuestas de la región. Estas propuestas de región predecidas se reúnen mediante una capa RoI que se utiliza para clasificar la imagen dentro de la región propuesta y pronosticar valores de desplazamiento para los marcos delimitadores.



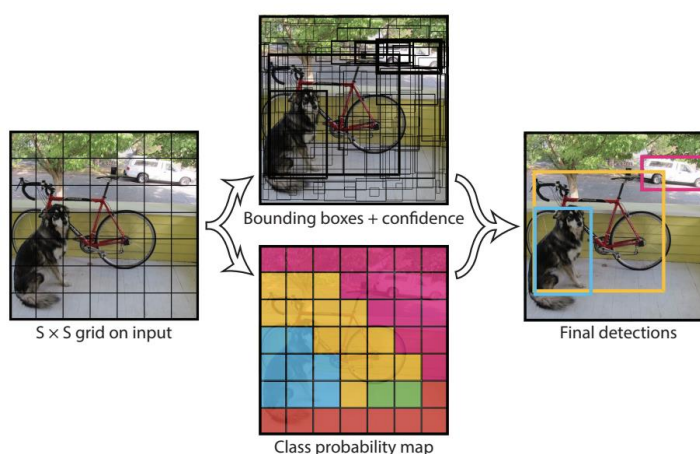
**Figura 10.** Comparación de prueba de velocidad de algoritmos  
Fuente: (Ren, He, & Girshick, 2016)

En la figura 10, el algoritmo faster R-CNN es el más rápido en comparación a los algoritmos antes descritos, por lo cual se puede decir que puede ser utilizado para detección de objetos en tiempo real.

#### 2.4.4 YOLO (You Only Look Once)

Todos los algoritmos que se usan para la detección de objetos descritos anteriormente, emplean regiones para localizar un objeto de interés dentro de la imagen. YOLO está diseñado como un sistema detector de objetos en tiempo real, este utiliza deep learning y redes neuronales convolucionales, la red YOLO no observa la imagen completa, sino que esta mira partes de la misma en donde se encuentran grandes probabilidades de contener el objeto (Redmon, Divvala, & Girshick, You Only Look Once., 2016).

En la figura 11 se observa el modelo del algoritmo You Only Look Once (YOLO), el mismo que tiene una única red convolucional, la cual es la encargada de predecir los marcos delimitadores y las probabilidades de clase para estos marcos.



**Figura 11.** Funcionamiento del algoritmo YOLO  
Fuente: (Redmon, Divvala, & Girshick, You Only Look Once., 2016)

YOLO trabaja ocupando una imagen y la fracciona en una cuadrícula  $S \times S$ , dentro de cada una de las cuadrículas tomamos  $m$  marcos de delimitación. Para cada uno de los marcos delimitadores, la red produce una probabilidad de clase y valores de desplazamiento para el marco delimitador. Los marcos delimitadores que poseen la probabilidad de clase por arriba de un valor umbral se seleccionan y se emplean para encontrar el objeto dentro de la imagen.

Los marcos utilizados en la delimitación de objetos realizan la predicción utilizando clústers que constan de 4 coordenadas para cada uno de ellos,  $t_x$ ,  $t_y$ ,  $t_w$ ,  $t_h$ . Para identificar la

celda desplazada desde la esquina superior izquierda hay que tener en cuenta  $c_x$ ,  $c_y$  con un ancho y alto descritos por  $p_w$ ,  $p_h$ , de tal forma que las predicciones responden a:

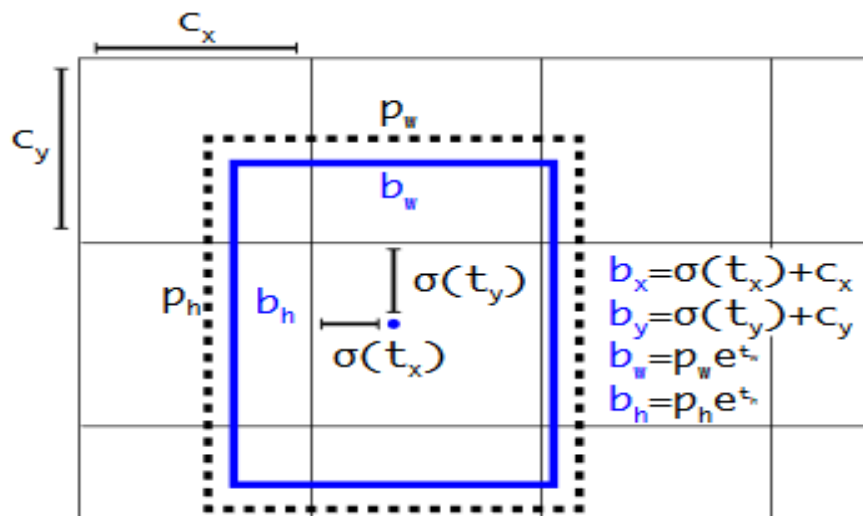
$$b_x = \sigma(t_x) + c_x \quad (1)$$

$$b_y = \sigma(t_y) + c_y \quad (2)$$

$$b_w = p_w e^{t_w} \quad (3)$$

$$b_h = p_h e^{t_h} \quad (4)$$

Para la etapa de entrenamiento se trabaja con la suma de la pérdida del error al cuadrado con una puntuación de objetividad para cada marco delimitador. Para el entrenamiento se usa la regresión lógica la cual predice la puntuación de objetividad, con lo cual si es 1 el marco delimitador se llena con un objeto verdadero antes que otro marco anterior. Si no es el mejor marco, pero se sobrepone a un objeto por arriba del umbral no se realiza la predicción.



**Figura 12.** Recuadro delimitador.

Fuente: (Redmon, Divvala, & Girshick, You Only Look Once., 2016)



En la figura 12. Se observa un marco delimitador, en el cual se ha encontrado el ancho y la altura, de igual manera se predice las coordenadas del centro del marco, esto en relación a la ubicación del filtro, todo esto gracias a la función  $\sigma$  (Redmon, Divvala, & Girshick, You Only Look Once, 2016).

YOLO es más rápido que otros algoritmos de detección de objetos ya que puede reconocer objetos a 45 fps (frames por segundo). El principal inconveniente del algoritmo YOLO es su baja probabilidad de aciertos con objetos pequeños dentro de la imagen, por ejemplo, presentaría dificultades para detectar una bandada de pájaros. Esto se debe a las restricciones espaciales del algoritmo.

## CAPITULO 3

### 3. MATERIALES Y MÉTODOS

#### 3.1 Introducción

En el presente capítulo se detallan las características técnicas del sistema embebido Nvidia Jetson Tx2 y el algoritmo YOLOv3 implementado bajo la estructura de la red neuronal convolucional.

#### 3.2 Hardware Nvidia Jetson Tx2

Jetson Tx2 es un dispositivo informático GPU de la familia NVIDIA Pascal™ con 8GB y 59.7GB / s de ancho de banda de memoria RAM. Cuenta con una variedad de interfaces de hardware estándar que facilitan su integración en una amplia gama de productos y factores de forma (Nvidia, 2019).



**Figura 13.** Kit de desarrollo Jetson Tx2.  
Fuente: (Nvidia, 2019)

### 3.2.1 Especificaciones técnicas del módulo

**Tabla 1**

*Características del hardware Jetson Tx2.*

<b>GPU</b>	NVIDIA Pascal™, 256 núcleos CUDA
<b>UPC</b>	HMP Dual Denver 2/2 MB L2 + Quad ARM® A57 / 2 MB L2
<b>Video</b>	Codificación 4K x 2K 60 Hz (HEVC) Decodificación 4K x 2K 60 Hz (compatibilidad con 12 bits)
<b>Memoria</b>	8 GB 128 bit LPDDR4 59.7 GB / s
<b>Monitor</b>	2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4
<b>CSI</b>	Hasta 6 cámaras (2 carriles) CSI2 D-PHY 1.2 (2.5 Gbps / carril)
<b>PCIE</b>	Gen 2   1x4 + 1x1 o 2x1 + 1x2
<b>Almacenamiento de datos</b>	EMMC de 32 GB, SDIO, SATA
<b>Otro</b>	CAN, UART, SPI, I2C, I2S, GPIOs
<b>USB</b>	USB 3.0 + USB 2.0
<b>Conectividad</b>	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth
<b>Mecánico</b>	50 mm x 87 mm (conector de placa a placa compatible con 400 pines)

Fuente: (Nvidia, 2019)

### 3.2.2 Paquete de desarrollo Jetpack

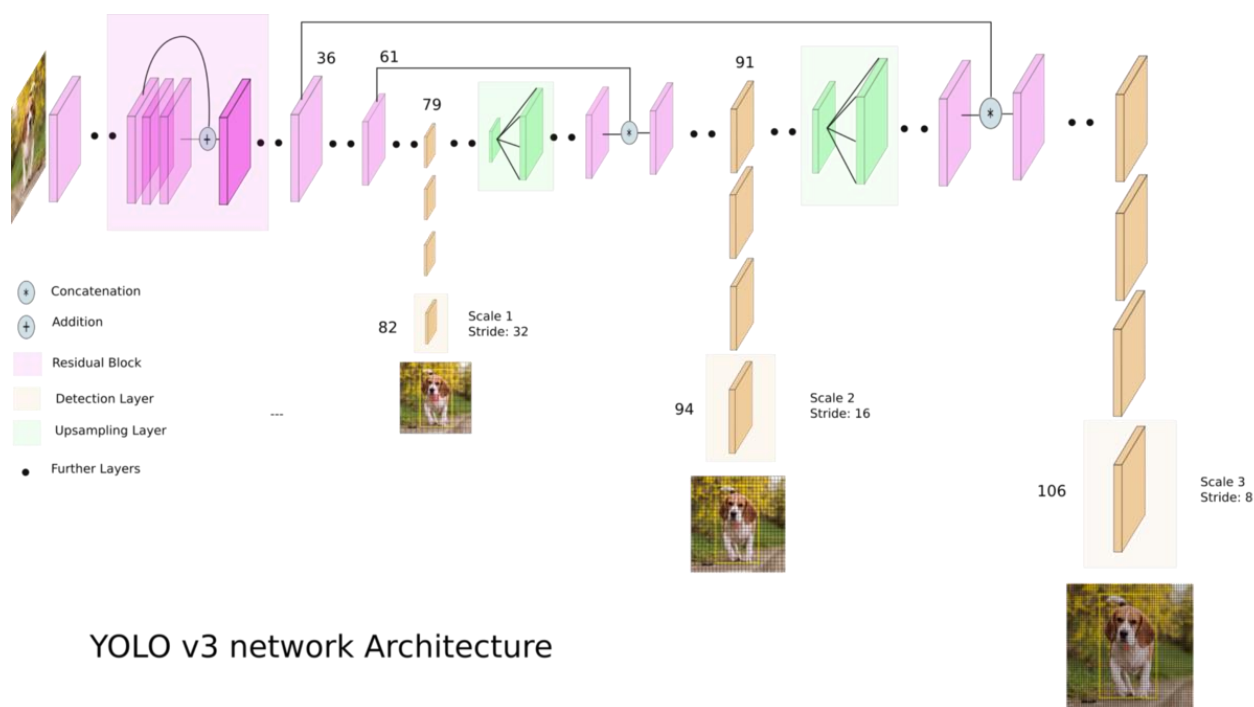
El Jetson Development Pack (JetPack) es un paquete que agrupa e instala todas las herramientas de software necesarias para el funcionamiento de la tarjeta Jetson Tx2:

- **Aprendizaje profundo:** TensorRT, cuDNN, flujo de trabajo NVIDIA DIGITS™
- **Visión por Computador:** NVIDIA VisionWorks, OpenCV
- **GPU Compute:** NVIDIA CUDA, CUDA Libraries
- **Multimedia:** soporte de ISP, imágenes de cámara, video CODEC

También incluye compatibilidad con ROS, OpenGL y herramientas avanzadas para desarrolladores.

### 3.3 YoloV3

YOLO v3 usa una modificación de Darknet, la cual inicialmente posee una red de 53 capas entrenadas en Imagenet. Para la detección, 53 capas adicionales se apilan en ella, lo que resulta en una arquitectura subyacente totalmente convolucional de 106 capas para YOLO v3. Esta es la causa principal que ralentiza a YOLO v3 (30 FPS), en comparación con YOLO v2 (45 FPS). Así es como se observa la arquitectura interna de YOLO ahora (Kathuria, 2018).



YOLO v3 network Architecture

**Figura 14.** Arquitectura de red Yolo V3

Fuente: (Kathuria, 2018)

En la figura 14 se muestra la arquitectura de red, la cual tiene conexiones de omisión de residuos y de muestreo. La característica más relevante de la versión 3 de YOLO es que efectúa detecciones en tres escalas diferentes. YOLO es una red completamente convolucional y su salida final se genera al aplicar un kernel  $1 \times 1$  en un mapa de características. En YOLO v3, la detección se produce aplicando núcleos de detección  $1 \times 1$  en mapas de características de tres tamaños diferentes en tres posiciones diferentes dentro de la red.

### 3.4.1 Detección a tres escalas.

La fórmula matemática del núcleo de detección es  $1 \times 1 \times (B \times (5 + C))$ . Donde B indica el número de marcos delimitadores que es capaz de pronosticar una celda en el mapa de

características, "5" es el número que indica los 4 atributos del marco delimitador y la confianza de acierto en la detección de un objeto, y C muestra el número de clases. En YOLO v3 entrenado en COCO,  $B = 3$  y  $C = 80$ , por lo que el tamaño del kernel es  $1 \times 1 \times 255$  (Kathuria, 2018).

YOLO v3 realiza la predicción a tres escalas diferentes, que aparecen únicamente al disminuir el muestreo de las dimensiones de la imagen de entrada en 32, 16 y 8 respectivamente.

La primera detección se produce en la capa 82. Para las primeras 81 capas, la imagen no está muestreada por la red, por lo tanto, la capa 81 tiene un paso de 32. Si poseemos una imagen de  $416 \times 416$ , el mapa de características resultante tendría un tamaño de  $13 \times 13$ . Una detección se produce mediante el kernel de detección  $1 \times 1$ , lo que nos da un mapa de características de detección de  $13 \times 13 \times 255$  (Kathuria, 2018).

Posterior a esto, el mapa de características presente en la capa 79 pasa por un número pequeño de capas convolucionales antes de muestrearlo hasta 2 veces a las dimensiones de  $26 \times 26$ . Este mapa de características se enlaza con el mapa de características de la capa 61. Luego, los mapas de características combinados nuevamente se someten a unas pocas capas convolucionales  $1 \times 1$  para unir las características de la capa anterior (61). Luego, la segunda detección se lo hace mediante la capa 94, lo que produce un mapa de características de detección de  $26 \times 26 \times 255$  (Kathuria, 2018).

Como siguiente paso se realiza un procedimiento semejante, en el cual al mapa de características de la capa 91 pasa por unas pocas capas convolucionales antes de ser concatenado en profundidad con un mapa de características de la capa 36. De igual forma, algunas capas convolucionales de 1 x 1 siguen para unir la información de la anterior capa (36). La detección final se lo realiza en la capa 106, elaborando un mapa de características de tamaño 52 x 52 x 255 (Kathuria, 2018).

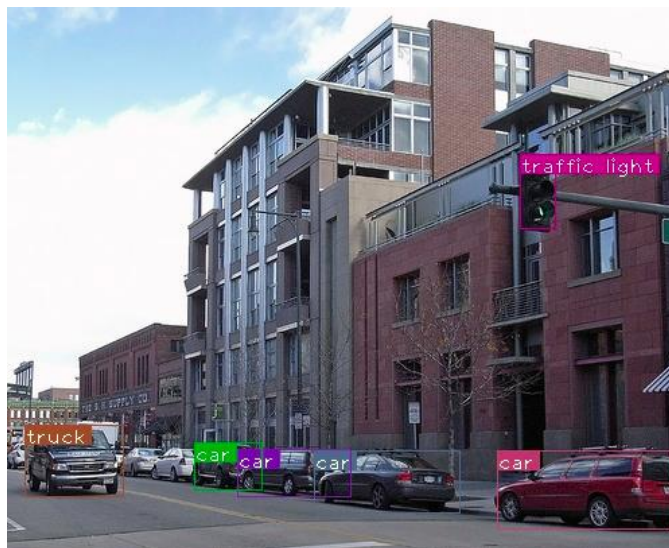
La capa de 13 x 13 es aquella que detecta objetos grandes, a su vez la capa de 52 x 52 detecta los objetos más pequeños, mientras que la capa de 26 x 26 detecta objetos medianos. Se presenta un análisis comparativo de algunos objetos seleccionados en la misma imagen por diferentes capas (Kathuria, 2018).

YOLO v3, en total tiene 9 cajas de anclaje. Tres para cada escala. Los anclajes están asignados de la siguiente forma tres anclajes más grandes para la primera escala, los tres siguientes para la segunda escala y los tres últimos para la tercera escala.

YOLO v3 predice marcos a 3 escalas diferentes. Para la misma imagen de 416 x 416:

- Detección de la primera escala

```
python detect.py --scales 1 --images imgs / img3.jpg
```



**Figura 15.** Detección a escala 1  
Fuente: (Kathuria, 2018)

En la figura 15, se prueba el algoritmo en la escala 1 con la cual se detecta los objetos grandes, o los objetos que se encuentran más cercanos a la cámara.

- Detección de la segunda escala.

```
python detect.py --scales 2 --images imgs / img3.jpg
```



**Figura 16.** Detección a escala 2  
Fuente: (Kathuria, 2018)



En la figura 16, se prueba la detección a escala 2 en la cual no se detecta ningún objeto.

- Detección de la tercera escala.

```
python detect.py --scales 3 --images imgs / img3.jpg
```



**Figura 17.** Detección a escala 3.

Fuente: (Kathuria, 2018)

En la figura 17, se prueba la detección a escala 3 en la cual se observan los objetos más pequeños, es decir los más alejados de la cámara.

## CAPITULO 4

### 4. IMPLEMENTACIÓN DEL PROTOTIPO DE SISTEMA AUTÓNOMO DE VISIÓN ARTIFICIAL

#### 4.1 Instalación de JetPack 3.2 en la tarjeta Jetson Tx2

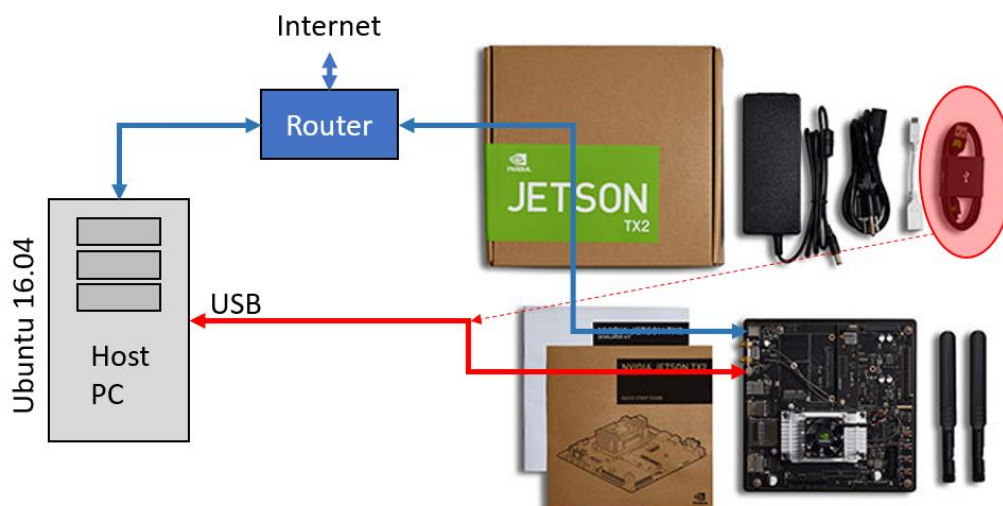
Para poder arrancar la tarjeta Jetson Tx2 debemos flashear la misma con el JetPack 3.2 disponible en la página de NVIDIA. Al instalar el Pack obtenemos herramientas tanto en la computadora anfitriona como en el módulo Jetson, entre las cuales se encuentran.

- Linux para Tegra r27.1 (vista previa de desarrolladores)
- TensorRT 1.0
- cuDNN 5.1
- VisionWorks 1.6
- CUDA 8.0
- API multimedia

Para la instalación se deben cumplir ciertos requisitos como los siguientes:

- Disponer de una PC o Laptop con sistema operativo Linux Ubuntu 16.04 o mayor.
- Disponer de una Tarjeta Jetson TX2 conectada a un monitor, teclado y mouse.
- Un router con conexión a internet.

Los elementos se interconectan como se muestra en la figura siguiente:



**Figura 18.** Estructura física para la instalación.

Fuente: (Nicholas, 2017)

Procedimiento:

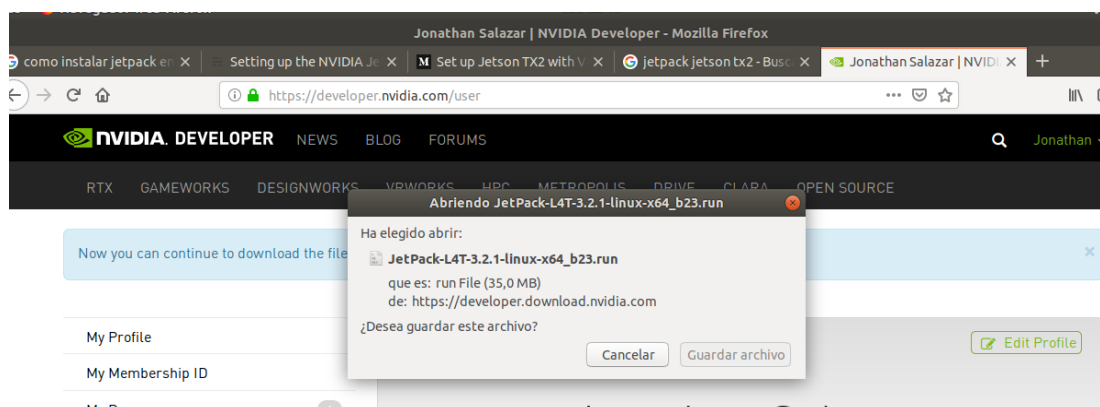
Paso 1: (Host)

Registrarse en el sitio web de NVIDIA, luego descargar el JetPack 3.2

La imagen muestra una captura de pantalla del sitio web de NVIDIA. El encabezado incluye el logo de NVIDIA y el texto 'AUTONOMOUS MACHINES'. El menú de navegación contiene 'Develop', 'Downloads', 'Community', 'Learn', 'SDKs', 'Buy', 'Join' y 'Login'. El contenido principal muestra el título 'JetPack 3.2.1 Release Notes' y un texto que describe el SDK. Debajo del texto hay tres botones: 'Download', 'Install Guide' y 'Documentation'. Una nota al pie indica que se requiere un Host PC con Ubuntu para ejecutar el instalador.

**Figura 19.** Jetpack versión 3.2.

Fuente: (Nvidia, 2019)



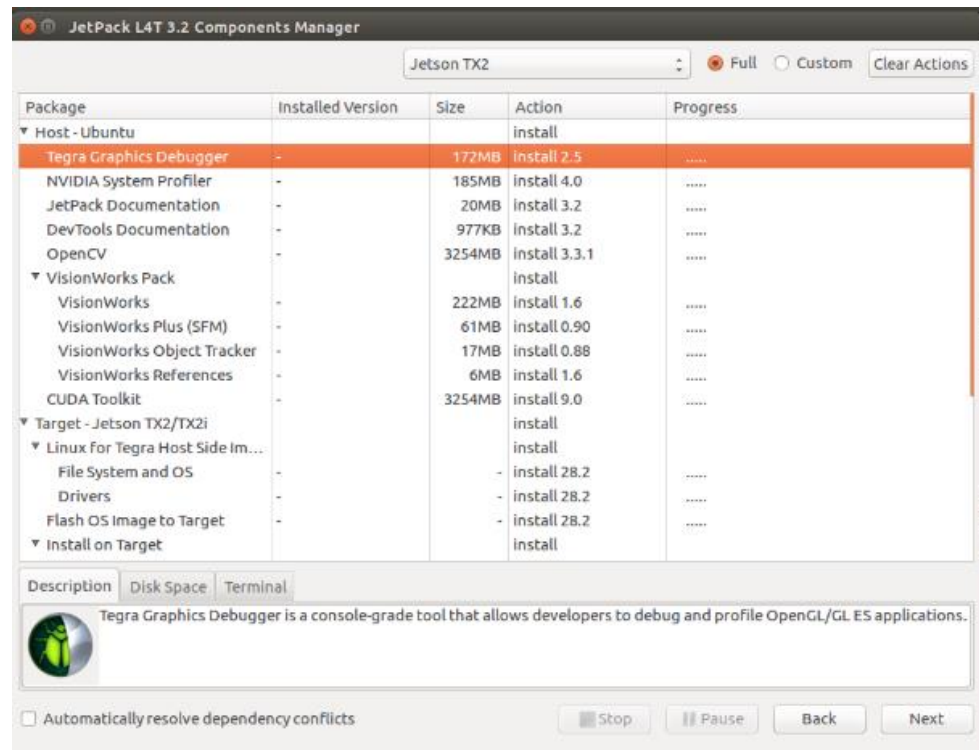
**Figura 20.** Archivo. run  
Fuente: (Nvidia, 2019)

Una vez finalizada la descarga del instalador Jetpack 3.2 procedemos a ejecutar los siguientes comandos en la ventana terminal de Ubuntu.

```
dell@dell-Inspiron-14-3467: ~/Descargas
Archivo Editar Ver Buscar Terminal Ayuda
dell@dell-Inspiron-14-3467:~$ cd Descargas
dell@dell-Inspiron-14-3467:~/Descargas$ ls
JetPack-L4T-3.2.1-linux-x64_b23.run
dell@dell-Inspiron-14-3467:~/Descargas$ chmod +x JetPack-L4T-3.2.1-linux-x64_b23
.run
dell@dell-Inspiron-14-3467:~/Descargas$ ls
JetPack-L4T-3.2.1-linux-x64_b23.run
dell@dell-Inspiron-14-3467:~/Descargas$ ./JetPack-L4T-3.2.1-linux-x64_b23.run
Creating directory _installer
Verifying archive integrity... All good.
Uncompressing JetPack 100%
```

**Figura 21.** Comandos para ejecutar el instalador  
Fuente: Elaborado por el autor

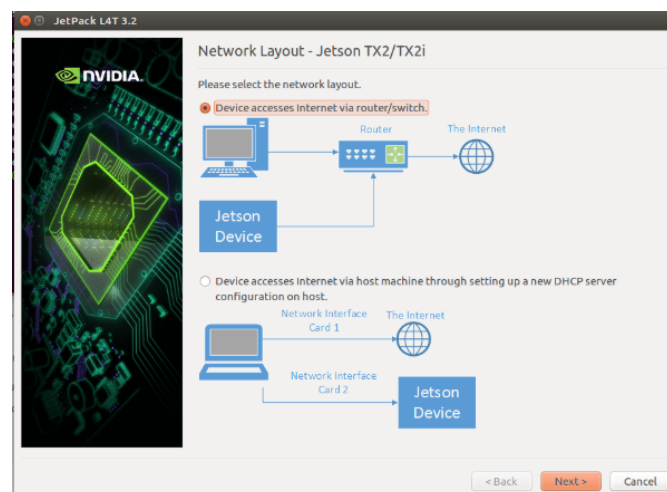
Una vez ejecutado los comandos procede la instalación en la cual debemos seguir todas las instrucciones que se nos presenten.



**Figura 22.** Instalación de componentes en el host.

Fuente: Elaborado por el autor

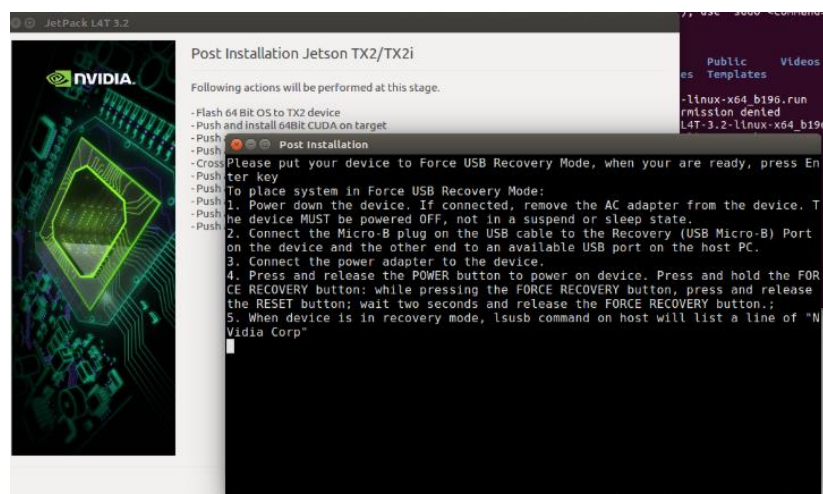
Al finalizar de descargar e instalar los componentes necesarios en el host es hora de proceder a flashear la tarjeta Jetson Tx2.



**Figura 23.** Network layout.

Fuente: Elaborado por el autor

Una vez dado click en next debemos seguir todas las indicaciones que nos muestra en pantalla.



**Figura 24.** Post Instalación.

Fuente: Elaborado por el autor

Al llegar a este punto debemos conectar el cable micro USB que viene incluido en el equipo un extremo en la tarjeta y el otro en el host.



**Figura 25.** Conexión cable micro usb.

Fuente: Elaborado por el autor

Luego procedemos a conectar el cable de poder para energizar la tarjeta. Seguido procedemos a presionar los siguientes botones.



**Figura 26.** Combinación de botones en la tarjeta  
Fuente: Elaborado por el autor

Finalizado estos pasos abrimos una nueva terminal y escribimos el comando **lsusb** y estamos listos para proceder a flashear la Jetson Tx2. Con esto se cargarán todos los componentes necesarios en la tarjeta Nvidia.

#### **4.2 Descarga de la red neuronal YOLO en la tarjeta Jetson TX2**

Actualmente hay 3 implementaciones del algoritmo cada una con sus ventajas y desventajas entre las cuales se encuentran.

- Darknet: esta es la implementación oficial de YOLO.
- AlexeyAB/Darknet: está basado en Darknet y su uso es tanto para Windows como para Linux.
- Darkflow: Es el port de Darknet a TensorFlow.

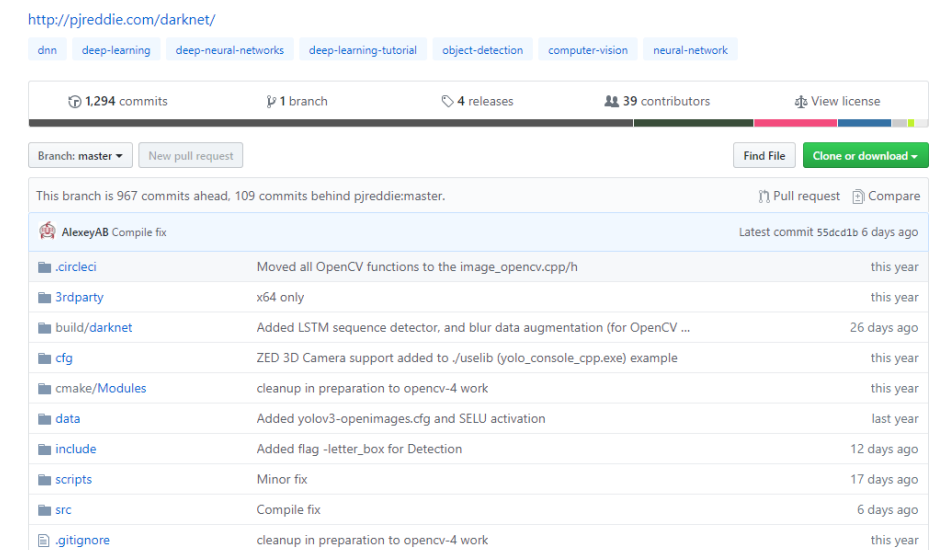
En este trabajo se utilizará la versión de YOLO AlexeyAB/Darknet en la cual se permite utilizar las características de cómputo en paralelo mediante CUDA del sistema embebido.

La ventaja de utilizar esta versión del algoritmo es que presenta las siguientes modificaciones con respecto al original proporcionado por Darknet.

- Permite la aceleración por GPU
- Permite la aceleración por CPU
- Aumenta la velocidad en 3x al utilizar núcleos Tensor en ciertas tarjetas específicas.
- Es más rápido y con menos tasas de fallos.

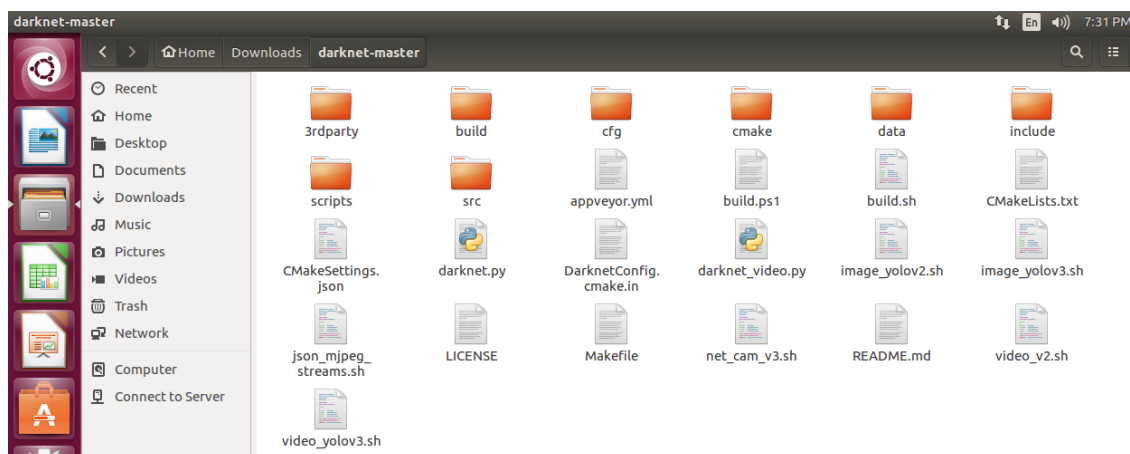


Para descargar la red neuronal ingresamos a la página web AlexeyAB/darknet:  
<https://github.com/AlexeyAB/darknet>



**Figura 27.** Github Alexey/AB  
Fuente: Github Alexey/AB

Al finalizar la descarga tendremos los siguientes archivos.

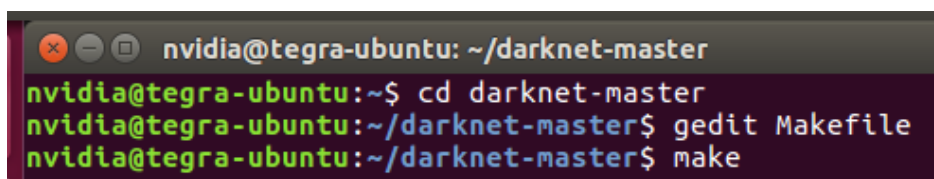


**Figura 28.** Red yolo (darknet-master).  
Fuente: Elaborado por el autor

Una vez descargada la red neuronal procedemos a descargar los pesos yolov3.weights y yolov3-tiny.weights del github de Alexey/AB.

Luego modificamos el archivo Makefile ya que en este se activan los núcleos CUDA de la tarjeta, el uso de la arquitectura correspondiente al sistema embebido Jetson TX2 y el uso de openCV y la aceleración de la CPU y GPU.

Para esto digitamos los siguientes comandos en consola.

A terminal window with a dark background and light text. The window title is 'nvidia@tegra-ubuntu: ~/darknet-master'. The terminal shows three lines of commands and their prompts: 'nvidia@tegra-ubuntu:~\$ cd darknet-master', 'nvidia@tegra-ubuntu:~/darknet-master\$ gedit Makefile', and 'nvidia@tegra-ubuntu:~/darknet-master\$ make'.

```
nvidia@tegra-ubuntu: ~/darknet-master
nvidia@tegra-ubuntu:~$ cd darknet-master
nvidia@tegra-ubuntu:~/darknet-master$ gedit Makefile
nvidia@tegra-ubuntu:~/darknet-master$ make
```

*Figura 29.* Edición archivo Makefile.

Fuente: Elaborado por el autor

Al ingresar estos comandos se despliega la siguiente pantalla en la cual debemos poner en 1 (lo que significa activar) los siguientes parámetros y seleccionar la arquitectura de red correcta para la tarjeta Jetson TX2.

```

GPU=1
CUDNN=1
CUDNN_HALF=1
OPENCV=1
AVX=0
OPENMP=1
LIBSO=1
ZED_CAMERA=0

# set GPU=1 and CUDNN=1 to speedup on GPU
# set CUDNN_HALF=1 to further speedup 3 x times (Mixed-precision on Tensor Cores) GPU: Volta, Xavier, Turing and higher
# set AVX=1 and OPENMP=1 to speedup on CPU (if error occurs then set AVX=0)

DEBUG=0

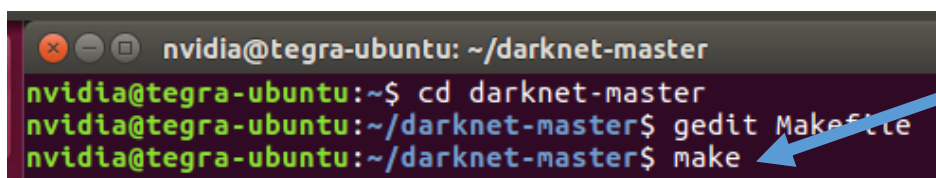
#ARCH= -gencode arch=compute_30,code=sm_30 \
# -gencode arch=compute_35,code=sm_35 \
# -gencode arch=compute_50,code=[sm_50,compute_50] \
# -gencode arch=compute_52,code=[sm_52,compute_52] \
# -gencode arch=compute_61,code=[sm_61,compute_61]
ARCH= -gencode arch=compute_62,code=[sm_62,compute_62]
OS := $(shell uname)

```

**Figura 30.** Archivo Makefile.

Fuente: Elaborado por el autor

Al realizar las modificaciones mostradas en la figura 30. Procedemos a ejecutar en consola el comando make.



```

nvidia@tegra-ubuntu: ~/darknet-master
nvidia@tegra-ubuntu:~$ cd darknet-master
nvidia@tegra-ubuntu:~/darknet-master$ gedit Makefile
nvidia@tegra-ubuntu:~/darknet-master$ make

```

**Figura 31.** Ejecución de comando make en consola

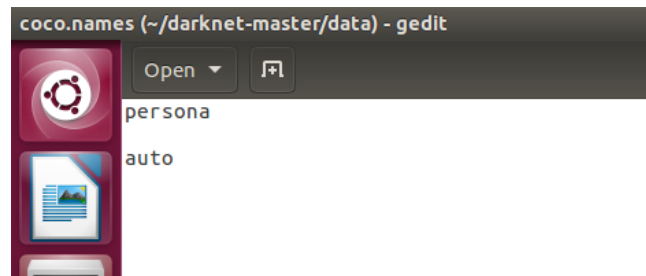
Fuente: Elaborado por el autor

Realizado todos estos pasos procedemos a la modificación del algoritmo yolov3 ya que este originalmente detecta 80 objetos diferentes y en el presente trabajo solo vamos a detectar 2 objetos; personas y autos (no buses ni camiones), para esto se realiza un cambio en el archivo “coco.data” que se encuentra en la carpeta “cfg” y en el archivo “coco.names” que se encuentra en la carpeta “data”

```
classes= 3
train = /home/pjreddie/data/coco/trainvalno5k.txt
valid = coco_testdev
#valid = data/coco_val_5k.list
names = data/coco.names
backup = /home/pjreddie/backup/
eval=coco
```

**Figura 32.** Archivo coco.data

Fuente: Elaborado por el autor

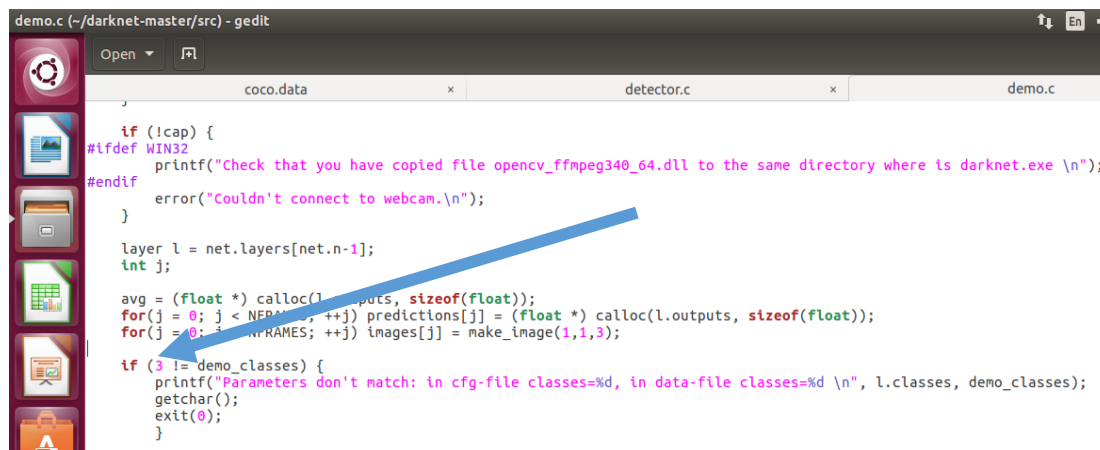


**Figura 33.** Archivo coco.names

Fuente: Elaborado por el autor

En la figura 32 se observa los dos objetos de interés por lo cual los demás objetos debemos eliminarlos hasta tener el archivo igual al de la imagen y damos clic en guardar.

Finalmente modificamos el archivo "demo.c" que se encuentra en la carpeta "src". En la figura 33 debemos ubicar el número 3 en la línea de código indicada.



```

demo.c (~/.darknet-master/src) - gedit
coco.data x detector.c x demo.c

if (!cap) {
#ifdef WIN32
    printf("Check that you have copied file opencv_ffmpeg340_64.dll to the same directory where is darknet.exe \n");
#endif
    error("Couldn't connect to webcam.\n");
}

layer l = net.layers[net.n-1];
int j;

avg = (float *) calloc(l.outputs, sizeof(float));
for(j = 0; j < NFRAMES; ++j) predictions[j] = (float *) calloc(l.outputs, sizeof(float));
for(j = 0; j < FRAMES; ++j) images[j] = make_image(1,1,3);

if (3 != demo_classes) {
    printf("Parameters don't match: in cfg-file classes=%d, in data-file classes=%d \n", l.classes, demo_classes);
    getchar();
    exit(0);
}

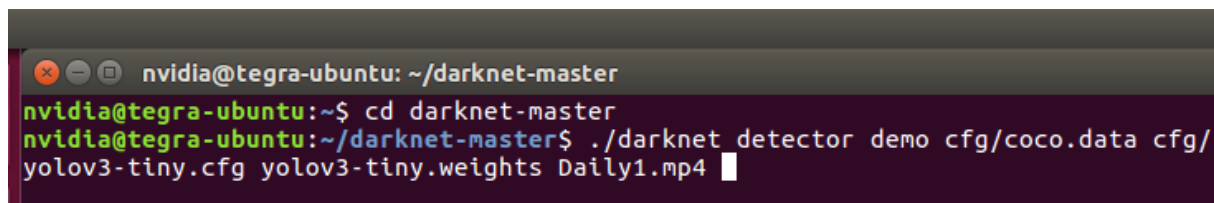
```

**Figura 34.** Archivo demo.c

Fuente: Elaborado por el autor

La red neuronal YOLO esta lista para ser utilizada, para correr el algoritmo ejecutamos el siguiente comando:

```
./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights <video file>
```



```

nvidia@tegra-ubuntu: ~/darknet-master
nvidia@tegra-ubuntu:~$ cd darknet-master
nvidia@tegra-ubuntu:~/darknet-master$ ./darknet detector demo cfg/coco.data cfg/
yolov3-tiny.cfg yolov3-tiny.weights Daily1.mp4

```

**Figura 35.** Línea de comando para ejecutar el algoritmo.

Fuente: Elaborado por el autor

## CAPITULO 5

### 5. PRUEBAS Y ANÁLISIS DE RESULTADOS

#### 5.1 Parámetros de evaluación.

Para medir la capacidad del sistema autónomo detector de objetos, se realizará pruebas con frames de imágenes en donde aplicando fórmulas estadísticas evaluaremos el número de objetos de interés (autos y personas), que es capaz de detectar el sistema. Los parámetros a tomar en cuenta para este análisis son Miss Rate (tasa de perdida) y Recall (tasa de aciertos) (Liu, Chen, Li, & Hu, 2018).

**Tabla 2**

*Relación entre valores positivos y negativos*

		Resultado de la detección	
		Personas o autos	No personas o autos
Frames de imágenes capturados en tiempo real.	Personas o autos	VP	FN
	No personas o autos	FP	VN

Fuente: (Aguaiza, 2018)

Miss Rate (MR), la tasa de pérdida mide la proporción de falsos negativos y el número total de muestras positivas, la cual se la calcula con la siguiente formula (Liu, Chen, Li, & Hu, 2018) .

$$MR = \frac{FN}{N}$$

FN corresponde a los valores falsos negativos.

N número total de muestras positivas.

Recall, es un parámetro que mide la fracción de detecciones positivas que están correctamente etiquetadas, se utiliza la siguiente expresión matemática (Liu, Chen, Li, & Hu, 2018).

$$Recall = 1 - MR$$

MR tasa de perdida.

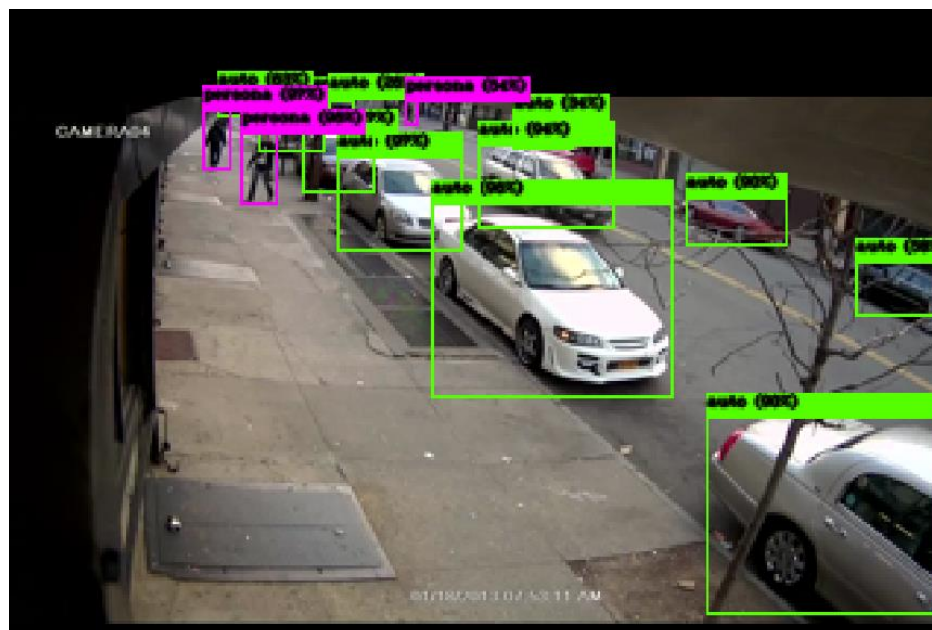
Un valor de Miss Rate correspondiente a 0.1 se considera como un valor apropiado para un sistema de detección de objetos, es una condición para que los sistemas sean considerados aptos en la práctica (Liu, Chen, Li, & Hu, 2018).

## 5.2 Pruebas con YoloV3

Con las características de la tarjeta Jetson Tx2 (256 núcleos cuda), puede procesar 4,1 a 4,4 frames por segundo. Sin embargo, hay que recalcar que el algoritmo está diseñado para trabajar a 30fps con una tarjeta Titan x.

Para realizar las pruebas se han tomado un total de 8 videos; 3 con baja resolución y 5 con alta resolución para así probar el rendimiento del detector en diversos escenarios.

1.- Prueba número 1, el siguiente video es tomado de una base de datos UCF-Anomaly-Detection-Dataset (CHARLOTTE), el cual posee una resolución de 320x240 pixeles, del mismo se ha tomado 5 capturas para verificar la tasa de perdida MR y Recall.



**Figura 36.** Detección del sistema tomado del video 120 de la base de datos.  
Fuente: Sistema autónomo de detección.

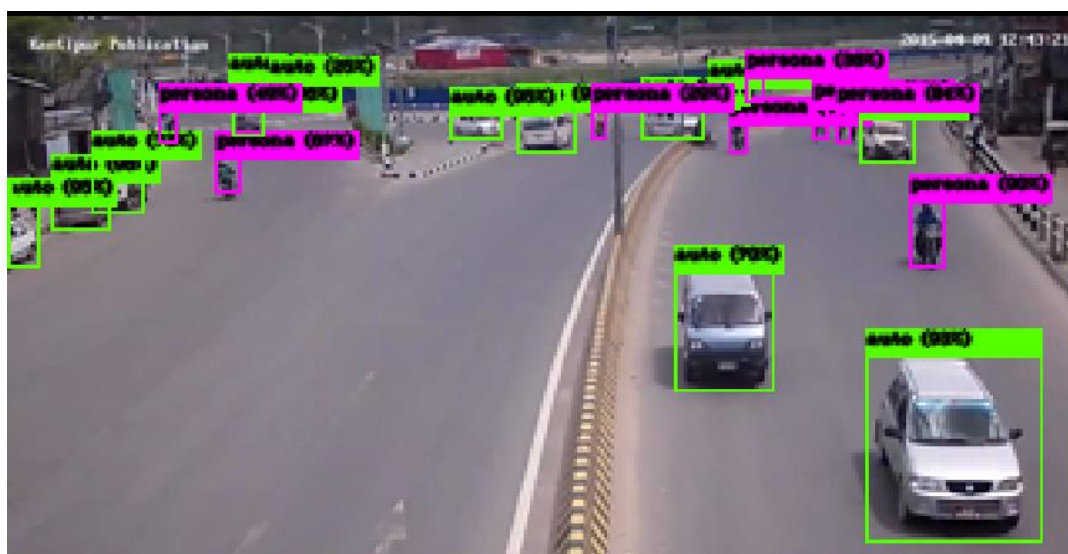


**Tabla 3**

*Frames capturados de video para la prueba 1, con el algoritmo YoloV3.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	0	13	0	1
<b>F2</b>	0	10	0	1
<b>F3</b>	0	11	0	1
<b>F4</b>	0	11	0	1
<b>F5</b>	0	13	0	1
<b>Promedio</b>	0	11,6	0	1

2.- Prueba número 2, video tomado de una base de datos UCF-Anomaly-Detection-Dataset (CHARLOTTE), el cual posee baja resolución de 320x240 pixeles del cual se tomó 5 frames capturados del video para realizar pruebas.



**Figura 37.** Detección de sistema autónomo, video 122 de la base de datos.

Fuente: Sistema autónomo de detección.

Para el análisis de los datos debido a la baja resolución es difícil distinguir algunos objetos por lo que es recomendable siempre una alta resolución en la realización de pruebas.

**Tabla 4**

*Frames capturados de video para la prueba 2, con el algoritmo YoloV3.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	1	<b>25</b>	0,04	0,96
<b>F2</b>	1	24	0,04	0,96
<b>F3</b>	2	23	0,09	0,91
<b>F4</b>	0	21	0,00	1,00
<b>F5</b>	2	22	0,09	0,91
<b>Promedio</b>	1,2	23	0,05	0,95

3.- Prueba número 3, video tomado de una base de datos UCF-Anomaly-Detection-Dataset (CHARLOTTE), el cual posee baja resolución de 320x240 pixeles del cual se tomó 5 frames capturados del video para realizar pruebas.



**Figura 38.** Detección de sistema autónomo, video 122 de la base de datos.

Fuente: Sistema autónomo de detección.

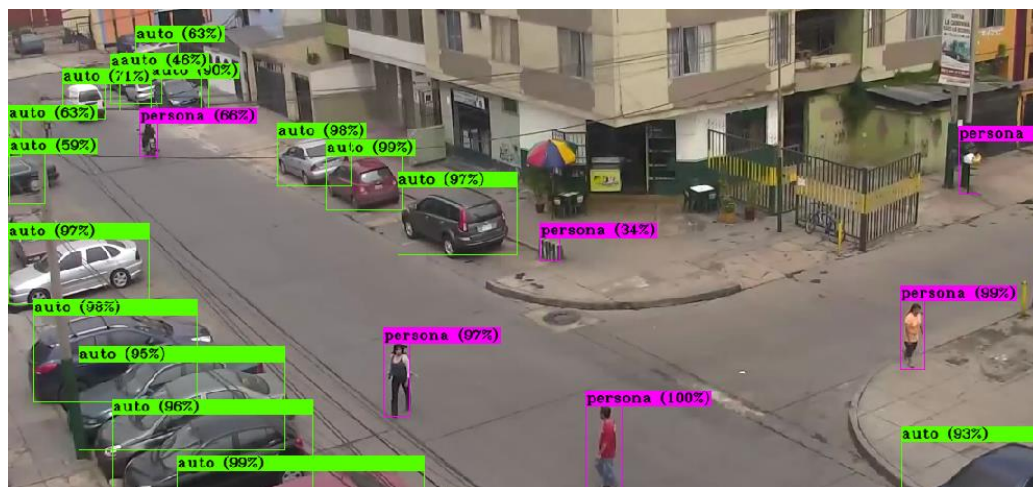
Debido a la baja resolución del video la persona que está en bicicleta el detector la enmarca 3 veces como como objeto persona.

**Tabla 5**

*Frames capturados de video para la prueba 3, con el algoritmo YoloV3.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	2	9	0,22	0,78
<b>F2</b>	0	9	0,00	1,00
<b>F3</b>	1	11	0,09	0,91
<b>F4</b>	1	8	0,13	0,88
<b>F5</b>	1	10	0,10	0,90
<b>Promedio</b>	1	9,4	0,11	0,89

4.- Prueba número 4, video que posee alta resolución de 1280x720 pixeles del cual se tomó 10 frames capturados del video para realizar pruebas.



**Figura 39.** Detección de sistema autónomo, video 2.

Fuente: Sistema autónomo de detección.

En la figura 39 se puede observar 2 falsos positivos, esto ocurre debido a la distancia a la que se encuentra la cámara y al umbral de detección que es de 0.25, es decir el sistema comienza a detectar objetos con un 25% de confianza, sin embargo, el objeto cercano detecta con un porcentaje de confiabilidad superior al 96%(se indica en el recuadro).

**Tabla 6**

*Frames capturados de video para la prueba 4, con el algoritmo YoloV3.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	<b>0</b>	<b>7</b>	0,00	1,00
<b>F2</b>	<b>1</b>	15	0,07	0,93
<b>F3</b>	<b>1</b>	8	0,13	0,88
<b>F4</b>	<b>2</b>	20	0,10	0,90

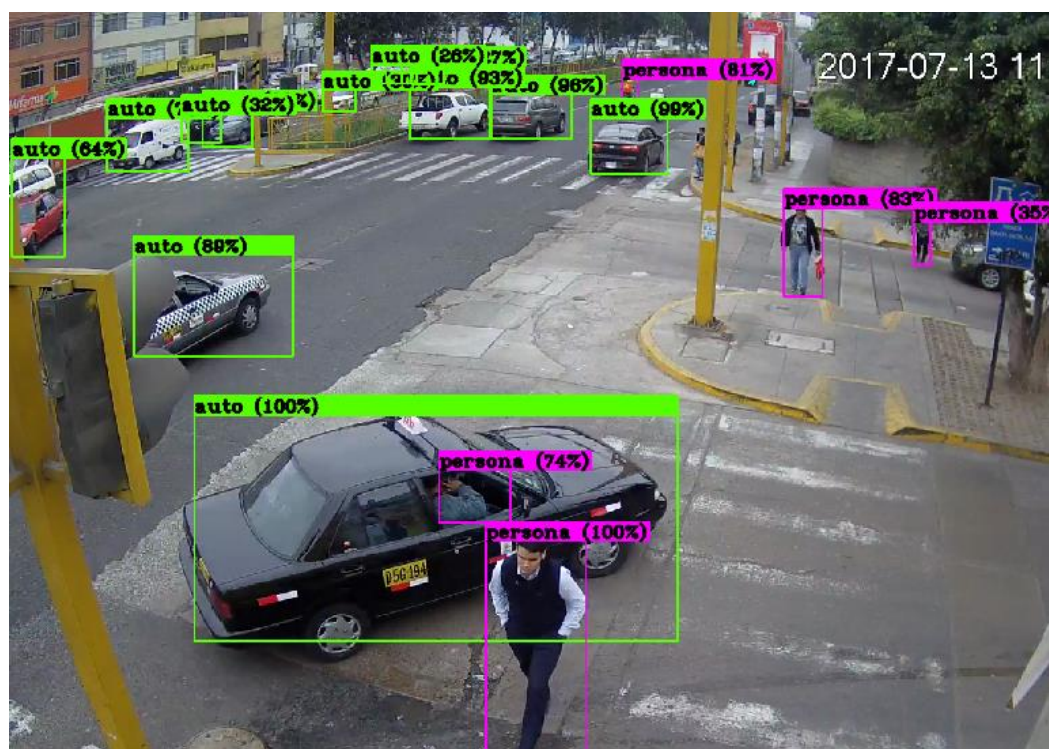
Continúa →



**Tabla 7***Frames capturados de video para la prueba 5, con el algoritmo YoloV3.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	<b>1</b>	<b>19</b>	0,05	0,95
<b>F2</b>	<b>1</b>	20	0,05	0,95
<b>F3</b>	<b>2</b>	25	0,08	0,92
<b>F4</b>	<b>2</b>	19	0,11	0,89
<b>F5</b>	<b>2</b>	21	0,10	0,90
<b>F6</b>	<b>1</b>	13	0,08	0,92
<b>F7</b>	<b>2</b>	19	0,11	0,89
<b>F8</b>	<b>2</b>	23	0,09	0,91
<b>F9</b>	<b>2</b>	23	0,09	0,91
<b>F10</b>	<b>2</b>	30	0,07	0,93
<b>Promedio</b>	1,70	21,20	0,08	0,92

6.- Prueba número 6, video tomado de una cámara de video vigilancia de youtube, el cual posee condiciones normales de iluminación y una resolución de 1280x720 pixeles del cual se tomó 10 frames capturados del video para realizar pruebas.



**Figura 41.** Detección de sistema autónomo, video 4 de cámara de video vigilancia.  
Fuente: Sistema autónomo de detección.

**Tabla 8**

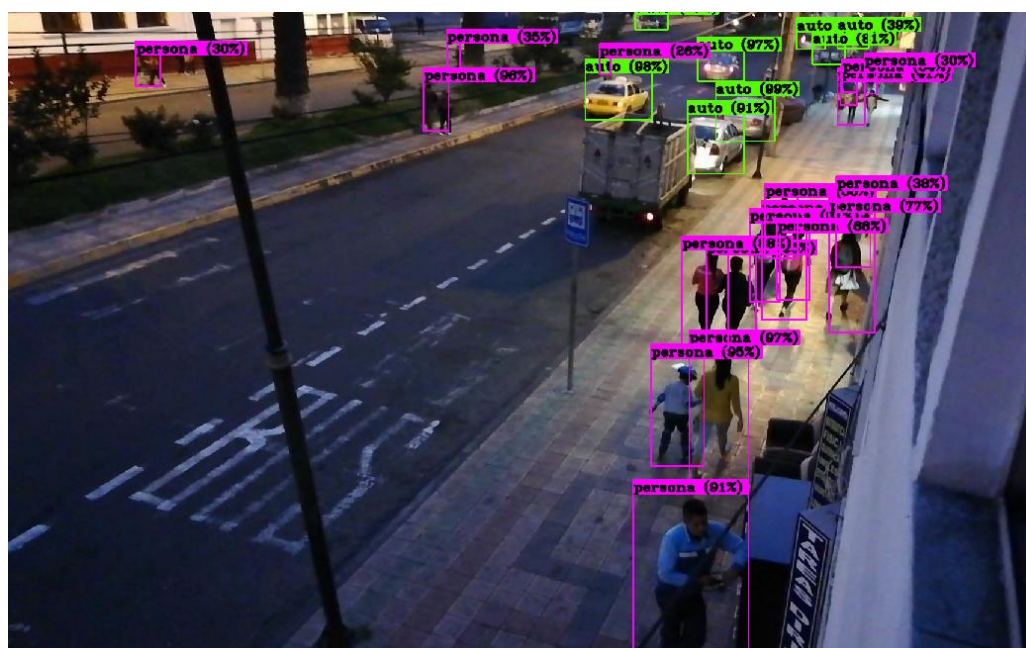
*Frames capturados de video para la prueba 6, con el algoritmo YoloV3.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	1	19	0,05	0,95
<b>F2</b>	2	20	0,10	0,90
<b>F3</b>	2	21	0,10	0,90
<b>F4</b>	1	22	0,05	0,95
<b>F5</b>	2	23	0,09	0,91
<b>F6</b>	1	20	0,05	0,95
<b>F7</b>	2	22	0,09	0,91

Continúa →

<b>F8</b>	2	19	0,11	0,89
<b>F9</b>	2	16	0,13	0,88
<b>F10</b>	1	13	0,08	0,92
<b>Promedio</b>	1,60	19,50	0,08	0,92

7.- Prueba número 7, video capturado en tiempo real alrededor de las 6:30 pm con la cámara de un teléfono celular conectado vía ip a la tarjeta Jetson Tx2 en la Ciudad de Ibarra. Se posee condiciones de iluminación media y una resolución de 1920x1080 pixeles del cual se tomó 10 frames capturados del video para realizar pruebas.



**Figura 42.** Detección de sistema autónomo, video capturado en tiempo real  
Fuente: Sistema autónomo de detección.



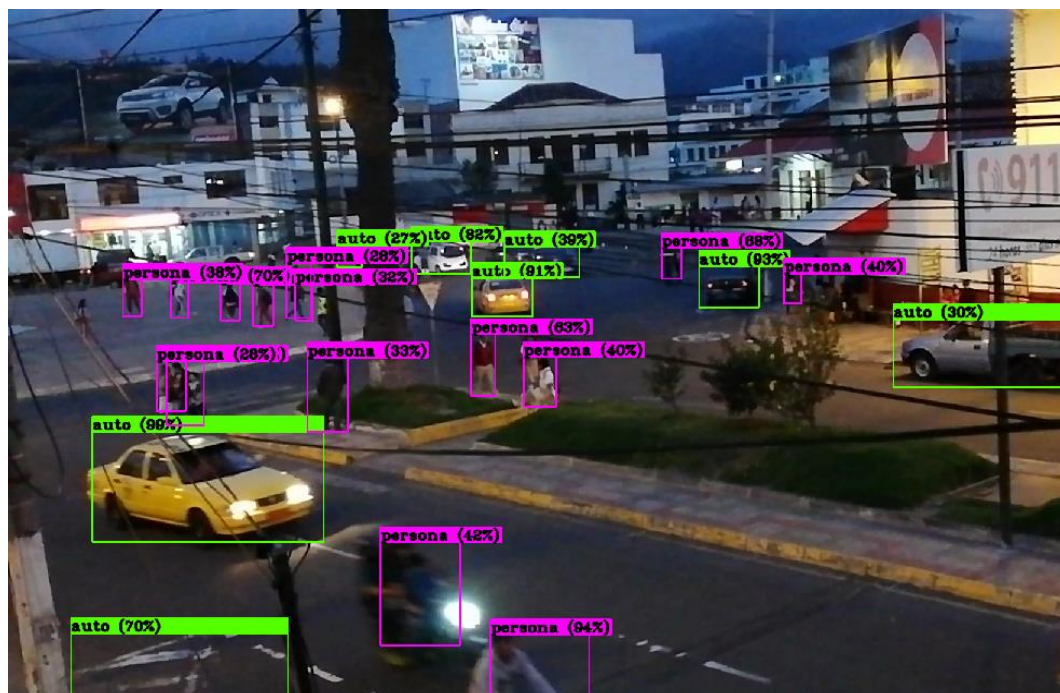
En la figura 42 la iluminación es baja, el detector es capaz de reconocer todos los objetos de interés con una tasa de aciertos de 94%, con lo cual se puede decir que es rápido y preciso el algoritmo.

**Tabla 9**

*Frames capturados de video para la prueba 7, con el algoritmo YoloV3.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	0	20	0,00	1,00
<b>F2</b>	0	22	0,00	1,00
<b>F3</b>	2	18	0,11	0,89
<b>F4</b>	1	19	0,05	0,95
<b>F5</b>	2	21	0,10	0,90
<b>F6</b>	2	15	0,13	0,87
<b>F7</b>	1	18	0,06	0,94
<b>F8</b>	2	19	0,11	0,89
<b>F9</b>	0	25	0,00	1,00
<b>F10</b>	0	18	0,00	1,00
<b>Promedio</b>	1,00	19,50	0,06	0,94

8.- Prueba número 8, video tomado en tiempo real en la ciudad de Ibarra, el cual posee condiciones de iluminación baja y una resolución de 1920x1080 pixeles del cual se tomó 10 frames capturados del video para realizar pruebas.



**Figura 43.** Detección de sistema autónomo, video capturado en tiempo real.  
Fuente: Sistema autónomo de detección.

En la figura 43, el video capturado en tiempo real presenta condiciones de luz baja tomado a las 6:30 pm en el cual se puede observar que el algoritmo presenta ciertos inconvenientes para detectar objetos muy pequeños o a su vez que estén muy alejados de la cámara, aun así para la prueba numero 8 el sistema autónomo tiene una tasa de aciertos del 88%, el cual es muy superior a otros algoritmos de detección de objetos.

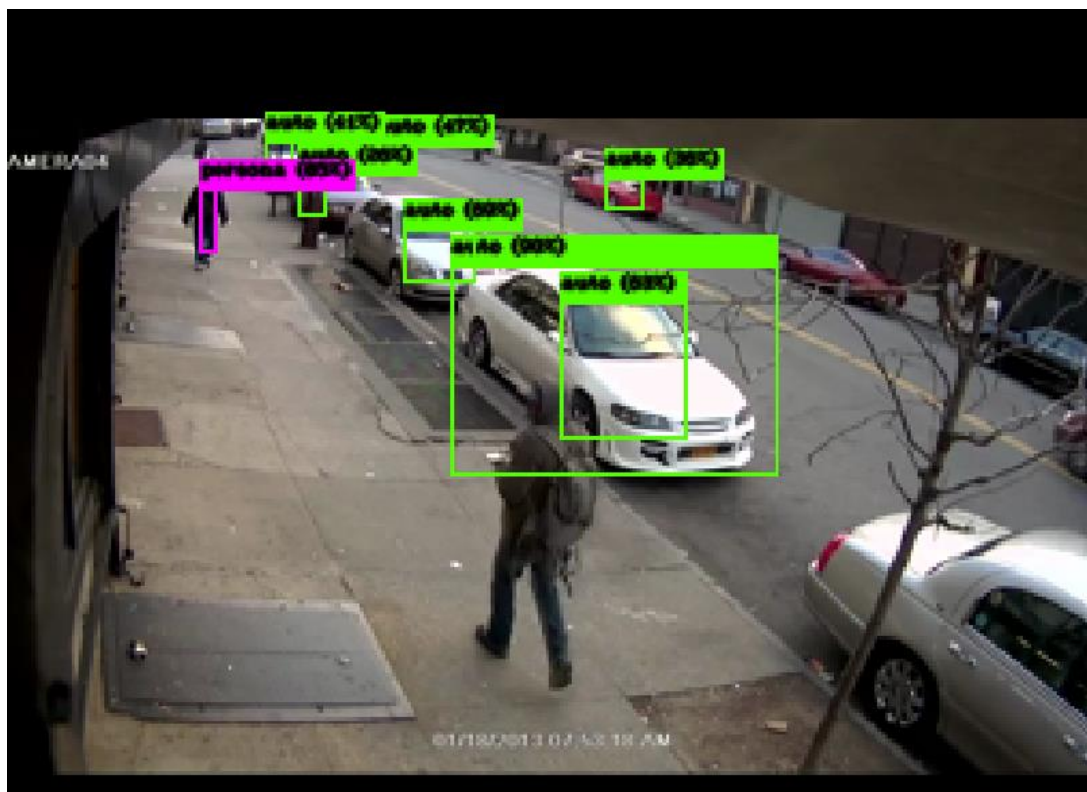
**Tabla 10***Frames capturados de video para la prueba 7, con el algoritmo YoloV3.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	2	23	0,09	0,91
<b>F2</b>	4	26	0,15	0,85
<b>F3</b>	2	23	0,09	0,91
<b>F4</b>	3	27	0,11	0,89
<b>F5</b>	4	26	0,15	0,85
<b>F6</b>	3	26	0,12	0,88
<b>F7</b>	3	25	0,12	0,88
<b>F8</b>	2	24	0,08	0,92
<b>F9</b>	2	27	0,07	0,93
<b>F10</b>	4	24	0,17	0,83
<b>Promedio</b>	2,90	25,10	0,12	0,88

### 5.3 Pruebas con YoloV3 Tiny

Con las características de la tarjeta Jetson Tx2 (256 núcleos cuda), puede procesar de 24 a 35 frames por segundo. En esta sección se repetirán las mismas pruebas con los mismos videos, pero se probará el algoritmo YoloV3-tiny.

1.- Prueba número 1, resolución de video 320x240 pixeles.



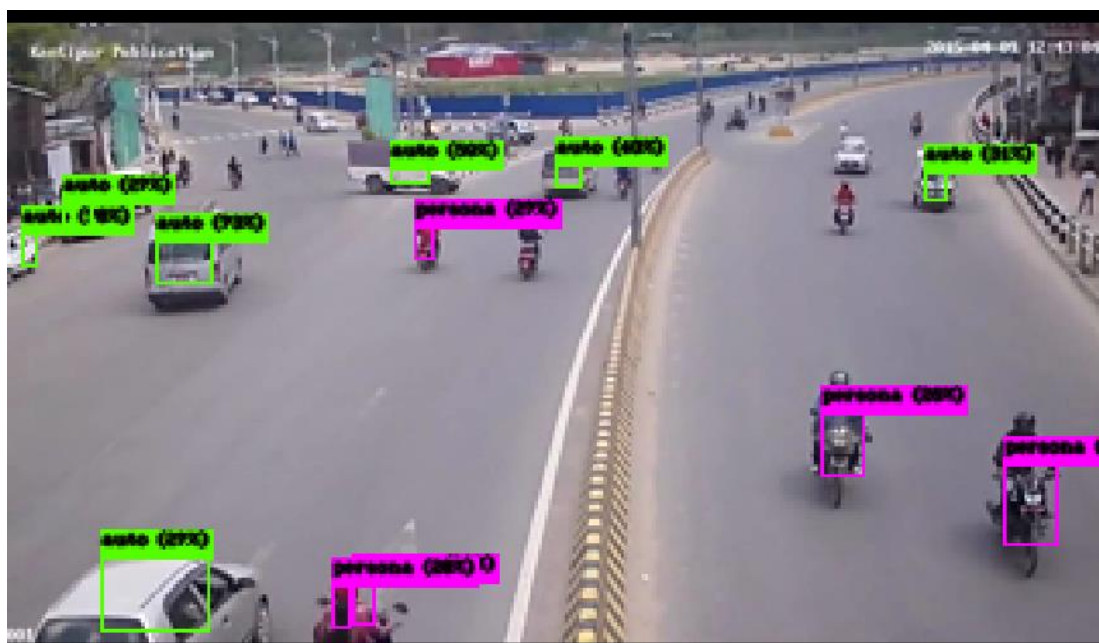
**Figura 44.** Detección de sistema autónomo, video 120 tomado de la base de datos.  
Fuente: Sistema autónomo de detección.

**Tabla 11**

*Frames capturados de video para la prueba 1, con el algoritmo YoloV3-tiny.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	5	11	0,45	0,55
<b>F2</b>	4	9	0,44	0,56
<b>F3</b>	4	10	0,40	0,60
<b>F4</b>	6	12	0,50	0,50
<b>F5</b>	4	11	0,36	0,64
<b>Promedio</b>	4,60	10,60	0,43	0,57

2.- Prueba número 2, resolución de video 320x240 pixeles.



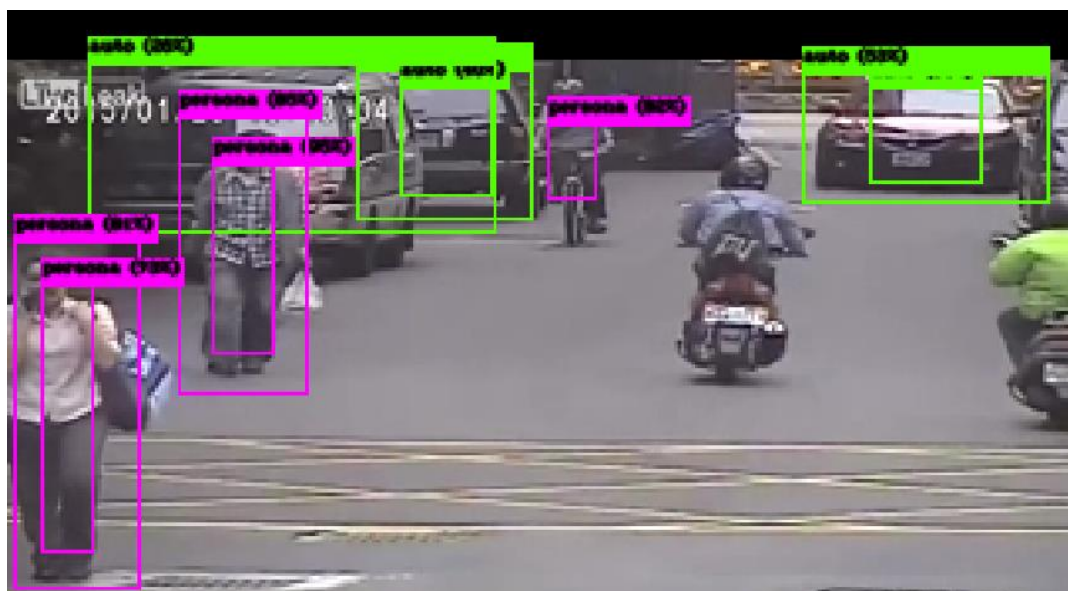
**Figura 45.** Detección de sistema autónomo, video 121 tomado de la base de datos.  
Fuente: Sistema autónomo de detección.

**Tabla 12**

*Frames capturados de video para la prueba 2, con el algoritmo YoloV3-tiny.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	12	24	0,50	0,50
<b>F2</b>	17	26	0,65	0,35
<b>F3</b>	14	23	0,61	0,39
<b>F4</b>	15	22	0,68	0,32
<b>F5</b>	10	15	0,67	0,33
<b>Promedio</b>	13,60	22,00	0,62	0,38

3.- Prueba número 3, resolución de video 320x240 pixeles.



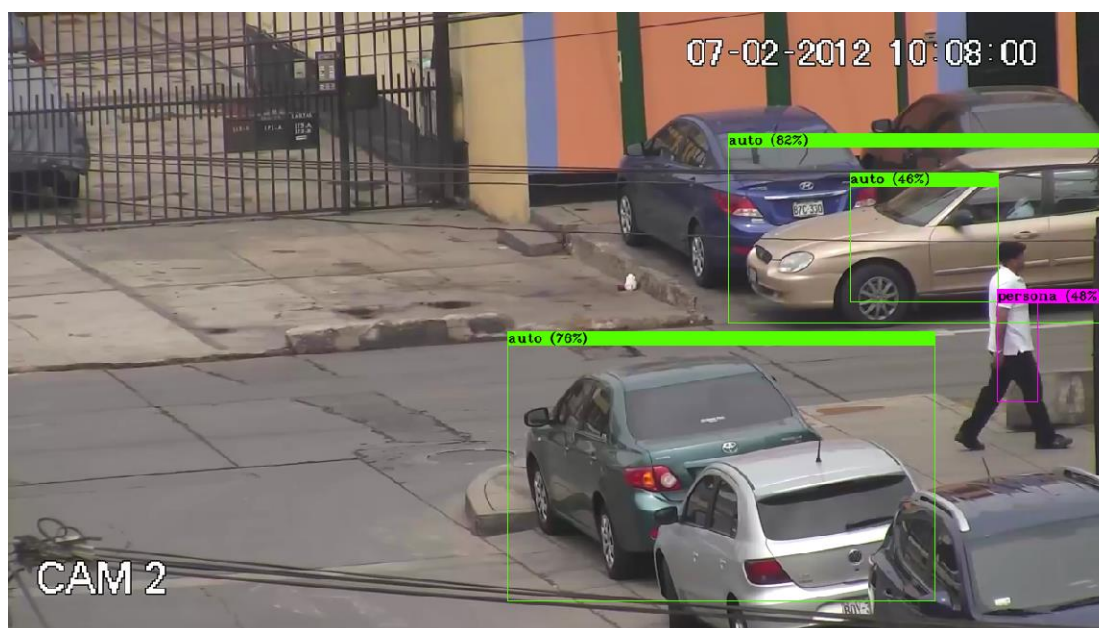
**Figura 46.** Detección de sistema autónomo, video 143 tomado de la base de datos.  
Fuente: Sistema autónomo de detección.

**Tabla 13**

*Frames capturados de video para la prueba 3, con el algoritmo YoloV3-tiny.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	4	10	0,40	0,60
<b>F2</b>	2	9	0,22	0,78
<b>F3</b>	3	9	0,33	0,67
<b>F4</b>	5	9	0,56	0,44
<b>F5</b>	5	10	0,50	0,50
<b>Promedio</b>	3,80	9,40	0,40	0,60

4.- Prueba número 4, resolución de video 1280x702 pixeles y condiciones normales de luminosidad.



**Figura 47.** Detección de sistema autónomo, video 2 tomado de la base de datos.  
Fuente: Sistema autónomo de detección.

**Tabla 14**

*Frames capturados de video para la prueba 4, con el algoritmo YoloV3-tiny.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	4	7	0,57	0,43
<b>F2</b>	4	6	0,67	0,33
<b>F3</b>	8	13	0,62	0,38
<b>F4</b>	18	23	0,78	0,22
<b>F5</b>	20	25	0,80	0,20
<b>F6</b>	20	25	0,80	0,20
<b>F7</b>	20	26	0,77	0,23

Continúa →

<b>F8</b>	15	20	0,75	0,25
<b>F9</b>	13	22	0,59	0,41
<b>F10</b>	4	6	0,67	0,33
<b>Promedio</b>	12,60	17,30	0,70	0,30

5.- Prueba número 5, resolución de video 1280x720 pixeles.



**Figura 48.** Detección de sistema autónomo, video 3 tomado de la base de datos.  
Fuente: Sistema autónomo de detección.

**Tabla 15**

*Frames capturados de video para la prueba 5, con el algoritmo YoloV3-tiny.*

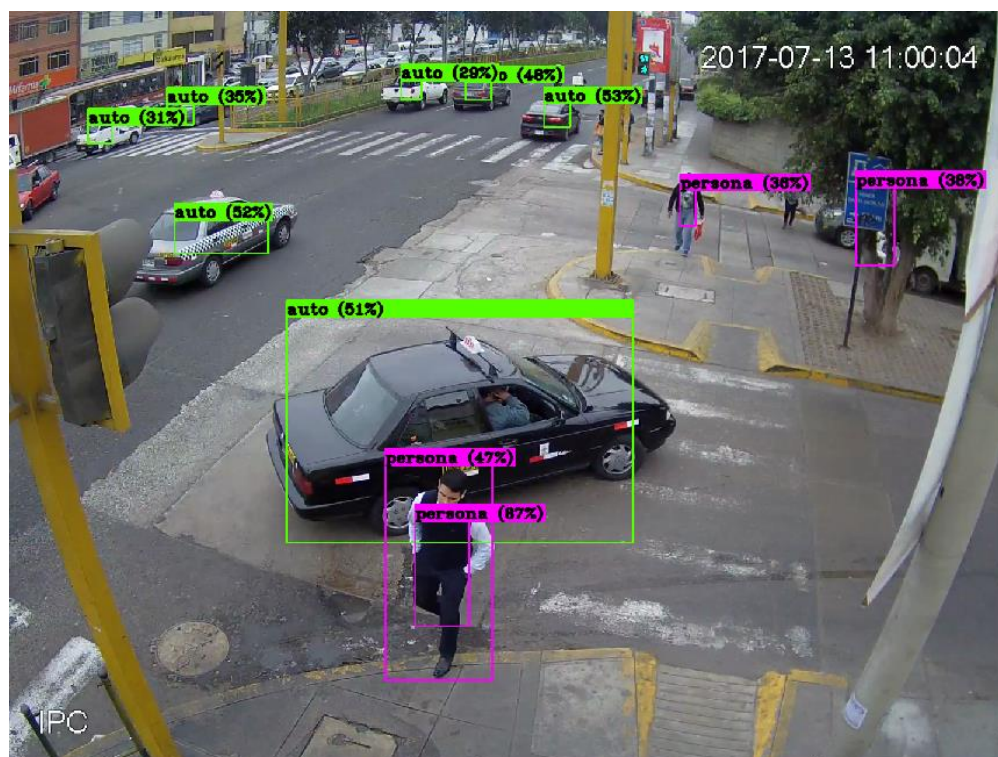
FRAMES	FN	N	MR	RECALL
<b>F1</b>	14	20	0,70	0,30
<b>F2</b>	16	22	0,73	0,27
<b>F3</b>	14	21	0,67	0,33
<b>F4</b>	15	23	0,65	0,35
<b>F5</b>	13	19	0,68	0,32

Continúa →



<b>F6</b>	10	18	0,56	0,44
<b>F7</b>	16	23	0,70	0,30
<b>F8</b>	16	22	0,73	0,27
<b>F9</b>	13	17	0,76	0,24
<b>F10</b>	25	32	0,78	0,22
<b>Promedio</b>	15,20	21,70	0,70	0,30

6.- Prueba número 6, resolución de video 1280x720 pixeles.



**Figura 49.** Detección de sistema autónomo, video 4 de cámara de video vigilancia.  
Fuente: Sistema autónomo de detección.

**Tabla 16***Frames capturados de video para la prueba 6, con el algoritmo YoloV3-tiny.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	11	20	0,55	0,45
<b>F2</b>	13	19	0,68	0,32
<b>F3</b>	14	20	0,70	0,30
<b>F4</b>	13	21	0,62	0,38
<b>F5</b>	13	18	0,72	0,28
<b>F6</b>	11	16	0,69	0,31
<b>F7</b>	14	19	0,74	0,26
<b>F8</b>	12	19	0,63	0,37
<b>F9</b>	11	16	0,69	0,31
<b>F10</b>	8	14	0,57	0,43
<b>Promedio</b>	12,00	18,20	0,66	0,34

7.- Prueba número 7, video en tiempo real con baja iluminación y resolución de 1920 x 1080 pixeles del cual se toma 10 frames.



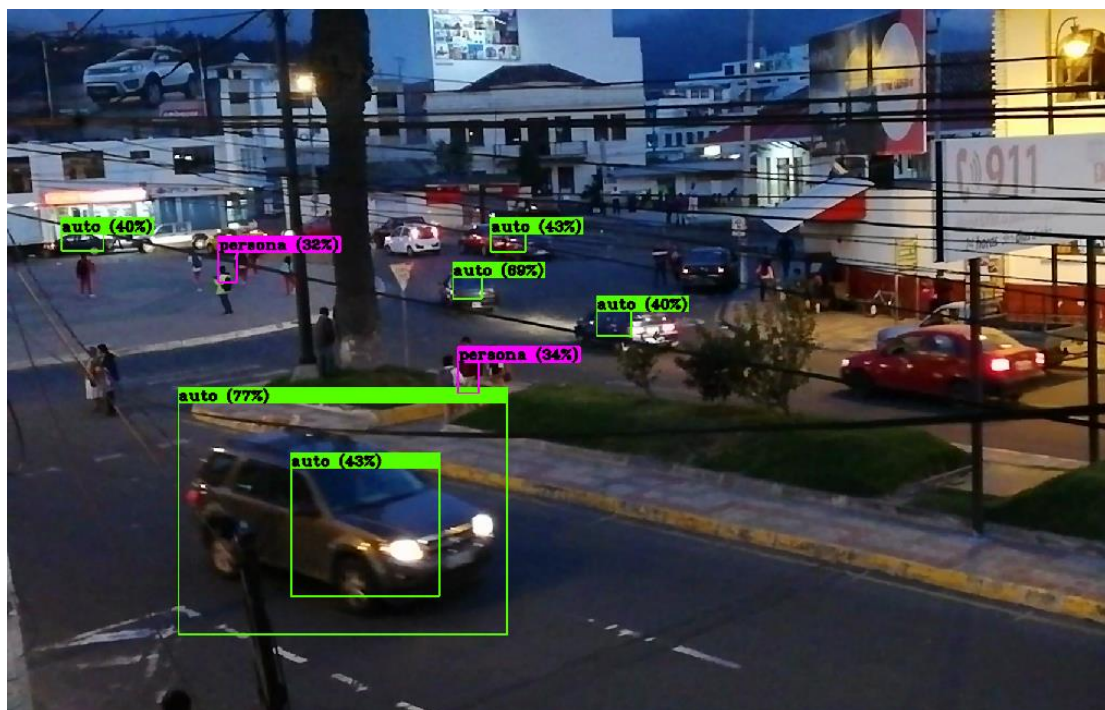
**Figura 50.** Detección de sistema autónomo, video capturado entiempro real.  
Fuente: Sistema autónomo de detección.

**Tabla 17**

*Frames capturados de video para la prueba 7, con el algoritmo YoloV3-tiny.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	9	18	0,50	0,50
<b>F2</b>	9	20	0,45	0,55
<b>F3</b>	9	15	0,60	0,40
<b>F4</b>	11	19	0,58	0,42
<b>F5</b>	10	18	0,56	0,44
<b>F6</b>	10	16	0,63	0,38
<b>F7</b>	13	19	0,68	0,32
<b>F8</b>	12	20	0,60	0,40
<b>F9</b>	8	21	0,38	0,62
<b>F10</b>	12	21	0,57	0,43
<b>Promedio</b>	10,30	18,70	0,55	0,45

8.- Prueba número 8, video en tiempo real con baja iluminación tomado a las 7pm y resolución de 1920 x 1080 pixeles del cual se toma 10 frames.



**Figura 51.** Detección de sistema autónomo, video capturado entiempro real.

Fuente: Sistema autónomo de detección.

**Tabla 18**

*Frames capturados de video para la prueba 8, con el algoritmo YoloV3-tiny.*

FRAMES	FN	N	MR	RECALL
<b>F1</b>	25	28	0,89	0,11
<b>F2</b>	23	25	0,92	0,08
<b>F3</b>	17	24	0,71	0,29
<b>F4</b>	18	24	0,75	0,25

Continúa →

<b>F5</b>	10	16	0,63	0,38
<b>F6</b>	13	21	0,62	0,38
<b>F7</b>	17	22	0,77	0,23
<b>F8</b>	26	28	0,93	0,07
<b>F9</b>	22	24	0,92	0,08
<b>F10</b>	17	26	0,65	0,35
<b>Promedio</b>	18,80	23,80	0,78	0,22

En la siguiente tabla se muestran los resultados de los parámetros de evaluación tanto para el modelo YoloV3 como para el modelo YoloV3-tiny.

**Tabla 19**

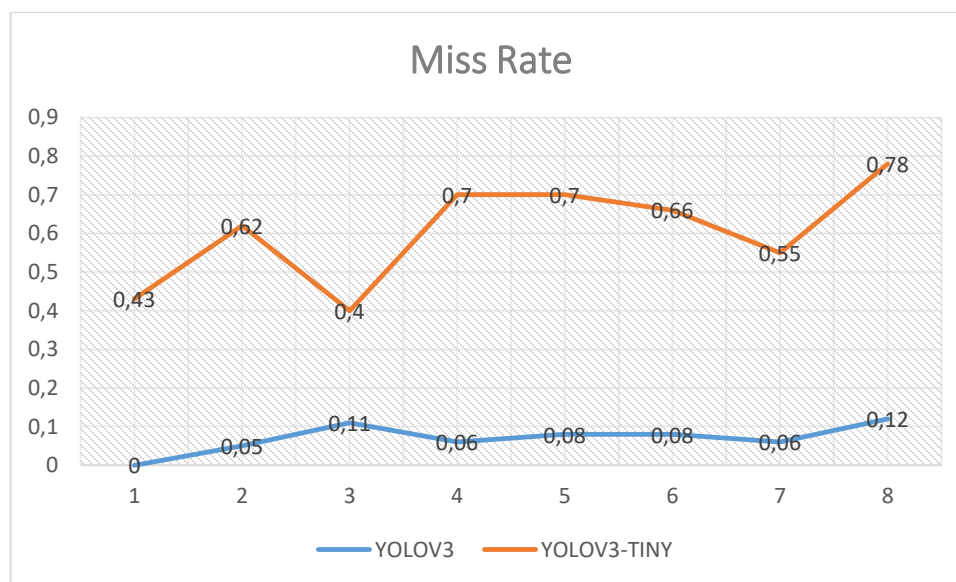
*Comparación de parámetros de evaluación YoloV3 y YoloV3-tiny*

	<b>PRUEBA NÚMERO</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>PROMEDIO</b>
<b>MISS RATE</b>	<b>YOLOV3</b>	0	0,05	0,11	0,06	0,08	0,08	0,06	0,12	0,07
	<b>YOLOV3-TINY</b>	0,43	0,62	0,4	0,7	0,7	0,66	0,55	0,78	0,61
<b>RECALL</b>	<b>YOLOV3</b>	1	0,95	0,89	0,94	0,92	0,92	0,94	0,88	0,93
	<b>YOLOV3-TINY</b>	0,57	0,38	0,6	0,3	0,3	0,34	0,45	0,22	0,40

Como se observa el modelo YoloV3 que presenta 106 capas de convolución alcanza en promedio un 93% en la tasa de aciertos, frente a un modelo YoloV3-tiny que alcanza en promedio un 40% debido a que solo presenta 23 capas de convolución.

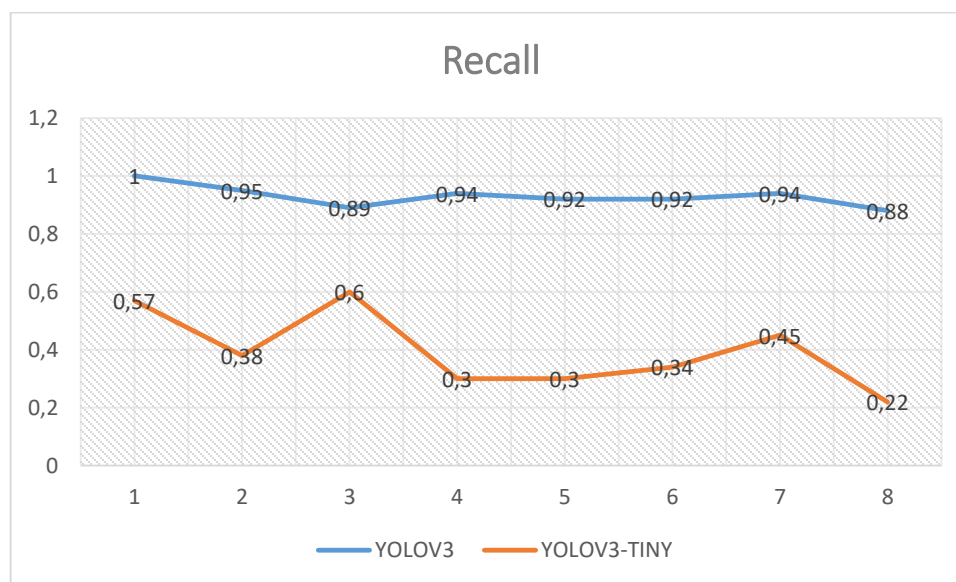
Al obtener 93% en la tasa de aciertos para el algoritmo YoloV3 supera la barrera del 90% con lo cual es considerado un buen detector de imágenes. De igual forma solo se recomienda

utilizar el algoritmo yoloV3-tiny solo en sistemas embebidos con bajos recursos computacionales.



**Figura 52.** Comparativa Miss Rate.

Fuente: Elaborado por el autor



**Figura 53.** Comparativa Recall.

Fuente: Elaborado por el autor

## CAPÍTULO 6

### 6. CONCLUSIONES Y RECOMENDACIONES

#### 6.2 Conclusiones

- Se implementó un detector de autos y personas en video capturado en tiempo real utilizando una modificación de la red neuronal yolov3 AlexeyAB/Darknet.
- El framework Linux montado en el kit de desarrollo de la placa Nvidia Jetson Tx2, funciona con una cámara de video IP inalámbrica para capturar video en ambientes con distinto nivel de iluminación.
- Se probaron las dos versiones del algoritmo detector YOLO la versión completa y la versión liviana (yolov3, yolov3-tiny), con distintos tipos de video tomados de bases de datos y capturados en tiempo real.
- El prototipo implementado detecta los objetos con un grado de acierto superior al 95% en objetos cercanos y para objetos muy alejados con un grado de acierto superior al 65%.
- Al evaluar el desempeño del detector, con la versión completa de yolov3 (106 capas de convolución) el prototipo procesa el video a 4.4 fps mientras que con el algoritmo yolov3-tiny (26 capas de convolución) corre hasta 32 fps.

### 6.3 Recomendaciones

- Para mejorar el desempeño en cuanto a rapidez de procesamiento del detector implementado se recomienda utilizar una versión superior de la placa Nvidia como por ejemplo la tarjeta Jetson AGX Xavier que según el fabricante supera en 20 veces el desempeño de la tarjeta utilizada en este proyecto.
- Para trabajos futuros se puede usar este prototipo en la implementación de aplicaciones tales como sistemas autónomos de video vigilancia, alerta en lugares públicos, control de tráfico, ayudas para personas no videntes, etc.



## REFERENCIAS BIBLIOGRÁFICAS

- Aguaiza, C. (2018). *Sistema de estimación de número de personas en tiempo real durante misiones de reconocimiento del ejército ecuatoriano utilizando vehículos aéreos no tripulados multirotor*. Sangolqui: ESPE.
- Ambriz, J., Eleuterio, R., & González, E. (2017). Implementación de visión por computadora a un móvil autónomo basado en raspberry pi. *Universidad & Ciencia*.
- Charlotte, U. (s.f.). *UCF-Anomaly-Detection-Dataset*. Nort of Carolina, EEUU.
- Cortés, C. (2017). *Herramientas Modernas en Redes Neuronales*. Madrid-España: Universidad Autónoma de Madrid.
- Girshick, R. (2015). Fast R-CNN. *Microsoft Research*, 1,2.
- Girshick, R., Donahue, J., & Darrell, T. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *arxiv.org*, 1,2.
- Kathuria, A. (23 de Abril de 2018). What's new in YOLO v3? *Towards Data Science*.
- Liu, Z., Chen, Z., Li, Z., & Hu, W. (2018). An Efficient Pedestrian Detection Method Based on YOLOv2. *Hindawi*, 10.
- Lopez, R. (2017). *Introducción al Deep Learning*. Buenos Aires: IARR.
- Matich, D. (2001). *Redes Neuronales: Conceptos Básicos y Aplicaciones*. Rosario: Universidad Técnica Nacional.

- Montaño Moreno, J. (2002). *Redes Neuronales Artificiales aplicadas al Análisis de Datos*. Palma de Mallorca: Universitat De Les Illes Balears.
- Nicholas, C. (16 de Octubre de 2017). Installing Jetpack 3.1 on Jetson TX2 with Tensorflow 1.3 / Keras / hdf5 (.h5). *Data Science Dev*.
- Nvidia. (2019). *Autonomous Machines*. EEUU.
- Peonza, W., & Dominguez, M. (2017). Combined Approach using Artificial Vision and Neural Networks for Facial Recognition. *sci-hub*.
- Redmon, J., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *IEEE Xplore*.
- Redmon, J., Divvala, S., & Girshick, R. (2016). You Only Look Once:. *University of Washington*.
- Ren, S., He, K., & Girshick, R. (2016). Faster R-CNN: Towards Real-Time Object. *arxiv*, 1,2,3,4.
- Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *ScienceDirect*.
- Terven, J., Salas, J., & Raducanu, B. (2013). Estado del Arte en Sistemas de Visión Artificial para Personas Invidentes. *Komputer Sapiens*.
- Xu, W., He, J., & Lan , H. (2016). Real-Time Target Detection and Recognition with Deep Convolutional Networks for Intelligent Visual Surveillance. *IEEE Xplore*.