UNIVERSIDAD RICARDO PALMA

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

Implementación De Algoritmo De Visión Artificial Y Red Neuronal Convolucional Embebida, En Un Automóvil A Escala Para La Conducción Autónoma En Circuito De Pruebas Controlado

TESIS

Para optar el título profesional de Ingeniero Electrónico

AUTOR

Díaz Villanueva, Julio Leonardo
ORCID: 0000-0003-1984-0547

ASESOR

Huamaní Navarrete, Pedro Freddy
ORCID: 0000-0002-3753-9777

Lima, Perú

2022

Metadatos Complementarios

Datos del autor

Díaz Villanueva, Julio Leonardo
DNI: 72153233

Datos de asesor

Huamaní Navarrete, Pedro Freddy
DNI: 10032682

Datos del jurado

JURADO 1
González Prado, Julio Cesar
DNI: 07702235
ORCID: 0000-0003-0384-7015

JURADO 2
Terukina Oshiro, Nelly Luz
DNI: 07808963
ORCID: 0000-0002-9654-7961

JURADO 3
Sánchez Bravo, Miguel Ángel
DNI: 08443357
ORCID: 0000-0001-9384-1391

JURADO 4
Chong Rodriguez, Humberto
DNI: 07811343
ORCID: 0000-0002-4643-0538

Datos de la investigación

Campo del conocimiento OCDE: 2.02.01
Código del Programa: 712026

DEDICATORIA

Entrego este manuscrito a Dios, a mis padres Carlos, Maribel a mi hermano Juan por siempre acompañarme guiarme y quererme, a mis abuelos Juan y Julio, a mis abuelas Blanca, Susana por los buenos momentos que pasamos y por la sabiduría que me daban. Gracias a ellos y al apoyo de toda mi familia y amigos pude tener la inspiración para lograr la redacción de esta tesis.

AGRADECIMIENTO

Agradecido siempre a mis profesores en mi periodo de estudiante el apoyo incondicional del profesor Huamaní asesor de esta tesis al cual estoy profundamente agradecido, como también las enseñanzas y el ejemplo del profesor Roselló cuando fui su alumno. Además, quiero agradecer a mis compañeros de carrera Gianfranco, Walter, Bill y Pedro, con los cuales pudimos sobrepasar los cursos más difíciles con horas de estudio en el G 310, como también a mis compañeros de colegio Juan, Junior y Marco los cuales siempre me dieron ánimos en mi carrera.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En este proyecto de tesis se implementó un sistema de conducción autónoma usando una red neuronal convolucional, y capaz de lograr un nivel de autonomía 4 respondiendo al entorno donde se encuentra; para eso, fue necesario crear módulos que uniéndose entre ellos se lograron alcanzar los objetivos propuestos. Tales módulos se encargaron del control de motores de arranque y dirección del automóvil, la captura de la imagen exterior la cual fue realizada por 2 lentes (de baja distorsión y del tipo angular) independientemente cada una, la captura de los datos de un mando vía USB para poder controlar el automóvil y obtener un comando de giro producido por este dispositivo para entrenar la red neuronal; luego, teniendo estos datos capturados, también se implementó un módulo de recolección de datos que permitió combinar la imagen y el comando de giro el cual fue balanceado para contar con una data de entrenamiento apropiada para poder entrenar la red neuronal convolucional; asimismo, esta red neuronal fue optimizadacon Keras Tuner y Tensor RT lo que permitió una mejora en las métricas VAL_LOSS con un valor de 0.01211 y en la de LOSS con un valor de 0.20611; además, se alcanzó una mejora de 2.247 en el tiempo de inferencia del automóvil, así como en el ahorro del 20% de la memoria RAM del sistema embebido utilizado.

Finalmente, para interactuar con el ambiente se crearon 2 módulos más. El primero encargado de recibir información proveniente de un microcontrolador Atmega 328p, conectado con un sensor ultrasonido que proporciona información de distancia entre el automóvil y un objeto que tenga adelante, y el segundo encargado del reconocimiento de las señales de tráfico, peatones y semáforos, pero utilizando la técnica Haar Cascade para reconocer patrones de las distintas clases. Y, con la unión de todos los módulos, se obtuvo un sistema de conducción autónoma que puede transitar sin intervención humana en un circuito de pruebas controlado, además de reaccionar al entorno donde se encuentra.

Palabras claves: Conducción Autónoma, Jetson-Nano, Keras-Tuner, Tensor-RT, Circuito de Pruebas.

ABSTRACT

In this thesis project a system of autonomous driving was implemented using a convolutional neural network, and capable of achieving a level of autonomy 4 responding to the environment where it is located; for that, it was necessary to create modules that joined together they were able to achieve the proposed objectives. Such modules were in charge of the control of the car's starter and steering motors, the capture of the exterior image which was performed by 2 lenses (low distortion and angular type) independently each one, the capture of data from a remote control via USB to be able to control the car and obtain a turn command produced by this device to train the neural network; then, having this data collected, a data collection module was also implemented to combine the image and the turning command which was balanced to have an appropriate training data to train the convolutional neural network; likewise, this neural network was optimized with Keras Tuner and Tensor RT which allowed an improvement in the VAL_LOSS metrics with a value of 0. 01211 and LOSS with a value of 0.20611; in addition, an enhancement of 2.247 was achieved in the automobile inference time, as well as in the saving of 20% of the RAM memory of the embedded system used.

Finally, two more modules were developed to interact with the environment. The first module is in charge of receiving data from an Atmega 328p microcontroller, connected to an ultrasound sensor that provides information on the distance between the car and an object in front of it, and the second module is in charge of recognizing traffic signals, pedestrians and traffic lights, but using the Haar Cascade technique to recognize patterns of different kinds. And, with the union of all the modules, we obtained an autonomous driving system that can drive without human intervention in a controlled test circuit, in addition to reacting to the environment where it is.

Keywords: Autonomous Driving, Jetson-Nano, Keras-Tuner, Tensor-RT, Test Track.

INTRODUCCIÓN

En la época que vivimos, la Inteligencia Artificial es usada en muchas aplicaciones: carros automáticos, procesos industriales, detección de enfermedades, visión artificial, etc, y en particular, si nos centramos en el campo de los carros autónomos, promete revolucionar la forma de desplazarnos en nuestro día a día teniendo como factor principal la seguridad de los pasajeros; no obstante, este último no ha venido ocurriendo por imprudencia temeraria del conductor, es así que en el año 2016 se tuvo el 62.8% de estos casos (Castro, 2016). De esta manera, todas estas causas de accidentes son fallas humanas; sin embargo, una inteligencia artificial con el debido entrenamiento reduciría notoriamente esta fatídica estadística, aunque es una tarea compleja que requiere de gran capacidad computacional y una base de datos extensa que se puede implementar en pequeña escala; y, luego, con los elementos necesarios se alcanzaría una implementación en un automóvil real. Además, en la actualidad, con la ayuda de sistemas embebidos de pequeñas dimensiones, es posible realizar prototipos a escala de automóviles con la asistencia de una inteligencia artificial para ayudar a realizar una conducción autónoma.

Dado esto, en este proyecto de tesis, se propuso desarrollar un automóvil a escala el cual puede conducirse de manera autónoma en un circuito de pruebas controlado, bajo la existencia de algunos elementos típicos de una autopista como señales de tráfico, semáforos y peatones. Para lo cual, se implementó una red neuronal convolucional más un algoritmo de detección de elementos de tránsito dentro de un sistema embebido JETSON-NANO, el cual representó al cerebro del automóvil desarrollado a escala.

De esta manera, en el primer capítulo de esta tesis se aborda el planteamiento y delimitación del problema de lo que implica la conducción autónoma, el objetivo a lograr con este proyecto y la justificación de esta. Asimismo, en el segundo capítulo se abarca todas las investigaciones tomadas en cuenta para poder desarrollar este proyecto. Luego, la implementación del sistema de conducción, módulos, optimizaciones es descrita en el tercer capítulo; mientras que la pruebas y resultados del proyecto se desarrollan en el último capítulo.

CAPÍTULO I: PLANTEAMIENTO Y DELIMITACIÓN DEL PROBLEMA

1.1. Formulación y delimitación del problema

La conducción autónoma es considerada como una de las tecnologías emergentes del siglo 21, su desarrollo tiende a estar centrado en países desarrollados y con una industria automovilística adelantada, como un ejemplo se tiene a la empresa Tesla fundada por Elon Musk donde desarrollan automóviles autónomos dotados de cámaras, sensores y un sistema que permite hacer realidad la conducción autónoma.

Entonces para poder determinar el problema general, partiendo de la información anteriormente señalada, primero se procedió a identificar qué herramientas se usarían para poder dar una solución basada en una red neuronal convolucional capaz de ser entrenada y prototipada en un sistema embebido, de tal forma que permita dar la validación en un circuito real de pruebas a escala con resultados y nivel de autonomía deseado. Este último, resultó ser de un nivel 4 donde el automóvil analizó el entorno y definió la ruta correcta, así como también la respuesta a cualquier cambio en el entorno. De esta forma, surgen las siguientes preguntas de investigación:

1.1.1. Problema General

¿Cómo implementar un algoritmo de visión artificial y una red neuronal convolucional embebida, en un automóvil a escala, para la conducción autónoma en un circuito de pruebas controlado?

1.1.2. Problemas Específicos

- ¿Cómo se implementa un automóvil a escala con motores de arranque traseros y dirección en la parte delantera usando un sistema embebido Jetson Nano?
- ¿Cómo se obtiene el DATASET de entrenamiento necesario para realizar una conducción autónoma y cómo podemos balancear esta información para que sea óptima para el entrenamiento usando algoritmos de big data?
- ¿Cómo se puede desarrollar un algoritmo de visión artificial, basado en el algoritmo Haar Cascade, para reconocer señales de tráfico y semáforos?

- ¿De qué manera se entrena la red neuronal convolucional con TENSORFLOW GPU como también usar KERAS-TUNER para sintonizar los hiper-parametros de la red neuronal utilizando el DATASET optimizado, y como ejecutar el framework TensorRT para mejorar su performance?
- ¿Cómo realizar la validación de la red neuronal convolucional en el circuito de pruebas a escala, midiendo el performance de la misma?

1.2. Objetivos

1.2.1. Objetivo General

Implementar un algoritmo de visión artificial y una red neuronal convolucional embebida, en un automóvil a escala para la conducción autónoma en un circuito de pruebas controlado.

1.2.2. Objetivos Específicos

- Implementar un automóvil a escala con motores de arranque traseros y dirección en la parte delantera usando un sistema embebido Jetson Nano.
- Obtener el DATASET de entrenamiento de la red neuronal convolucional para realizar una conducción autónoma, y a su vez balancear esta información para hacerla más adecuada el proceso de entrenamiento.
- Desarrollar un algoritmo de visión artificial, basado en el algoritmo Haar Cascade, para reconocer señales de tráfico, peatones y semáforos.
- Entrenar la red neuronal convolucional con TENSORFLOW GPU y usar KERAS-TUNER para la sintonización de hiper-parametros utilizando el DATASET optimizado, y ejecutar el framework TensorRT para mejorar su performance.
- Realizar la validación de la red neuronal convolucional en el circuito de pruebas a escala, midiendo el performance de la misma.

1.3. Importancia y justificación del estudio

Según el “National Highway Traffic Safety Administration” (2018) indica que casi el 90% de accidentes se producen por falla humana, lo cual impulsa a los fabricantes de autos a crear nuevos sistemas para evitar tragedias, como por ejemplo sensores en el contorno del automóvil, sistemas de aviso si el automóvil se desvía del carril, detección

de peatones y otras funciones en el automóvil. Otro dato importante es el impacto económico del uso de esta tecnología; pues, en el año 2010 se perdieron aproximadamente $ 242 billones de dólares por accidentes y $594 billones en daños humanos, lo cual es un signo que empleando esta forma de conducción existirá un ahorro monetario. Asimismo, Almenara (2019) dio a conocer una estadística de las ciudades con mayor tráfico del mundo, en donde la capital de nuestro país se encuentra en el puesto 3 y siendo superadas por Bogotá y Mumbai; y, según Mena (2018), un ciudadano promedio pasa 4 horas en el tráfico limeño significando casi 12 años de su vida en este desorden que altera su estado de ánimo, y en algunos casos llevando a casos de estrés. Por lo cual, utilizar automóviles autónomos podría ahorrar hasta 50 minutos de tráfico según lo investigado por Rojas (2015), lo que otorgaría más tiempo a las personas para la realización de otras actividades.

Dados los motivos expuestos, se decidió experimentar con una red neuronal convolucional implementada en un sistema embebido y montado en un vehículo a escala, con la cual se pudo realizar pruebas de manejo autónomo en un circuito real controlado. Asimismo, fue posible observar de manera física el comportamiento de esta, además de poder observar su performance y de acuerdo a esto obtener información valiosa para implementar este modelo en un vehículo a escala real, que sería de gran ayuda para el desarrollo de nuevas tecnologías en el país.

1.4. Limitaciones del Estudio

El presente trabajo de tesis se limitó a la implementación de una red neuronal convolucional entrenada específicamente, en un circuito de pruebas de dimensiones de 3x3 metros en el cual se tuvo algunos tipos de elementos que podremos encontrar en una autopista; como, por ejemplo, señales verticales dispuestas por el Ministerio de Transportes y Comunicaciones (MTC) en el Manual de Dispositivos de Control de Tránsito Automotor para Calles y Carreteras (2016). De esta manera, a continuación, se listan:

- Señales de Prioridad: En este campo se empleó solamente la señal Pare. (R-1), la cual indica a los conductores que deberían efectuar la detención de su vehículo.
- Señales Prohibitivas o Restrictivas: Se emplearon señales para indicar la velocidad máxima permitida (R-30G) a la cual podrá circular el automóvil, las velocidades fueron de 30 % y 50% de velocidad máxima como valores expresados en el duty cycle de los motores de arranque.

Además, este proyecto se limita a contar con líneas de borde para el supuesto pavimento y de un color distinto, de tal forma que indiquen el límite del carril con la vereda u otro tipo de referencias que permitan que el vehículo pueda desarrollar la conducción autónoma.

Asimismo, este trabajo de tesis se limita a un número de 2 semáforos para poder simular un entorno más real y observar la respuesta del sistema propuesto; igualmente, se buscó que obtenga un nivel de autonomía 4 en el cual no habría interacción humana. Todas estas acciones fueron embebidas en una Jetson Nano, en donde se ejecutó el algoritmo para realizar la conducción autónoma. Por último, todas las pruebas fueron realizadas en un entorno real controlado en el cual fue posible manipular variables del entorno, para poder observar como el automóvil a escala reacciona a cambios repentinos.

CAPÍTULO II: MARCO TEÓRICO

2.1. Marco Histórico

En el siglo XX hubieron bastantes experimentos sobre carros autónomos, tal es el caso del año 1926 donde una compañía de Milwaukee ubicada en Wisconsin creó un carro que era controlado mediante señales de radio, pero el experimento que fue el más trascendente de esa época fue en 1939 en la feria futurística, donde el ingeniero Norman Bel Geddes presentó una idea sobre carros eléctricos conducidos de manera autónoma en carreteras automáticas, lo que dio origen a que grandes empresas como General Motors tuvieran interés en el desarrollo de un carro autónomo Bejerano, P.G. (2014).

Asimismo, el eje central de esta tecnología se centra en las decisiones de la máquina, lo cual nos hace pensar: ¿Las máquinas pueden pensar?, esta misma incógnita fue auto formulada por Alan Turing en el año 1950 en el artículo "Computing Machinery and Inteligence" Turing, A.M. (1950), presentado en la revista Mind y en donde propuso una serie de pruebas para retar a una máquina a determinar si se podía considerar que esta tenga conciencia propia. Esta prueba se llamó el test de Turing y actualmente es usado en diferentes investigaciones. Luego, promediando el año 1956, se usa por primera vez el término inteligencia artificial de la mano del ingeniero informático Jhon McCarthy, que junto con sus colegas Claude Shannon y Marvin Minsky definieron esta palabra como la ciencia e ingeniería de "Hacer máquinas inteligentes".

Posteriormente, en 1982, el alemán Ernst Dickmanns logró que un automóvil se desplace de derecha e izquierda manteniendo una barra en posición vertical y con la ayuda de una cámara, y usando un microprocesador de 8 bits; esto dio inicio a un sinfín de ideas, las cuales con los avances tecnológicos de la época actual son posibles de implementar y ejecutar.

2.2. Investigaciones Relacionadas con el tema

Como punto de partida se analizan las bondades que ofrece el uso de una GPU en este proyecto, por lo cual se toma como referencia la investigación de NVIDIA (2015) titulada "GPU-Based Deep Learning Inference: A Performance and Power Analysis", que comenta cuales son las mejoras en términos de performance y consumo energético. Teniendo en cuenta que, en el año 2012, con el desarrollo de la red neuronal ALEXNET, se estableció un hito en la historia del Deep Learning lo que impulsó al uso de las redes neuronales a varios campos como puede ser el reconocimiento de patrones, conducción

autónoma, detección de enfermedades, etc. Esto fue lo que impulsó a los desarrolladores a trabajar con GPUs y dejando a lado el lento proceso de una CPU. Todo esto originó los optimizadores para trabajar con estas tecnologías, tal como la librería CUDNN basada en la arquitectura CUDA y nuevos tipos de datos como el FP16, lo que amplió la mejora en términos energéticos y de memoria en comparación con el FP32.Además, se hicieron pruebas en el proceso de inferencia de una red neuronal el cual es simplemente dar una entrada nueva a una red neuronal entrenada, para que deduzca el resultado final; para esto, es recomendable primero entrenar la red neuronal con un amplio BATCH SIZE, para luego probar la red con un variado grupo de imágenes y conocer su performance; como resultado final se concluyó que trabajar con una GPU para temas de inteligencia artificial y ramas afines, resulta beneficioso y brinda mejoras en la performance ya sea deinferencia como también energéticas, algo más para rescatar fue el uso de un sistema TEGRA X1 el cual rindió de una manera aceptable en comparación con otros dispositivosde mayor costo; en la tabla 1 se muestran las mejoras de trabajar con una GPU en comparación con una CPU.

Tabla 1

Comparación de performance entre CPU-GPU.

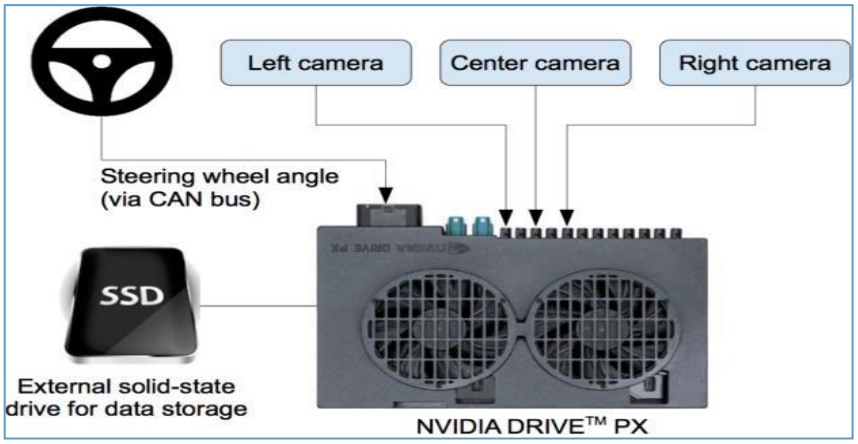
RED NEURONAL (ALEXNET)	**BATCH SIZE**	**TEGRA X1 (FP32)**	**TEGRA X1 (FP16)**	**CORE I7 6700K (FP32)**
INFERENCIA	1	47 img/seg	67 img/seg	62 img/seg
CONSUMO		5.5W	5.5W	49.7W
INFERENCIA/ CONSUMO		8.6 img/sec/W	8.6 img/sec/W	1.3 img/sec/W
INFERENCIA	128 (TEGRA) 48 (CORE I7)	155 img/seg	258 img/seg	242 img/seg
CONSUMO		6.0W	5.7W	62.5W
INFERENCIA/ CONSUMO		25.8 img/sec/W	45.0 img/sec/W	3.9 img/sec/W

Fuente: GPU-Based Deep Learning Inference A Performance and Power Analysis

Por otro lado, en el año 2016, la empresa NVIDIA utilizó una red neuronal convolucional que fue implementada en su proyecto End-to-End Deep Learning for Self-Driving Cars de conducción autónoma DAVE-2, el cual consiste en 3 cámaras ubicadas en los costados y centro del automóvil que se conectaba con una plataforma NVIDIA DRIVE PX, la cual recibía tanto las imágenes como el ángulo de giro de un timón usando el protocolo de comunicación CAN BUS. Luego de esto se continua con un proceso de data augmentation, donde se dio mejoras a la imagen para contar con una variedad de estas. Después de este proceso se pasó a entrenar este sistema con una red neuronal convolucional con 9 capas, en las cuales incluye convolución, una de normalización y flatten; al final se probó la red neuronal convolucional en un sendero cerca de Monmoth Country, los resultados fueron alentadores ya que se obtuvo un promedio de 90% de autonomía del sistema. A continuación, en la figura 1 se muestra el sistema de recolección de datos del proyecto DAVE 2:

Figura 1

Sistema de Recolección de Datos de DAVE 2.



Fuente: https://developer.nvidia.com/blog/deep-learning-self-driving-cars

Por otro lado, en el trabajo de Vijitkunsawat & Chantngarm (2020), seimplementaron 3 tipos de algoritmos para el manejo autónomo y así examinar superformance. Como primer paso se tuvo que escoger el algoritmo deseado (SVM, ANN-MLP, CNN-LSTM), luego se pasó a la parte de la generación de la data de entrenamientola cual fue tomada con una cámara y con una velocidad de 5 cuadros por segundo, así como también un tamaño de las fotos igual a 320*240 pixeles. En este caso, se obtuvo 3600 fotos de las cuales el 80% de estas fueron destinadas al entrenamiento, mientras que

el 20% fue para la validación del modelo; para obtener resultados se realizaron pruebas de 2 tipos, las cuales fueron con el circuito libre mientras que en la otra se colocó un obstáculo. De los datos obtenidos de las pruebas realizadas el mayor porcentaje de acierto obtenido fue con una CNN-LSTM con menor velocidad en el automóvil a escala, lo cual nos indica que el bloque LSTM mejora sustancialmente la inferencia dando un mejor resultado. Cabe resaltar que se puede mejorar esta con un mayor número de datos de entrada y usando un mejor motor de inferencia; a continuación, se muestran los resultados en las tablas 2 y 3.

Tabla 2

Comparación de arquitecturas sin obstáculos en la ruta.

SIN OBSTÁCULO			
ALGORITMO	**VELOCIDAD**		
	ASERTIVIDAD %		
	1 KM/H	**2 KM/H**	**3 KM/H**
SVM	82.5	74.4	68.2
ANN-MLP	78.1	72.6	67.3
CNN-LSTM	88.7	81.3	75.9

Fuente: Comparison of Machine Learning Algorithm’s on Self-Driving Car Navigation using Nvidia Jetson Nano.

Tabla 3

Comparación de arquitecturas con obstáculos en la ruta.

CON OBSTÁCULO			
ALGORITMO	**VELOCIDAD**		
	ASERTIVIDAD %		
	1 KM/H	**2 KM/H**	**3 KM/H**
SVM	79.8	72.7	67.3
ANN-MLP	73.4	70.1	66.1
CNN-LSTM	81.2	80.7	77.9

Fuente: Comparison of Machine Learning Algorithm's on Self-Driving Car Navigation using Nvidia Jetson Nano.

Asimismo, Yan Han and Erdal Oruklu (2017) trata en su investigación de como detectar señales de tráfico usando una red convolucional; para esta acción, el entrenamiento se hizo en una computadora con un procesador INTEL I3 y se implementó en un sistema embebido JETSON TX1; como primera parte del procesamiento de la imagen se encuentran las etapas de detección de señales usando una región de interés o ROI, para luego pasar a una detección de umbrales para determinar la validez de la ROI, lo que ayudaría al final a obtener una clasificación de las señales usando una red convolucional basada en LENET, para el entrenamiento; después de realizar este paso, se implementó la red neuronal en un sistema embebido NVIDIA TX1 el cual gracias a sus 256 NVIDIA CORES dan un buen soporte a las aplicaciones de AI, en este sistema se obtuvo un 96.2% de acierto en una imagen de 1300*800, el cual es aceptable y prometedor para futuras investigaciones como es el caso de la que se está desarrollando en este trabajo de tesis; por lo cual, la manera como se elige un posible candidato con el uso de umbrales se puede usar para poder optimizar el uso de memoria, así como también colocar diferentes valores para diversas formas de señales de tránsito.

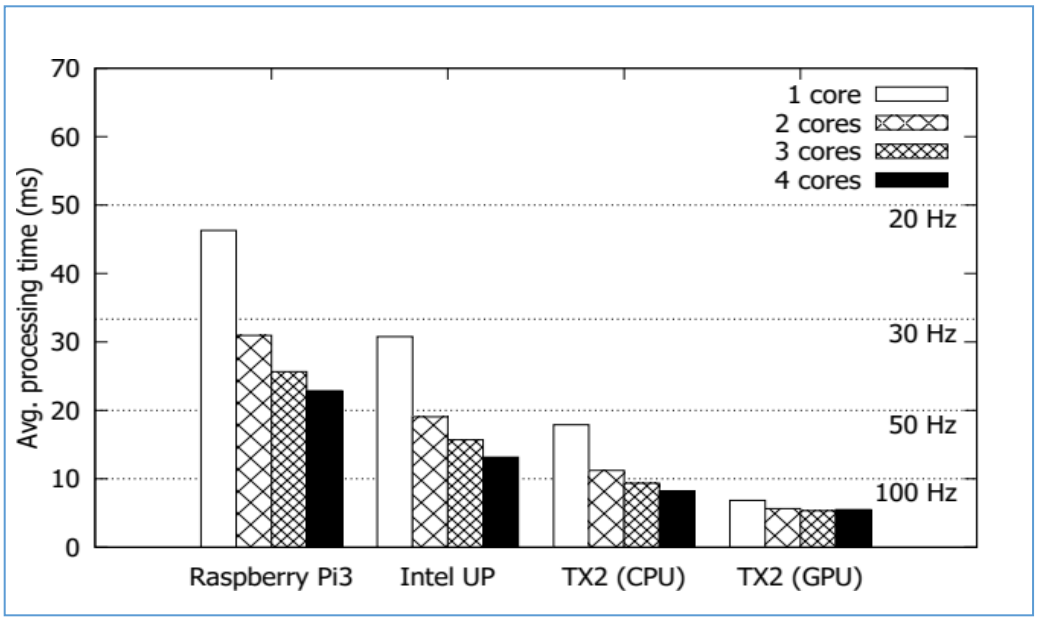
Por otro lado, en la investigación de Deepika (2017), se tuvo como objetivo clasificar obstáculos (peatones, automóviles, etc.) a partir de la segmentación de una imagen, delimitación de una región de interés usando la técnica CONVEX-HULL, así como encerrar un posible candidato para obstáculo y dar a conocer si este está dentro o fuera

del área de interés usando el POINT POLYGON TEST. A continuación, se define cada uno de estos procesos empezando con la segmentación de la imagen de entrada para luego realizar la obtención de una región de interés usando el algoritmo CONVEX-HULL; teniendo esta área bien definida se realiza un análisis para determinar si un objeto está dentro o fuera usando algoritmos como el CROSSING NUMBER o WINDING NUMBER, para las pruebas se colocó un peatón dentro de la ROI lo cual tuvo un 86.38% de asertividad dando a conocer que el peatón estaba dentro de la ROI, mientras que para un carro dentro de la ROI se tuvo un 96.67% de asertividad, cabe resaltar que estos resultados fueron obtenidos usando un sistema embebido NVIDIA JETSON TX1 donde se ejecutó el algoritmo.

En el proyecto DeepPicar (2018), usando la red neuronal de DAVE 2, embebida en una placa RASPERRY PI 3B, se logró desarrollar un sistema de conducción autónoma de bajo costo. Este funciona obteniendo una imagen de entrada la cual tenía que ser modificada para tener una dimensión de 200*66 pixeles y ser compatible con la red neuronal usada, con esta imagen se procedió a realizar el proceso de inferencia con la red neuronal arrojando un comando de giro que va hacia los motores, y así lograr cambiar la dirección del automóvil; luego de esto, se hicieron pruebas como el tiempo de inferencia del modelo, lo cual arrojó un tiempo promedio de 23.74 ms; además, para completar todos los procesos requeridos para la conducción se utilizaron los 4 núcleos del Cortex-A53 de la Raspberry Pi. A la misma vez, se realizaron pruebas en un sistema embebido Intel UP y otro de la marca NVIDIA Jetson TX2, de los resultados obtenidos claramente se observa que el tiempo de ejecución de la Raspberry Pi 3 (B) es mayor a las demás, dándole una desventaja notoria frente a otros sistemas embebidos. Sin embargo, observando el otro extremo se nota la gran diferencia de una TX2 al utilizar su GPU y dando un tiempo de ejecución de 6 ms aproximadamente frente a los 23.74ms, de esta manera se alcanzó una mejora de casi un 400% de menor tiempo de inferencia, lo que comprueba lo dicho en la investigación “GPU-Based Deep Learning Inference: A Performance and Power Analysis”. Para finalizar, se observa que el rendimiento es sorprendente y el costo de una TX2 ronda los $600, mientras que una Raspberry Pi 3 (B) cuesta $35; de esta manera, para un modelo a escala, el costo sería muy elevado por lo cual se debe buscar un equilibrio entre costo rendimiento. En la figura 2 se observan los resultados obtenidos de las pruebas.

Figura 2

Comparación de sistemas embebidos.

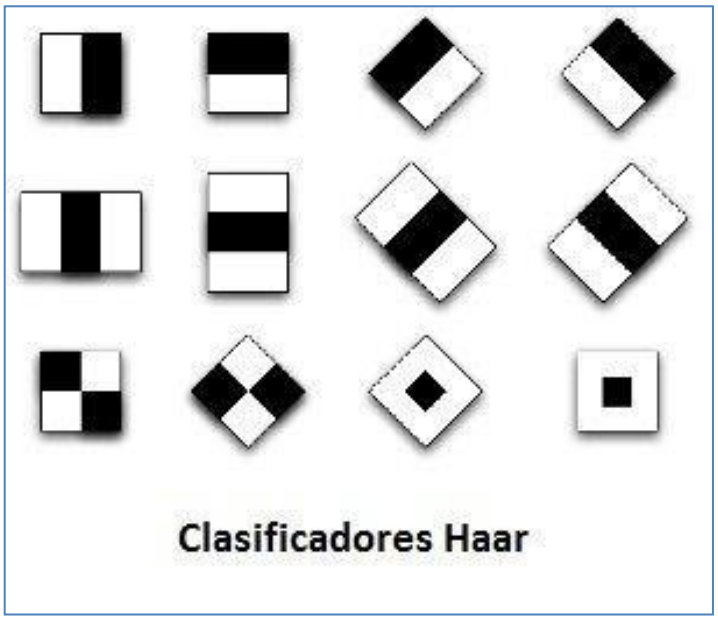


Fuente: DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car

Complementariamente, en la investigación realizada por Viola-Jones (2001), se propuso un sistema de detección de rostros usando un clasificador de tipo cascada, para eso primero se hace un enfoque a las características principales del rostro de una persona en una imagen, para lo cual debe analizarse una determinada sección con la ayuda de las denominadas Haar Features propuestas por Alfréd Haar en el año 1909, tal como son mostradas en la siguiente figura 3.

Figura 3

Clasificadores Haar.



Fuente: https://unipython.com/deteccion-rostros-caras-ojos-haar-cascad/

Por lo cual, con ellas se puede obtener las características de una determinada zona, y para eso se toma un ejemplo de una característica Haar Ideal. Ver la figura 4.

Figura 4

Característica Haar Ideal.

0	0	1	1
0	0	1	1
0	0	1	1
0	0	1	1

Fuente Propia

Donde, los pixeles blancos obtienen un valor de 0 y los negros 1 (caso ideal), pero en una imagen verdadera se tendrán diferentes tonalidades como se puede observar a continuación (figura 5).

Figura 5

Imagen real de prueba.

Fuente Propia

Lo que plantea Viola-Jones es comparar en cuanto el caso real se asemeja a lo ideal, para lo cual se tendrá que sumar los valores de cada pixel en el área delimitada por la característica probada (área blanca y negra), obtener el promedio de esta y luego restar estos valores, tal como se muestra en la siguiente ecuación. Asimismo, la variable "n" representa la cantidad de pixeles en la imagen, e I(x) es la intensidad o valor que posee cada pixel tomado.

$$\Delta = \frac{1}{n} * \sum_{negro}^{n} I(x) - \frac{1}{n} * \sum_{blanco} I(x) \qquad (1)$$

Aplicando esta fórmula en el caso ideal, obtenemos que $\Delta = 1$. Pero, si lo hacemos en la imagen real obtenemos un valor de $\Delta = 0.775 - 0.2125 = 0.56$, mientras este valor sea aproximadamente a 1 se puede decir que usando esa característica Haar se puede clasificar esa parte de una imagen, pero para poder ayudar a realizar el cálculo de suma de pixeles por zona se usó la técnica de imagen integral con la cual se mejoró el tiempo de cálculo al sumar los pixeles por zona. De esta forma, para poder utilizar esta técnica,

se muestra el siguiente ejemplo en la figura 6 (imagen original), mientras que en la figura 7 se observa la representación integral de la figura anterior.

Figura 6

Imagen de prueba.

4	3	4	1	5
1	4	4	4	6
3	9	3	2	5
2	4	5	8	7
3	5	7	3	3

Fuente Propia

Figura 7

Representación integral de la figura 6.

Fuente Propia

Por el motivo que se tiene que encontrarse clasificadores Haar, en toda la imagen se necesita realizar el promedio de pixeles en una determinada área; por lo cual, esta operación sin usar una representación integral necesitará de un algoritmo cuadrático para solucionarlo. De esta manera, dicha operación demandará más tiempo al algoritmo en una imagen de resolución mayor, por lo cual al usar una imagen integral la operación sería simplemente una constante, demorando menos tiempo y haciéndolo más rápido al momento de procesar; como ejemplo, se muestra el siguiente caso donde se desea determinar la suma de pixeles en un área determinada (ver la figura 8).

Figura 8

Área a determinar.

4	3	4	1	5
1	4	4	4	6
3	9	3	2	5
2	4	5	8	7
3	5	7	3	3

Fuente Propia

En este caso tenemos que hallar la suma de pixeles en esa área, dando lo siguiente: 4+4+4+9+3+2= 26. Tal como se aprecia en la figura 9.

Figura 9

Área a determinar usando imagen integral.

4	7	11	12
5	12	20	
8	24		
10	3		
1			

Fuente Propia

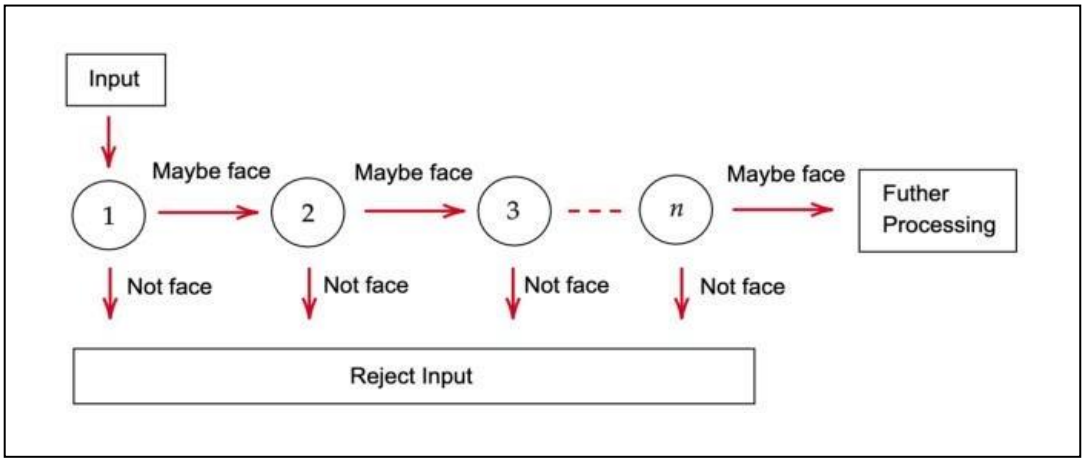
En este caso usando una imagen integral se tiene que hallar el valor de esa área, pero gracias a esta técnica se puede hallar este valor con la siguiente operación: 42-12-8+4= 26. Esto se debe a que cada nuevo valor operado es en sí, el valor de la suma de los pixeles ubicados arriba de cada posición y a la izquierda de este, dando en si una mejor técnica de cálculo en toda la imagen. Luego de este proceso, se continúa con el algoritmo Adaboost el cual sirve para poder seleccionar las mejores características de Haar de las más de 25000 existentes, las cuales pasan solo si se desempeñan mejor que otras al momento de ser escogidas al azar. Luego de su clasificación, estas se combinan en un clasificador fuerte que a su vez se compone de clasificadores débiles que constituyen en una combinación lineal para mejorar su performance. Ver la siguiente ecuación.

$$F(x) = a_1 f_1(x) + a_2 f_2(x) + a_3 f_3(x) + \ldots\ldots\ldots\ldots + a_n f_n(x) \qquad (2)$$

Donde $a_n\ f_n(x)$ se comporta como clasificador débil y $F(x)$ como un clasificador fuerte. Siendo $a_n\ f_n(x)$ el "n-ésimo" clasificador débil. Por lo cual, luego de obtener estos clasificadores se conectan en cascada con el propósito de realizar una clasificación, y si en alguna parte de la cascada no detecta el objetivo automáticamente lo descarta y pasa a otra entrada hasta que luego de una serie de verificaciones se obtiene lo deseado, en la siguiente figura 10 se muestra este proceso.

Figura 10

Clasificador en Cascada.



Fuente:https://towardsdatascience.com/understanding-face-detection-with-the-viola-jones-object-detection-framework-c55cc2a9da14

2.3. Bases Teóricas

2.3.1. Neurona artificial

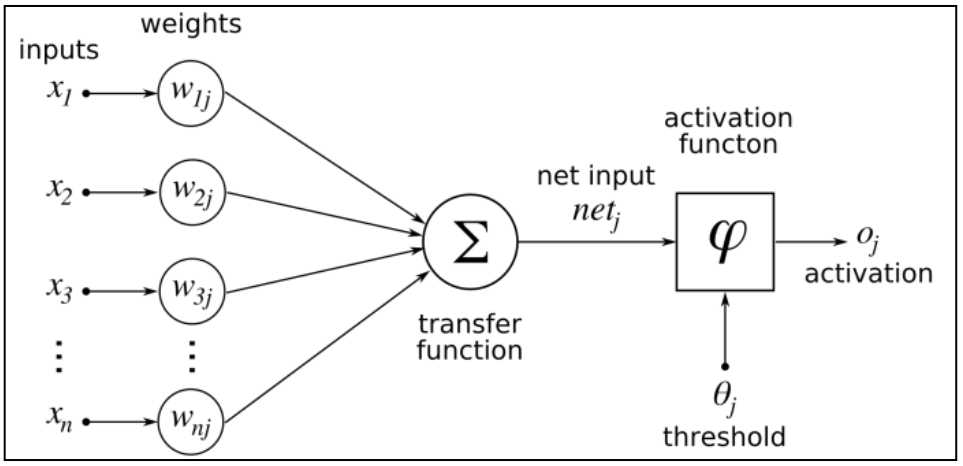
En el año 1986 David E.Rumelhart y James L.Mclelland (1987) proponen el modelo siguiente de neurona artificial, el cual cuenta con entradas (x_1, x_2, x_3, …, x_n) , y las cuales tienen pesos sinápticos (w_{1j}, w_{2j}, $w_{3\,j}$, …, $w_{n\,j}$) y vías sinápticos ($\boldsymbol{\theta_j}$). Por lo cual, estos valores juntos generan un potencial postsináptico. Luego, para poder obtener la salida se aplica una función de activación:

$$\boldsymbol{y = f * (\sum_{i=1}^{n} w_{i\,j} * x_i - \theta_j)} \qquad (3)$$

En la figura 11 se observa la representación de una neurona artificial en función a lo descrito en la ecuación anterior.

Figura 11

Representación gráfica de una neurona artificial.



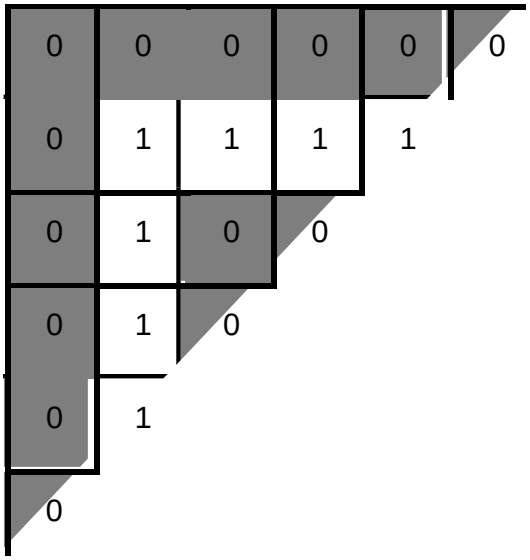
Fuente:https://www.researchgate.net/figure/Figura-1-Modelo-de-una-neurona-artificial_fig1_267247220

2.3.2. Red Neuronal Convolucional

En el año de 1989, Y. LeCun (1998) propone el modelo de una red neuronal de topología malla para el procesamiento de datos ya sea de una dimensión, 1D, como por ejemplo muestras en un intervalo de tiempo, o de dos dimensiones, 2D, como una imagen para aplicar una operación matemática denominada convolución. Esta última está basada en una multiplicación de una matriz con algunas de las capas de neuronas, para poder lograr una estructura especial con la cual se mantenga toda la data fundamental; con ello, se logra un aprendizaje y a la misma vez se reducen los parámetros de la red neuronal. Asimismo, en vez de pasar toda la data de neurona a neurona se utiliza una "ventana" que recorre toda la imagen de entrada, y solo pasa lo que recubre la dimensión de esta a la neurona siguiente. Con este proceso se puede capturar las características especiales de una imagen, tal como se observa en el ejemplo de la figura 12. En dicha figura, se reconoce la letra "o", para eso se aplica una ventana por toda la imagen que extraiga las características de la imagen.

Figura 12

Imagen de entrada.



Fuente propia

De esta manera, al aplicar una ventana de 3*3 (figura 13) en toda la imagen, se logra dividir momentáneamente en las regiones de color azul, amarillo, verde y rojo con un avance o también llamado “Stride” por 1 pixel, tal como se observa en las figuras 14 y 15.

Figura 13

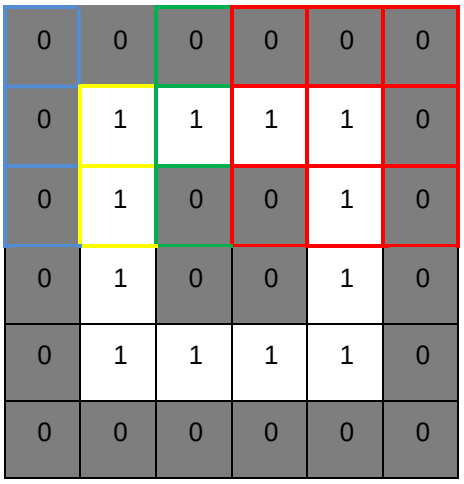
*Filtro de 3*3.*

1	1	1
1	0	1
1	1	1

Fuente Propia

Figura 14

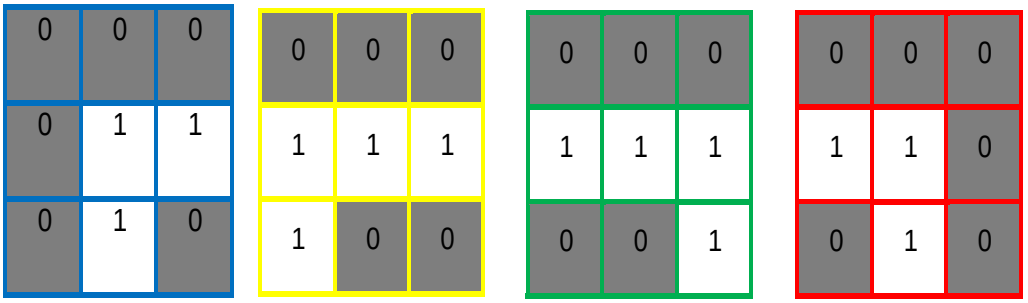
*Imagen de 6*6.*



Fuente Propia

Figura 15

*Características de la imagen de dimensión 3*3 antes de aplicar el filtro.*

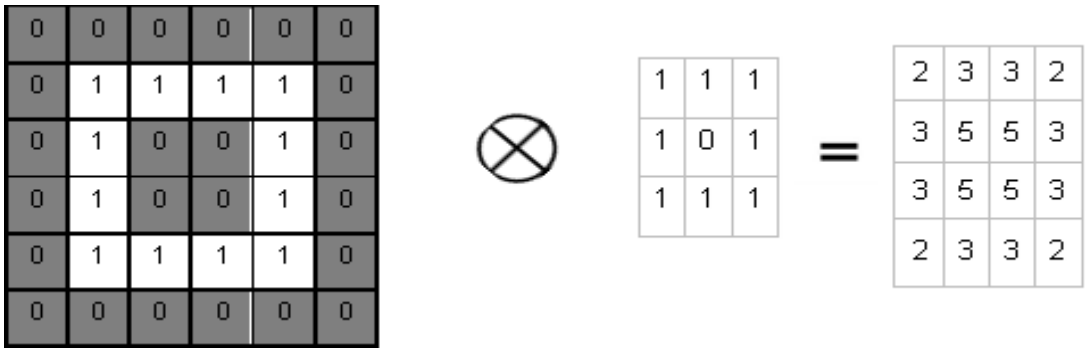


Fuente Propia

Luego, cada paso de la ventana da una característica de la imagen original, por lo cual se realiza la convolución usando un elemento multiplicador dando como resultado por cada característica obtenida, un valor que representará a una matriz de salida o también a un mapa de características. Ver la figura 16.

Figura 16

Proceso de convolución.



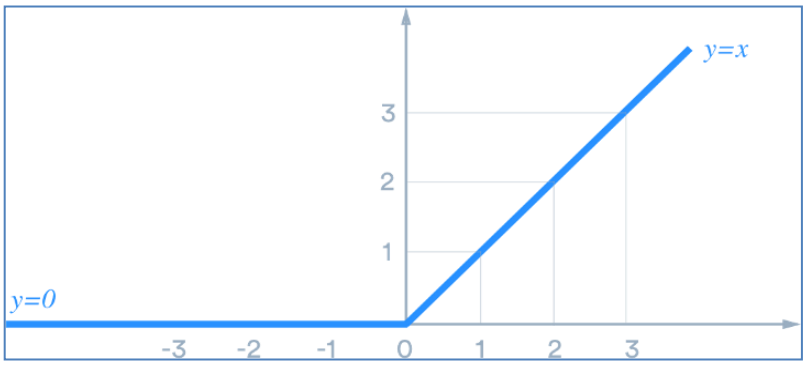
Fuente Propia

El resultado del mapa de características puede ser diferente según el tipo de filtro que se haya utilizado, lo cual permitiría obtener diferentes características de una imagen permitiendo hacer que el resultado sea aún más preciso.

Luego de la convolución se aplica una función de activación, y debido a la complejidad existente se opta por aplicar una función no lineal ya que la data en la mayoría de las veces no se ajusta a una recta; como ejemplo, se encuentra la función Relu que ayuda a hacer el procesamiento menos lineal debido a que las imágenes no obedecen a este tipo de patrón, y en consecuencia hace más robusto el procesamiento. En la figura 17 se muestra la función de activación Relu.

Figura 17

Función de activación Relu.

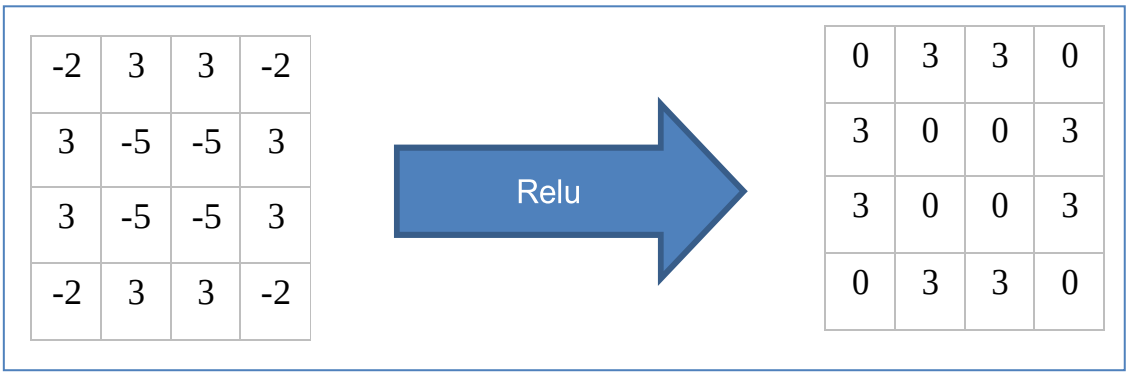


Fuente: https://ichi.pro/es/una-guia-practica-de-relu-202570656444919

De esta manera, si contamos una matriz de dimensión 4x4 y le aplicamos la función de activación Relu, el resultado sería lo mostrado en la figura 18.

Figura 18

Función Relu en una imagen.



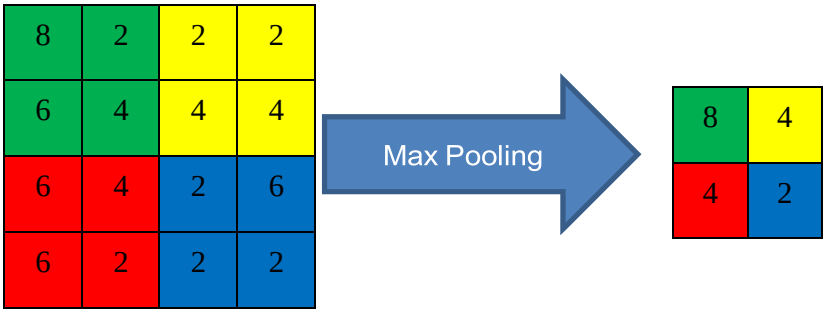
Fuente Propia

De esta manera, luego de obtener el mapa de característica (feature map) después de realizar la convolución y aplicar la función de activación no lineal, se continúa con la operación de pooling el cual es fundamental para las redes neuronales convolucionales ya que permite reducir notoriamente el gasto computacional; asimismo, genera invarianzade datos (por ejemplo, la imagen puede girar). Existen dos 2 tipos de operaciones de pooling, que a continuación se detallan.

- Max Pooling: Cuando se elige el valor de mayor dimensión de una determinada sección. A continuación, se aplica un operador Max Pooling con dimensión de 2*2 en un feature map de 4*4. Ver la figura 19.

Figura 19

Aplicación del Max Pooling en una sección de imagen.

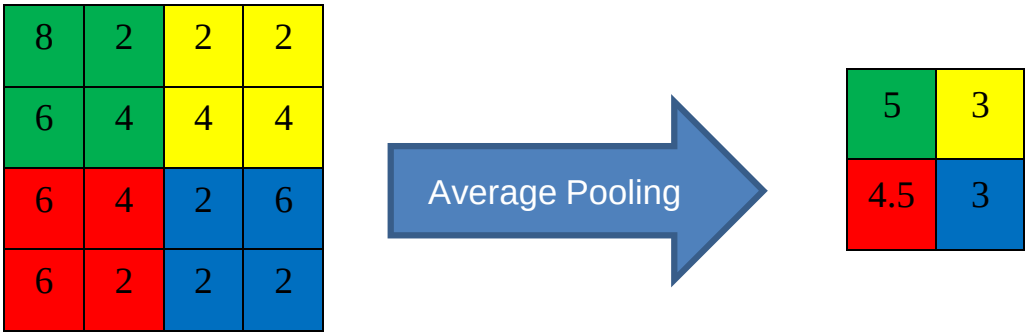


Fuente Propia

- Average Pooling: Cuando se elige el valor promedio de una determinada sección. A continuación, se aplica un operador Average Pooling con dimensión de 2*2 en un feature map de 4*4. Ver la figura 20.

Figura 20

Aplicación del Average Pooling en una sección de imagen.

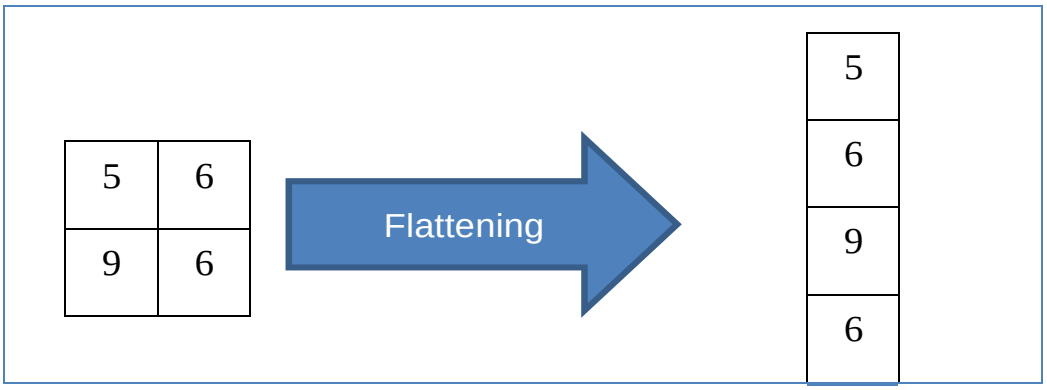


Fuente Propia

Luego, una vez obtenido el feature map con la aplicación del polling, se pasa a realizar un mapa de características agrupadas. Para lo cual, se realiza un operador flattening de la matriz obtenida. Ver la figura 21.

Figura 21

Operador Flattening en una sección de imagen.



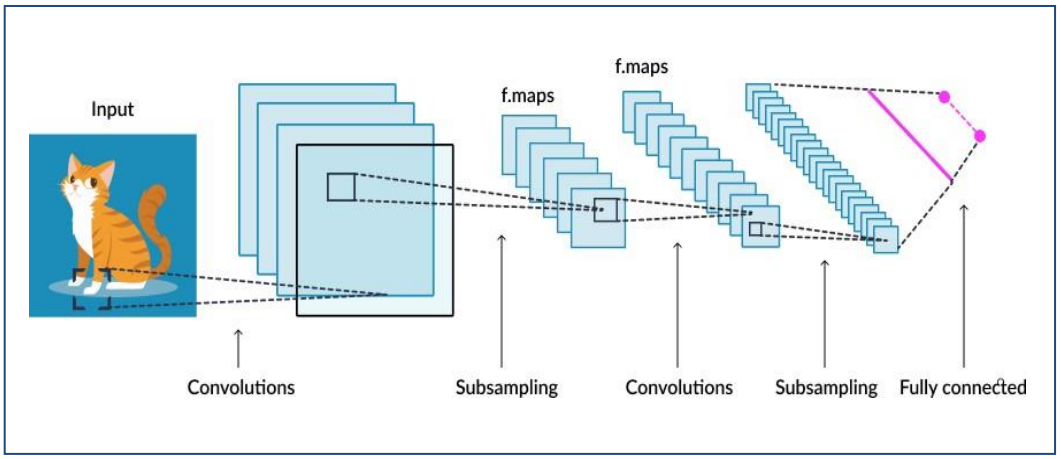
Fuente Propia

Finalizada la aplicación del flattening, se procede con colocar tal información a la entrada de una red neuronal la cual usa los datos obtenidos y realiza una mezcla de características, con la finalidad de encontrar patrones que permitan la clasificación o

regresión de una imagen. En la figura 22 se observa todo el proceso llevado a cabo por una red neuronal convolucional.

Figura 22

Representación gráfica de una red neuronal artificial convolucional.



Fuente: https://sandeep-bhuiya01.medium.com/disadvantages-of-cnn-models-95395fe9ae40

2.3.3. Métricas de Evaluación LOSS y VAL_LOSS

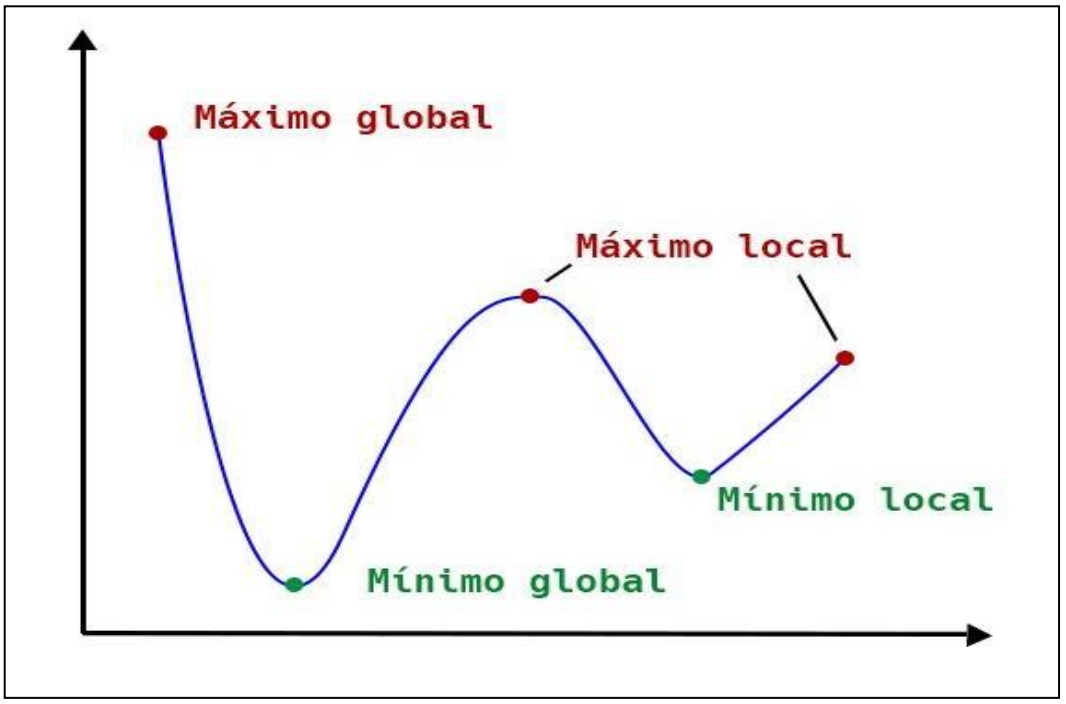
Como es clásico en cualquier proyecto que involucra redes neuronales, siempre es habitual el uso de las métricas para poder medir el desempeño de esta; por lo cual, en este proyecto se optó por utilizar la métrica LOSS que mide el error entre la salida del sistema y la deseada; es decir, mientras menos sea esta diferencia se va a tener un mejor resultado. Así como también, se optó por emplear la métrica VAL_LOSS que mayormente es usado como indicador en el caso que el sistema se encuentran con problemas de overfitting, lo cual otorgaría dificultades durante el entrenamiento.

2.3.4. Gradiente Descendente

Propuesto por Augustin Louis Cauchy, y es usado en Machine Learning para hallar mínimos de una función tales como: LOSS, VAL_LOSS. Para explicar esta técnica, seguidamente se muestra la figura 23.

Figura 23

Representación en 2D del Gradiente Descendiente.



Fuente: https://koldopina.com/gradient-descent/

De la figura anterior, dado que el objetivo es encontrar el mínimo, lo primero que se hace es colocar un punto al alzar y calcular la pendiente, si esta es positiva el punto subirá, pero si es negativa (como queremos) el punto bajará de acuerdo con la Learning Rate colocado en los parámetros de la red neuronal. De esta manera, haciendo este proceso se hallarán los puntos mínimos de la función, pero en algunos casos este punto se puede quedar atrapado no en el mínimo global sino en un mínimo local; de esta forma, no se alcanzaría el valor esperado y por lo cual se puede aún solucionar ajustando la Learning Rate o aplicando On Dropout en la red neuronal.

2.3.5. TensorFlow

TensorFlow es una biblioteca de código abierto que se basa en un sistema de redes neuronales. Esto significa que puede relacionar varios datos en red simultáneamente, de la misma forma que lo hace el cerebro humano. Por ejemplo, puede reconocer varias palabras del alfabeto porque relaciona las letras y fonemas. Otro caso es el de imágenes y textos que se pueden relacionar entre sí rápidamente, gracias a la capacidad de asociación del sistema de redes neuronales (TensorFlow, 2021).

2.3.6. Keras

Keras empezó siendo una librería que contenía frameworks más como TensorFlow, Theano, o CNTK, y cuya función principal es facilitar la definición, entrenamiento y uso de modelos de Deep Learning. En la actualidad, Keras está dentro de las funciones de TensorFlow facilitando su uso en nuevos modelos lo que ayuda a definir, entrenar y usar modelos. Pero también, se puede utilizar Keras en combinación con módulos de TensorFlow. Un caso de uso común es utilizar tf.data para generar un pipeline de entrada de datos, y utilizar tf.keras.layers para definir un modelo.

2.3.7. Jetson Nano

Cuenta con un CPU ARM Cortex-A57 MPCore de 4 núcleos (capaz de proporcionarnos 472 gigaflops de potencia), una GPU Nvidia Maxwell con 128 núcleos CUDA (capaz de ejecutar la librería de procesado de datos CUDA-X AI), 4 Gb de RAM, 16 GB de almacenamiento y 4 puertos USB 3.0. El consumo de la Jetson Nano oscila entre 5 y 10W, Nvidia Developer (2019). A continuación, en la figura 24, se muestra la imagen de la Jetson Nano utilizada en este trabajo de tesis.

Figura 24

JETSON NANO.

Fuente: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

2.3.8. Hiper-parámetros

En los modelos de Machine Learning se cuenta con 2 tipos de parámetros:

- Parámetros Entrenables, los cuales aprenden durante el entrenamiento.
- Hiper-parámetros, los cuales deben ser sintonizados antes del entrenamiento.

Dado que es posible manipular o sintonizar los hiper-parámetros, surge la posibilidad de abrir un sinfín de configuraciones con las cuales se puede entrenar un sistema; por lo tanto, es esencial escoger una configuración adecuada para el sistema; y, gracias a la librería Keras-Tuner (2021) se puede realizar 3 tipos de sintonización, los cuales son:

- Búsqueda Aleatoria: es el tipo de búsqueda más sencilla del grupo, aleatoriamente combina los hiper-parámetros en el espacio de búsqueda predeterminado.
- Búsqueda Hyperbanda: es una búsqueda aleatoria mejorada propuesta por Li, Lisha, y Kevin Jamieson (2016), en donde se optimiza el método anterior ejecutando de forma momentánea la selección con mejor iteración y así usarla en las siguientes optimizaciones.
- Búsqueda Bayesiana: se le considera un tipo de búsqueda probabilística la cual usa las optimizaciones pasadas para generar un modelo estadístico, con el cual va haciendo una búsqueda a partir de un mapa de probabilidades.

Gracias a estos métodos, se puede mejorar la eficacia de la red; y, además de eso,

realizar una comparativa entre los distintos tipos de búsqueda.

2.3.9. Atmega 328p

Microntrolador de la familia Atmel que dispone de registros de 8 bits, un ciclo de reloj máximo de 16 MHz, 8 ADCs de 10 bits y 6 pines PWM ideal para aplicaciones prácticas y de bajo consumo. Atmel (2021).

2.3.10. Python

Lenguaje de programación de código libre el cual es empleado en diferentes ámbitos, uno de ellos es la Inteligencia Artificial.

2.3.11. Dataset

Conjunto de información esencial para el entrenamiento de una red neuronal, se puede manipular con diferentes librerías como Pandas.

2.3.12. OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de software de visión artificial y aprendizaje automático de código abierto. OpenCV se creó para proporcionar una infraestructura común para aplicaciones de visión por computadora y para acelerar el uso de la percepción de la máquina en los productos comerciales. Al ser un producto con licencia BSD, OpenCV facilita que las empresas utilicen y modifiquen el código.

Asimismo, esta biblioteca tiene más de 2500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos de aprendizaje automático y visión por computadora clásicos y de última generación. Estos algoritmos pueden usarse para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en videos, rastrear movimientos de cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D a partir de cámaras estéreo, unir imágenes para producir una alta resolución imagen de una escena completa, encontrar imágenes similares en una base de datos de imágenes, eliminar ojos rojos de imágenes tomadas con flash, seguir los movimientos oculares, reconocer paisajes y establecer marcadores para superponerlos con realidad aumentada, etc. Además, OpenCV tiene más de 47 mil personas de usuarios, comunidad y número estimado de descargas superior a 18 millones. La biblioteca se utiliza ampliamente en empresas, grupos de investigación y organismos gubernamentales.

OpenCV (2021).

2.3.13. CUDA

Compute Unified Device Architecture o en sus siglas CUDA es una plataforma de computación acelerada la hace uso de una GPU para poder cálculos con carga computacional más difícil mientras que las demás líneas de código son procesadas por la CPU, dado que un CPU comercial tope de gama dispone de 16 núcleos y una GPU convencional posee miles de núcleos la hace más idónea para cargas pesadas es por eso que al usar la librería CUDA en nuestras compilaciones de código haremos el proceso mucho más rápido que sin esta ayuda. (CUDA Zone, 2021)

2.3.14. Tensor-RT

NVIDIA TensorRT™ es una plataforma para inferencia de aprendizaje profundo de alto rendimiento. Incluye un optimizador de inferencia de aprendizaje profundo y entorno de ejecución que ofrece baja latencia y alta capacidad de procesamiento para aplicaciones de inferencia de aprendizaje profundo el cual utiliza los tipos de datos tales como:

- FP32

También llamado formato de coma flotante de simple precisión, como su etiqueta dice contiene 4 bytes de información o 32 bits, 1 para el signo, otros 8 para el exponente y los restantes para la fracción, sus valores pueden ir desde $-3.4*10^{38}$ hasta $3.4*10^{38}$.

- FP16

Llamado como coma flotante de media precisión contiene solo 16 bits de información que se dividen en 1 para el signo, 5 para el exponente y los otros 10 para la fracción, este tipo de dato tiene un rango entre -65504 a 65504.

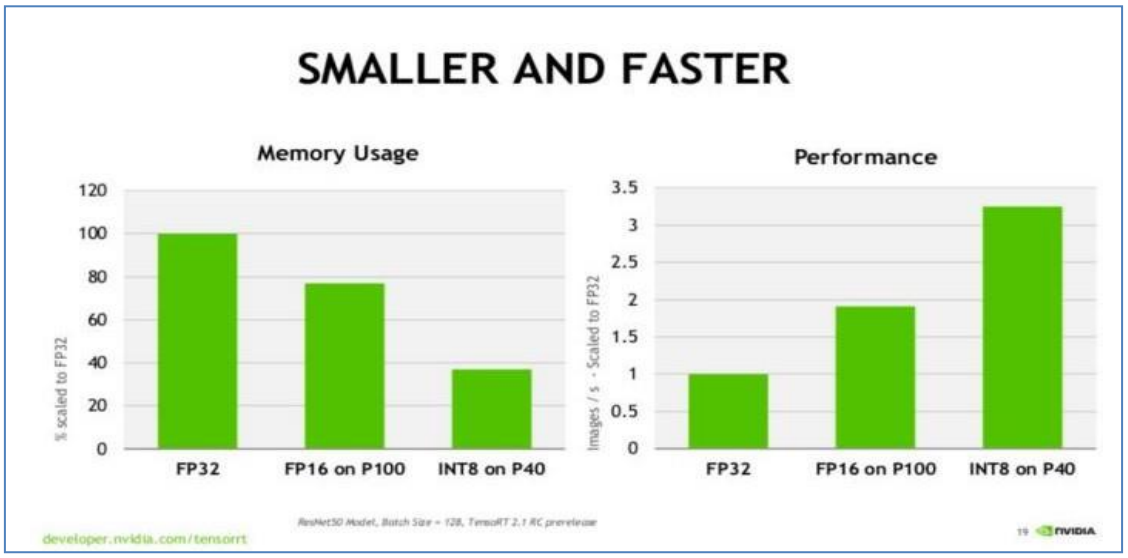
- INT 8

Es un tipo de dato entero con 8 bits con valores que van desde los -128 a los 127.

Teniendo en cuenta estas definiciones y siguiendo lo documentado en (Tensor-RT, 2021), la disminución del tamaño de dato es beneficioso para el uso de memoria y del ancho de banda aritmético del dispositivo, mejorando notoriamente el tiempo de inferencia. En la figura 25 se muestran las comparaciones de los usos de tipos de datos.

Figura 25

Comparación de Performance según tipo de dato.



Fuente: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html

2.3.15. Niveles de Conducción Autónoma

Tomando en cuenta lo dicho por "National Highway Traffic Safety Administration" (2021) existen 5 niveles de autonomía en automóviles:

- Nivel 0

Se requiere participación total de un humano al momento de realizar la conducción.

- Nivel 1

Un sistema de ayuda al conductor puede ayudar de manera parcial al conductor controlando solo una acción.

- Nivel 2

El sistema de ayuda al conductor posee mayor dominio del automóvil, puede controlar varias acciones a la vez y así ser más autónomo.

- Nivel 3

Cuenta con detección de objetos con lo cual ante una eventualidad este puede realizar una acción.

- Nivel 4

Ya no es necesario que un humano intervenga, el sistema realiza todas las acciones y responde a variaciones del entorno.

- Nivel 5

El automóvil cuenta ya con un sistema de automatización que en caso de algún fallo tendrá un respaldo con el cual podrá salir de cualquier imprevisto.

2.4. Diseño de la Investigación

El tipo de investigación es aplicada y tecnológica. En cuanto al método de investigación, es empírico y experimental.

2.4.1. Población de estudio

Conducción autónoma de vehículos a escala con aplicaciones de Inteligencia Artificial.

2.4.2. Diseño muestral

Un vehículo a escala con aplicación de una red neuronal convolucional.

2.4.3. Relación entre variables

La variable independiente es: Red Neuronal Convolucional.

La variable dependiente es: Conducción autónoma de vehículo a escala.

2.4.4. Técnicas e instrumentos de recolección de datos

El instrumento utilizado en la recolección de datos fue la Cámara USB con resolución de 8 Megapixeles, pero para mejorar la captura de imágenes se optó por hacer pruebas con un sensor IMX219 con una tasa de cuadros de hasta 60 fps a 1080p con lentes intercambiables las cuales fueron de ángulo de captura de 170° y otro de baja distorsión.

Asimismo, dado que se utilizó una red neuronal convolucional y un algoritmo de visión artificial, la técnica empleada fue la captura de imágenes que sirvió para someterla a la entrada de la red neuronal y generar una forma de regresión; posteriormente, se generó un comando de giro y se alcanzó el reconocimiento de las señales de tránsito señaladas anteriormente.

2.4.5. Procedimientos para la recolección de datos

Para la recolección de datos se emplearon rutinas de programación implementadas en el Lenguaje de Programación Python, los cuales consistieron en obtener las capturas de las imágenes con el ángulo de giro correspondiente usando la Liberia de código abierto

OpenCV; luego, esta data lograda sirvió para el entrenamiento de la red neuronal convolucional utilizada.

2.4.6. Técnicas de procesamiento y análisis de datos

Para procesar los datos se usó la librería “Pandas” con la cual fue posible leer los extensos dataframes así como también observarla y analizarla como también el uso del compilador Spyder en quinta versión con la cual podemos compilar y procesar información de manera más amigable como también el uso de archivos en formato .csv los cuales contenían la información de los entrenamientos también usamos la versión Python 3.9 para el siguiente proyecto.

CAPÍTULO III: DISEÑO DE INGENIERÍA

En este capítulo se aborda lo correspondiente al diseño de ingeniería para lograr el alcance de cada uno de los objetivos específicos planteados en esta tesis; asimismo, este desarrollo comprende primeramente el desarrollo de la implementación de los sistemas de alto y bajo nivel en Hardware y Software, donde se aborda lo referente a la implementación del automóvil y la creación de la parte software para su funcionamiento, para luego pasar al capítulo donde se obtiene el DATASET de entrenamiento de la red neuronal convolucional con el fin de realizar una conducción autónoma, y a su vez balancear esta información para hacerla más adecuada al proceso de entrenamiento. Concluido esto, se continúa con el desarrollo del algoritmo de visión artificial capaz de detectar elementos cotidianos en circuitos reales; finalizando este proceso, se pasa a la etapa crucial del proyecto la cual es el entrenamiento de la red neuronal convolucional con TENSORFLOW GPU (KERAS-TUNER) utilizando el DATASET optimizado, y a su vez la ejecución del framework TensorRT para mejorar su performance; teniendo como último paso la validación de la red neuronal convolucional en el circuito de pruebasa escala y toma de resultados del proyecto en general. A continuación, en la figura 26, semuestra un diagrama de bloques general del proyecto completo.

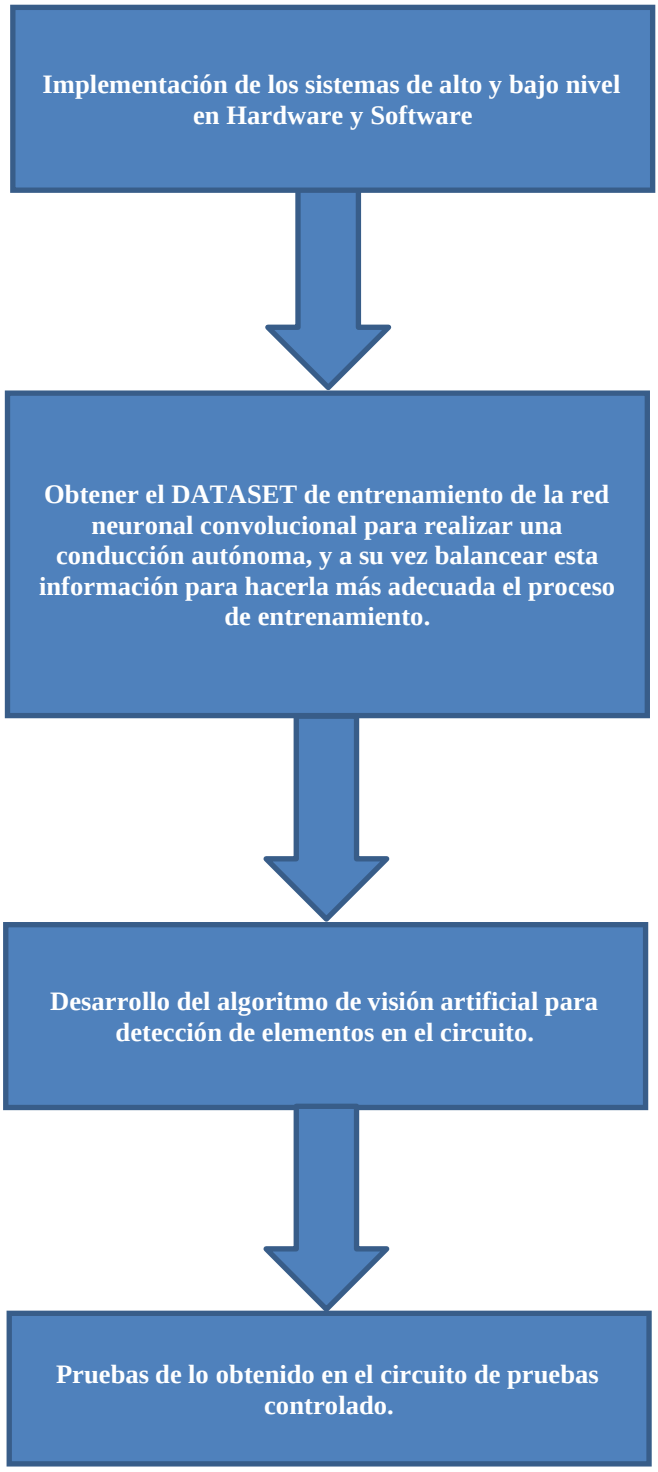
3.1. Implementación de los sistemas de alto y bajo nivel en Hardware y Software

3.1.1. Implementación del sistema de Alto Nivel

Para la implementación de un automóvil a escala primero se optó por seleccionar un sistema embebido el cual pueda soportar la carga computacional que demanda el presente proyecto, en el mercado actual podemos encontrar diferentes opciones tales como las más famosas tratándose de la Raspberry Pi 4 el problema de esta es la carencia de una GPU dedicada por lo cual se optó por buscar otra opción como es la Jetson Nano la cual si cuenta con una GPU dedicada Maxwell con 128 CUDA cores teniendo en cuenta que el uso de una GPU en aplicaciones de AI es fundamental para obtener mejores resultados y así no tener problemas al ejecutar la red neuronal convolucional deseada; luego, en la tabla 4 se observan las comparaciones de especificaciones técnicas entre los 2 sistemas embebidos.

Figura 26

Diagrama de bloques general del proyecto completo.



Fuente propia

Tabla 4

Comparación de sistemas embebidos.

	Raspberry Pi 4	**Jetson Nano**
CPU	Quad Core ARM Cortex-A72 1.5 GHz	Quad Core ARM Cortex-A57 1.42 GHz
GPU	Broadcom VideoCore VI	NVIDIA Maxwell 128 CUDA CORES
MEMORIA	4GB LPDDR4	4GB LPDDR4
GPIO	40 Pinout	40 Pinout
PRECIO	S/. 325.00	S/. 750.00

Fuente Propia

Luego de escoger el sistema Jetson Nano como eje central del proyecto se continuó con la selección de los demás componentes, uno de ellos fue el módulo PCA 9685 el cual sirvió como un header de 16 pines para conectar lo referente al control de los motores. Esta decisión se tomó dado que la corriente que fluye por el GPIO de la Jetson Nano no es suficientemente grande para poder realizar el control de elementos externos necesarios para el funcionamiento, dada esta razón se optó por escoger el módulo PCA 9685 el cual usa el protocolo de comunicación I2C que también es encontrado en la Jetson Nano por los pines 3 y 5 del header, y por donde se conectaron ambos dispositivos para poder tener una comunicación entre la Jetson Nano y su nuevo header de conexión. Ahora, para poder simular un pulso en alta y baja se usó la librería adafruit_pca9685 en lenguaje Python, la cual se controló mediante el duty cycle de una onda PWM la simulación de estos estados; por ejemplo, si se desea un estado en alto el duty cycle de la PWM debe ser 100%, y sabiendo que el PCA 9685 maneja 16 bits, el estado en alto debería llenar todos los bits. Lo cual dio como valor para obtener un estado en alto, la siguiente cantidad: $2^{16} = 65536$.

Po otro lado, para el caso de un estado en nivel bajo se tuvo un duty cycle de 0%, por lo cual el valor que se ingresó fue el código 0. Sabiendo esto, se pasó a la parte de arranque del automóvil a escala, para lograr esta tarea se utilizó un driver L 298N con el cual fue posible mover los motores DC de hasta 2ª. Es así como, en este módulo se conectaron 2 motores reductores DC de 9V que son controlados por los pines IN1, IN2, ENA (responsables del motor derecho) e IN3, IN4, ENB (responsables del motor izquierdo).

Los pines IN1, IN2, IN3 e IN4 determinaron la dirección de giro del motor, tal como se observa en la tabla 5.

Tabla 5

Modos de Operación del driver L298N.

Pines IN1, IN3	**Pines IN2, IN4**	**ESTADO DEL MOTOR**
X	X	MOTOR APAGADO
0	0	MOTOR FRENADO
0	1	GIRO HACIA ADELANTE
1	0	GIRO HACIA ATRÁS
1	1	MOTOR FRENADO

Fuente Propia

Por otro lado, los pines de ENA y ENB se encargaron de controlar la velocidad del motor mediante una señal PWM, lo cual es una ventaja ya que gracias al módulo PCA 9685 fue posible generar este tipo de ondas; por ejemplo, para una velocidad del 50% el valor en decimal en código corresponde a:

$$65536 * 0.5 = 32768$$

Sabiendo que el PCA 9685 tiene 65536 bits

Luego, para la dirección del automóvil se contó con un servomotor SG90 el cual proporcionó un giro a la derecha, izquierda o centrado para mantener la dirección del automóvil. Este servomotor se controló gracias al módulo PCA 9685 que junto con su librería fue posible realizar movimientos con un determinado ángulo de giro proveniente de un joystick. Luego se contó con una cámara de video, la cual tuvo la finalidad de obtener una imagen de entrada para poder realizar la regresión del ángulo de giro en ese momento; y, como último elemento se tuvo una batería externa que sirve de alimentación para el sistema de alto nivel.

De esta forma, todos estos componentes formaron parte del sistema de alto nivel el cual se puede apreciar en la figura 27. Además, todo esto fue controlado por módulos de programación escritos en el lenguaje Python los cuales forman parte de la tarjeta Jetson Nano. A continuación, se describen:

- Módulo Joystick

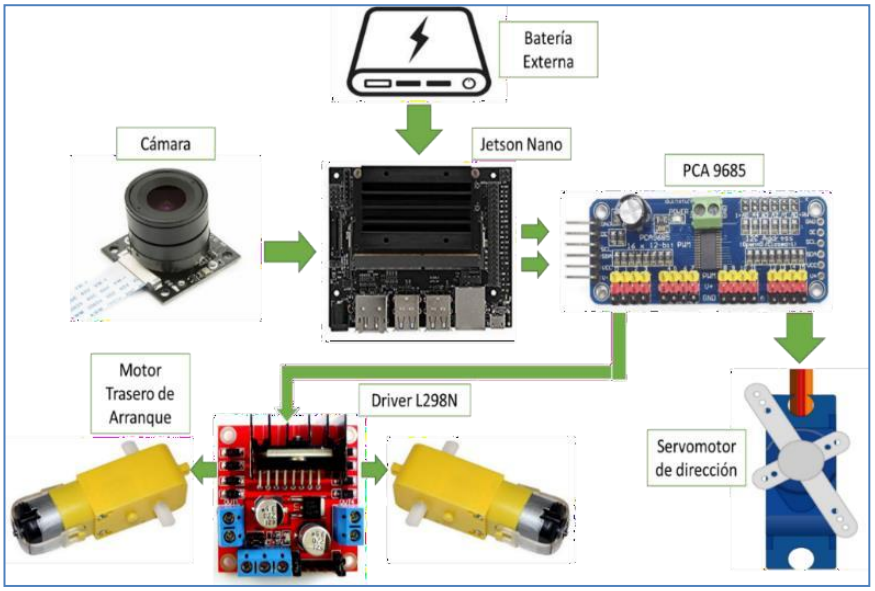
Encargado de leer el valor proveniente del joystick analógico de un gamepad, el cual tiene valores de -1 a 1 para el control del automóvil; estos comandos determinaron el ángulo de giro del automóvil desde 0° a 180° grados, por lo cual fueron convertidos previamente a un ángulo antes de ser suministrados hacia el servomotor. Dado esto, se realizó una regresión lineal utilizando las siguientes expresiones matemáticas. Donde, "m" es la pendiente de la recta, "b" la intersección por el eje de las ordenadas, "x" el conjunto de datos de las abscisas, e "y" el de las ordenadas.

$$m = \frac{\sum x \sum y - n \sum xy}{(\sum x)^2 - n \sum x^2} \quad (4)$$

$$b = \bar{y} - m\bar{x} \quad (5)$$

Figura 27

Hardware usado en el sistema de alto nivel.



Fuente Propia

Haciendo uso de la relación de 0° es igual a -1 y 180° es 1, y a su vez usando las expresiones 4 y 5 se obtuvo lo siguiente:

$$m = 90\,, \qquad b = 90$$

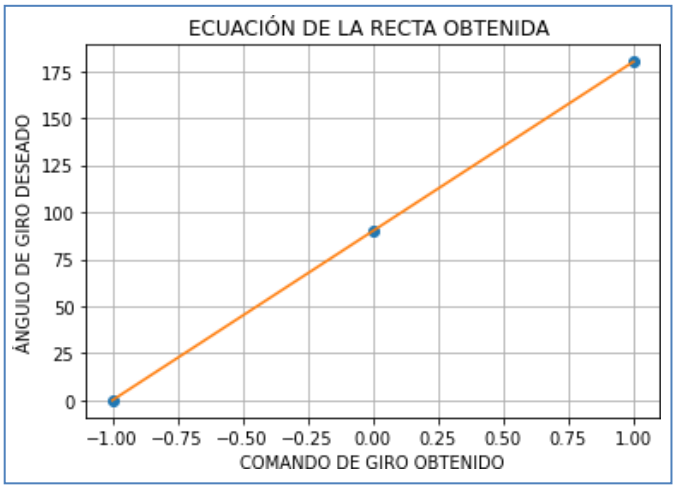
De esta forma, se obtuvo la siguiente ecuación de la recta y la correspondiente figura

28 donde se observan los datos originales y la aproximación de la ecuación.

$$y = 90x + 90 \quad (6)$$

Figura 28

Ecuación de la Recta.



Fuente Propia

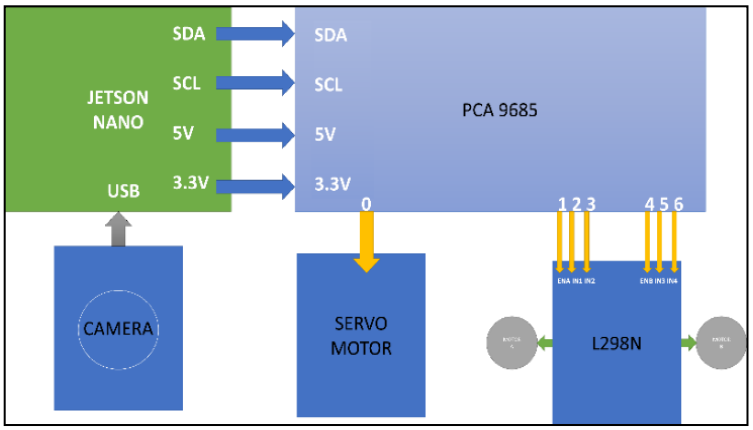
Según el coeficiente de determinación R^2 obtenido de la ecuación anterior, el valor de 0.999 nos establece una aproximación confiable y segura para determinar el ángulo de giro deseado. Teniendo estos datos, se continuó con el siguiente módulo.

- Módulo Motores

Encargado de controlar la velocidad de los motores traseros para la aceleración del automóvil; como también, controla la dirección según el valor dado por el módulo joystick para el giro en las direcciones de izquierda o derecha según el ángulo de giro. Este módulo contiene además una función la cual requirió de los pines PWM del PCA 9685 donde fueron conectados los controles del driver encargado del control del motor de la derecha, utilizando los pines 5 y 6 para el modo de operación y el 4 para regular la velocidad mediante una PWM; por otro lado, el motor de la izquierda utilizó los pines 2 y 3 para el modo de operación y el 1 para regular la velocidad mediante una PWM. Además, el servomotor utilizó el campo de pines 0 para su funcionamiento; en la siguiente figura se muestra el lugar de conexión utilizado en este proyecto.

Figura 29

Conexiones del sistema.



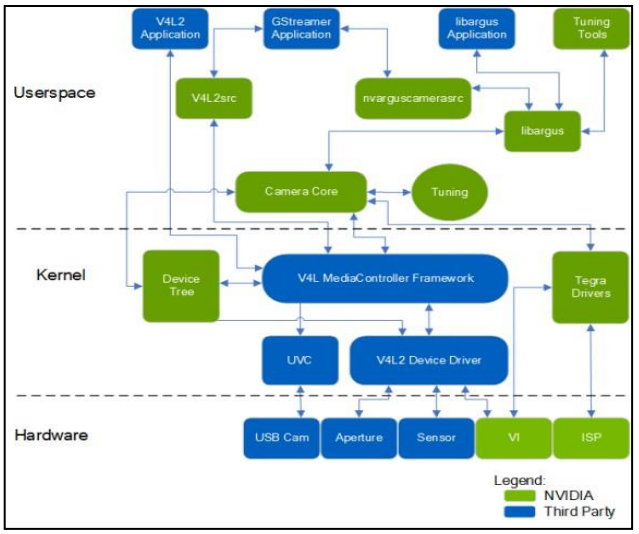
Fuente Propia

- Módulo Cámara

Encargado de integrar la cámara con el sistema de alto nivel. Para lograr esto se hizo uso de un pipline utilizando la aplicación Gstreamer, el cual se muestra en la figura30. En esta figura se observan las diferentes formas de obtener una imagen haciendo uso de un dispositivo JETSON de la marca NVIDIA.

Figura 30

Pipline Gstremear.



Fuente: https://forums.developer.nvidia.com/t/gstreamer-nvarguscamerasrc-and-v4l2src-elements-comparison-on-flowchart/74875

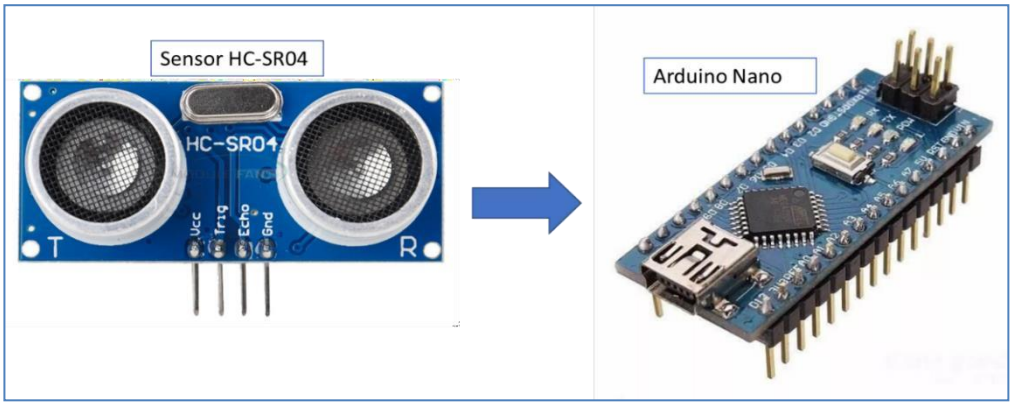
Según este pipline primero se tiene la fase de hardware, donde se encuentra la fuente de video que en este caso fue un sensor IMX 219 que se conectó a la aplicación ISP y encargada de realizar tareas como debayer, corrección de color, contraste de blancos, etc. Luego, se pasó a Tegra con los cuales se optimizó el proceso de decodificación de información; una vez completada esta operación se continuó con la información a Camera Core que va de la mano con libargus para poder suministrar la imagen a la aplicación. En este proyecto se utilizaron 2 lentes distintos, una de baja distorsión y otra de tipo angular; en la tabla 6 se muestran las características de cada lente utilizado.

3.1.2. Implementación del sistema de Bajo Nivel

Como siguiente paso de la implementación se desarrolló otro sistema de tipo bajo nivel, el cual se encargó de enviar datos de un sensor ultrasonido HC-SR04. Es así que, este sensor tuvo el trabajo de enviar 8 pulsos con una frecuencia de 40 MHz por el emisor piezoeléctrico comandado por el pin Trig, los cuales colisionan en una superficie entre los 4 a 450 cm de distancia. Luego de esto, son captados por el receptor y se acciona un estado alto en el pin Echo hasta que se termine la recepción. De esta forma, calculando el tiempo transcurrido se procedió a obtener la distancia entre el sensor ubicado en la parte delantera del automóvil y un objeto delante de este. Para esta tarea se usó un microcontrolador ATMEGA 328P embebido en una placa de desarrollo Arduino Uno Nano, con la que usando los registros internos de esta y su periférico de comunicación serial se enviaron los datos obtenidos hacia la Jetson Nano, en donde el módulo central se encargó de captar dichos datos para realizar una determinada acción. A continuación, en la figura 31 se muestra una representación de este sistema.

Figura 31

Hardware del Sistema de Bajo Nivel.



Fuente Propia

Tabla 6

Diferencias entre lentes.

	IMX 219 Wide Angle	**IMX 219 Low Distortion**
Resolución	**8 Megapixeles**	**8 Megapixeles**
Frame Rate	21 fps a 8 MP, 60 fps a 1080 p, 180 fps a 720 p.	21 fps a 8 MP, 60 fps a 1080 p, 180 fps a 720 p.
Campo de Visión Horizontal	145° grados	75° grados
Campo de Visión Vertical	175° grados	105° grados
Precio	S/. 236.00	S/. 236.00

Fuente Propia

Este sistema se comunica de manera serial con el del alto nivel usando un código de programación desarrollado en C++ en el ATMEGA 328P, el cual tiene la tarea de usar los registros de este microcontrolador y capturar los datos del sensor ultrasonido. Este cálculo fue realizado de la siguiente manera. Entonces, conociendo que:

$$D = V * T \tag{7}$$

Donde:

D: Distancia del sensor al objeto.

V: Velocidad del sonido (343 m/s).

T: Tiempo de viaje de la señal en microsegundos (µSeg).

Se procedió a realizar una conversión para el cambio de unidades:

$$D = \left(\frac{343m}{s}\right) * \left(\frac{1seg}{\frac{1000000}{\mu Seg}}\right) * 1\mu Seg * T \tag{8}$$

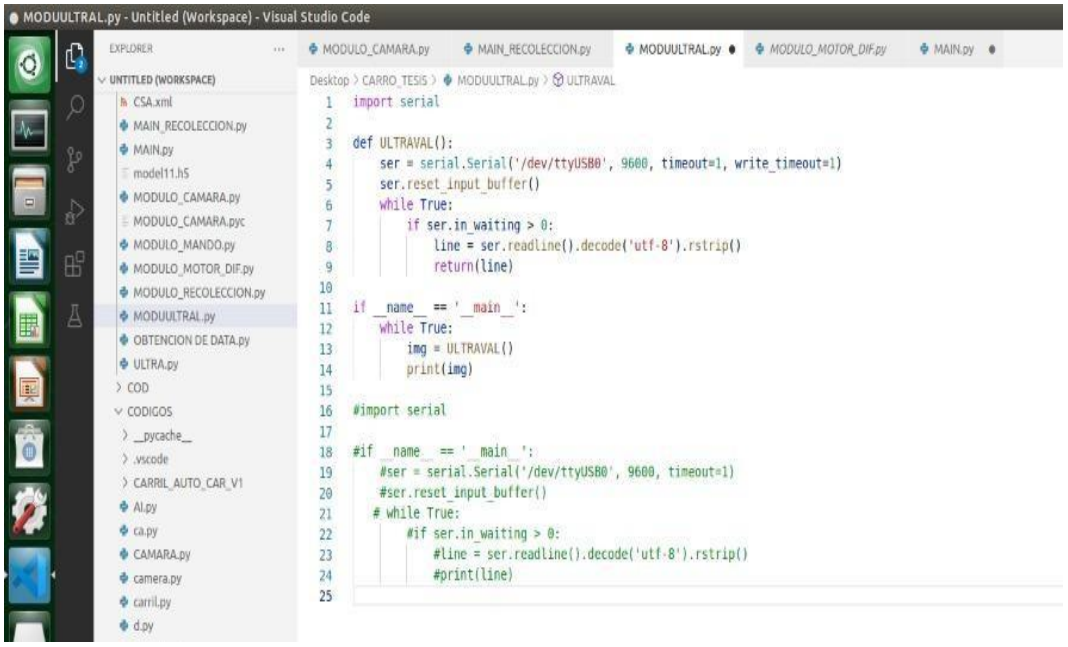
Luego, con esta expresión solo se tiene el tiempo de ida, por lo cual se duplicó la expresión para poder obtener el tiempo total de viaje de los pulsos de onda enviados por el ultrasonido. De esta manera, se tuvo como expresión final la siguiente.

$$D = 2 * (\frac{343m}{s}) * (\frac{1seg}{1000000\ us}) * 1us * T \quad (9)$$

Esta data del sensor fue enviada de manera serial hacia la tarjeta Jetson Nano donde se encuentra un módulo escrito en el lenguaje de programación Python que contiene la librería pyserial para poder recepcionar los datos. En la figura 32 se muestra el código utilizado en la programación dentro del ATMEGA 328p y dentro de la Jetson Nano, los cuales son comunicados por un puerto serial con un baudrate de 9600 con 8 bits de tamaño de trama, y 1 bit de parada y no paridad.

Figura 32

Código de Recepción de Datos del ATMEGA 328p en la Jetson Nano.



Fuente Propia

De igual manera, se tuvo que configurar el ATMEGA 328p para poder entablar una comunicación serial entre los 2 dispositivos. Para ello, primero se tuvo que realizar una programación de los registros internos del microcontrolador usando el periférico USART0; este tuvo que ser configurado con un BAUDRATE a 9600, para lo cual se utilizó la figura 33 que proporciona el fabricante.

Figura 33

Configuración del BAUDRATE según modo de configuración.

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Asynchronous normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous double speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous master mode	$BAUD = \frac{f_{OSC}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$

Fuente Atmega

De la figura anterior, se aprecia que existen varias configuraciones posibles de utilizar para la transmisión; sin embargo, se optó por utilizar el modo asíncrono de doble velocidad. Por lo cual, se tuvo que hacer una configuración en el registro UCSR0C colocando los bits UMSELL00 Y UMSELL01 a 0, y generando que el periférico pase a modo asíncrono; mientras que con el registro UCSR0A colocando a 1 el bit U2X0 se otorgó una doble velocidad de transmisión. Una vez realizada esta operación, se halló el valor requerido que fue almacenado en el registro UBRR0 para obtener un BAUDRATE de 9600 y el cual se halló con la siguiente operación.

$$UBRR0 = \frac{\frac{F\_CPU}{8}}{9600} - 1 \qquad (10)$$

Luego de esto se tuvo que seleccionar la forma en la cual se iba a transmitir la información, para eso se seleccionó el tamaño de la información a utilizar. Por lo cual, en los registros UCSR0C y UCSR0B, y en los bits UCSZ02, UCSZ01 y UCSZ00, se colocó la combinación necesaria para seleccionar un tamaño de 8 bits y se desactivó la paridad con los bits UPM01 y UPM00; y, como configuración final, se colocó un bit de parada con el bit USBS0. A continuación, en las figuras 34, 35, 36 y 37, se muestran las configuraciones de cada registro.

Figura 34

Configuración del modo de operación del periférico.

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, even parity
1	1	Enabled, odd parity

Fuente Atmega

Figura 35

Configuración bit de parada del periférico.

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

Fuente Atmega

Figura 36

Configuración de paridad de operación del periférico.

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)[1]

Fuente Atmega

Figura 37

Configuración de tamaño de data enviada.

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Fuente Atmega

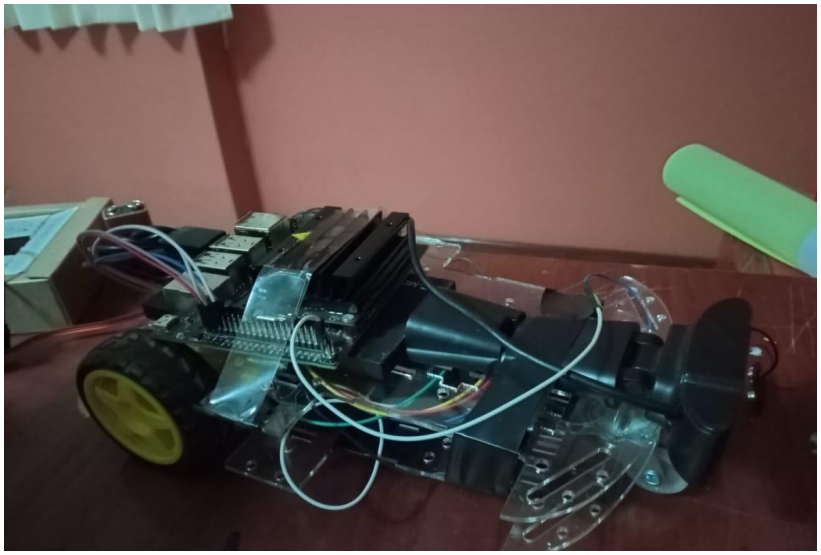
Habiendo finalizado con este proceso, se continuó con la rutina de envío de información. Para ello, previamente se convirtió la data obtenida por el ultrasonido utilizando la sentencia sprintf; es decir, se colocó la variable que contiene la información en formato de carácter. Luego de eso, en el registro UCSR0A se encontró el bit UDRE0. Entonces, cuando este tomó el valor de 1 indicó que el buffer UDR0 donde se almacena la data a ser enviada se encuentra libre; por lo cual, usando una sentencia while se hizo la carga de los datos y se enviaron con el formato ya establecido en la configuración mencionada anteriormente.

Inmediatamente, después de tener las configuraciones hechas en ambos dispositivos, se procedió a la conexión de los 2 elementos mediante un cable USB y se realizó una prueba de comunicación. Luego, el valor obtenido de este proceso se almacenó en un módulo con el cual al invocarlo dio a conocer a cuantos centímetros de distancia se encuentra un objeto.

Después de haber seleccionado el hardware y realizado el software para el funcionamiento, se escogió un chasis que pueda soportar todos los requerimientos necesarios. Al inicio, se optó por una solución rústica, la cual cumplió el objetivo, pero no era lo adecuado por ello se decidió emplear un chasis dedicado para este tipo de proyectos. A continuación, en las figuras 38 y 39 se muestran los prototipos en su versión rústica, así como también en su versión final ensamblado y listo para los demás objetivos.

Figura 38

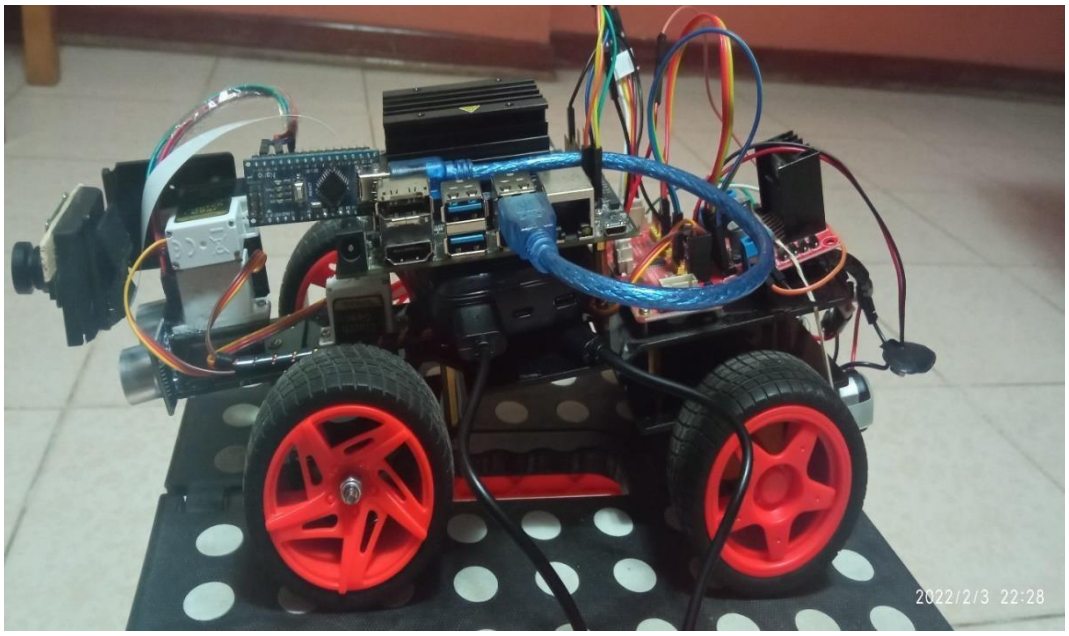
Modelo inicial del sistema.



Fuente Propia

Figura 39

Modelo final del sistema.



Fuente Propia

3.2. Obtención del DATASET de entrenamiento de la red neuronal convolucional

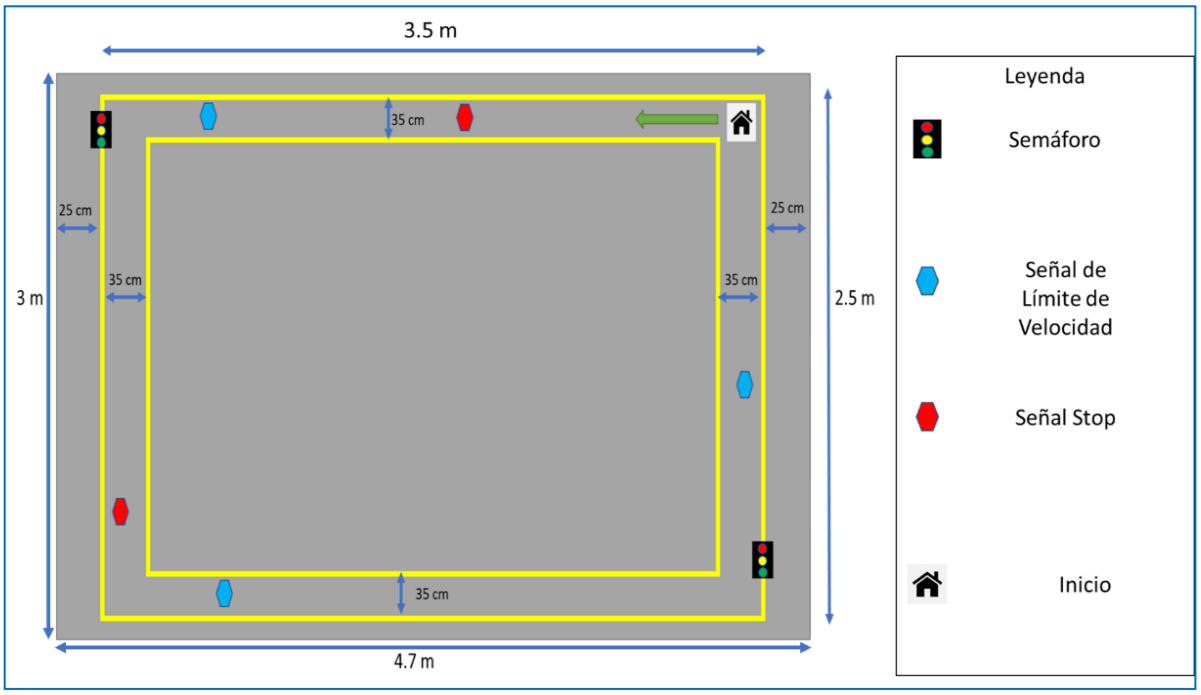
Esta obtención fue para realizar la conducción autónoma, y a su vez balancearla para hacerla más adecuada el proceso de entrenamiento.

3.2.1. Dimensionamiento del circuito de pruebas y estructura

El circuito de pruebas utilizado tiene una medida de 4.7 metros de largo por 3.5 metros de ancho. Dentro de este se tiene un carril el cual está delimitado por 2 bandas de color negro, las cuales separan el área de interés con el resto del circuito; además, cuenta con 2 semáforos ubicados en los extremos, 2 señales de alto y 3 señales de límite de velocidad. A continuación, en la figura 40, se muestra un plano del circuito de pruebas diseñado y mostrando la ruta que se realiza, mientras que la figura 41 representa el montaje final del circuito.

Figura 40

Plano del Circuito de Entrenamiento.



Fuente Propia

Figura 41

Montaje Final del Circuito de Entrenamiento.

Fuente Propia

3.2.2. Creación de Módulos para la obtención de la data de entrenamiento

Dado que se trabaja con una red neuronal convolucional, se requirió de la información de esta para poder entrenarla. Para ello, se realizaron módulos en el lenguaje de programación Python y en unión a los módulos realizados en el objetivo anterior, se logró obtener un DataSet de entrenamiento. Estos, se listan a continuación.

- Módulo Recolección

Realiza las funciones para poder capturar los valores de imagen del momento de la conducción, con una frecuencia de 30 milisegundos por foto y el valor generado por el joystick en ese instante. Además, toda esa información fue almacenada en un archivo .csv el cual sirvió de registro de la imagen obtenida y su atributo de giro. También, se implementó una función para que el módulo sea capaz de crear sesiones de recolección cada vez que era invocado, y así lograr el almacenamiento de dicha información en carpetas separadas. Asimismo, facilitar la selección de varias sesiones de entrenamiento para dicha etapa, y tener una mayor selección de datos para las demás etapas de entrenamiento.

- Módulo Main_Recolección

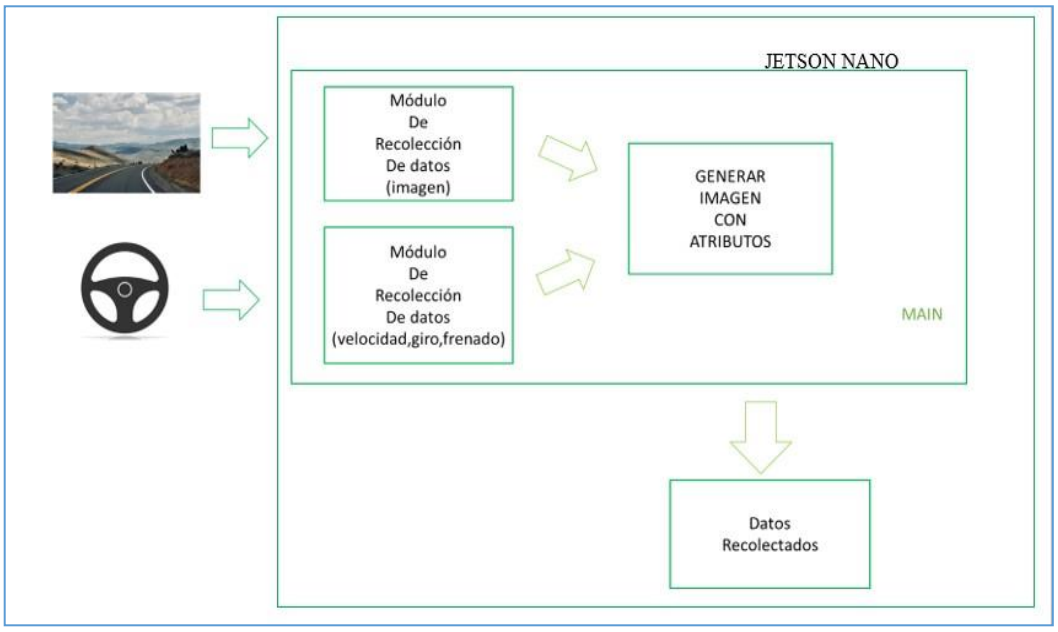
Este módulo con apoyo de los anteriores se encargó de capturar el valor analógico del joystick y de la cámara, los cuales se guardaron en un DataSet con el cual se entrenó la red neuronal convolucional. Además, fue capaz de generar diferentes sesiones de entrenamiento y así poder tener más información.

3.2.3. Adquisición de datos de entrenamiento para el sistema de conducción autónoma

Para tener una mejor visión del proceso de adquisición de información a continuación se muestra un diagrama de bloques del proceso de adquisición de los datos (ver la figura 42). Es necesario resaltar que el proceso de entrenamiento fue realizado sin considerar los elementos: señales de tráfico, peatones, etc.

Figura 42

Adquisición de datos.



Fuente Propia

Para la obtención del DataSet de entrenamiento de forma variada, así como también lo suficientemente grande para poder tener resultados eficaces, se realizaron 10 vueltas en un sentido (el que indica la flecha verde de la figura 40), y luego otras 10 vueltas en el sentido contrario con cada uno de los lentes de prueba.

Una vez obtenida esta información nos quedaría un archivo de tipo .csv y una carpeta con las imágenes del entrenamiento exportamos dicha información hacia una Workstation donde procesaremos la información obtenida, gracias a la librería pandas con la cual primero hacemos un dataframe en el cual colocaremos el nombre de la imagen como también el valor de giro obtenido en ese instante, en la figura 43 se enseñará el archivo .csv generado por la fase de entrenamiento.

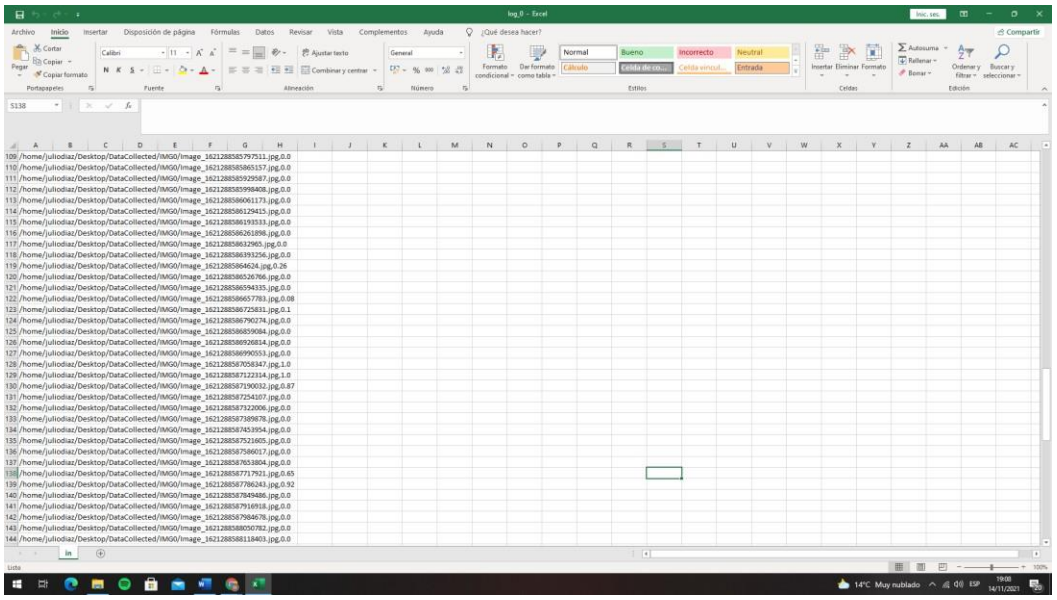
3.2.4. Uso de librerías en Python para balanceo de la data obtenida

Con el dataset obtenido se procedió a cargar esta información con la librería Pandas. Para ello, primero se tuvo que realizar un código de programación para seleccionar las sesiones que se desean tomar para el entrenamiento y al mismo tiempo visualizar la data total que se tiene entre las sesiones obtenidas. Posteriormente, se procedió a realizar un histograma en la cual se hizo una visualización de los comandos de giro y la cantidad de estos en toda la data obtenida. En la figura 44 se muestra el histograma obtenido con la

lente de baja distorsión, y en la figura 45 el histograma con la lente angular.

Figura 43

Dataframe obtenido.



Fuente Propia

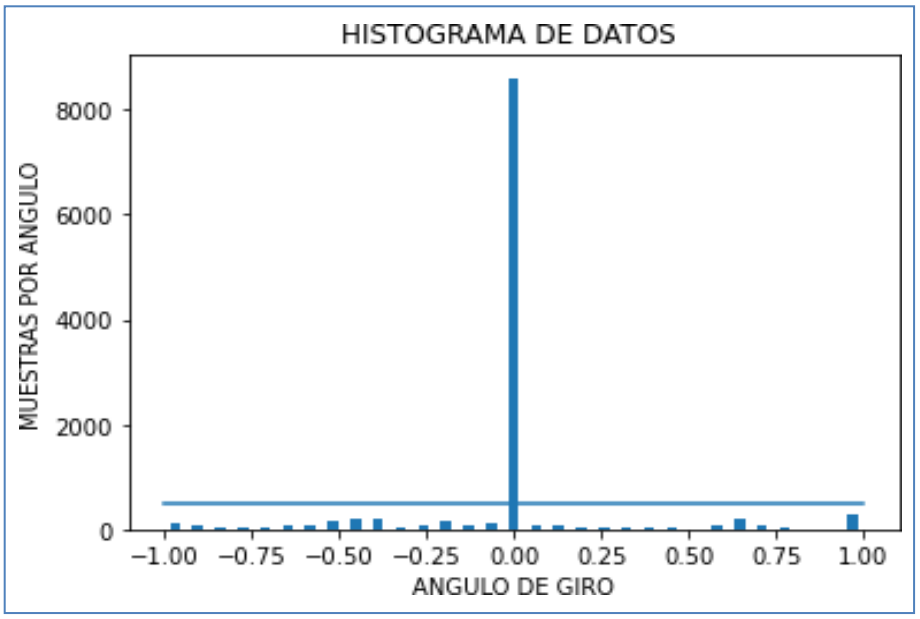
De las figuras 44 y 45, a simple vista, se observa la dominancia de los datos en 0° (sin ángulo de giro), esto se debe a que el circuito de pruebas dispone de solo 4 esquinas de giro por lo cual se cuenta con un DataSet desbalanceado lo que generó un valor de inferencia bueno en la clase mayoritaria (ángulo de 0° grados), y un bajo recall de los demás casos. Para solucionar este problema se ejecutaron varias opciones que a continuación se describen.

- Subsampling de datos con giro de 0° grados

Consistió en eliminar de manera aleatoria los datos de la clase mayoritaria para así poder tener un DataSet más balanceado. Esta opción puede ser perjuiciosa dado que se eliminan datos importantes para el entrenamiento, pero dada la topología del circuito de pruebas podemos apreciar que donde se realiza este caso las imágenes no tienen tanta varianza; por lo tanto, es posible ejecutar este procedimiento.

Figura 44

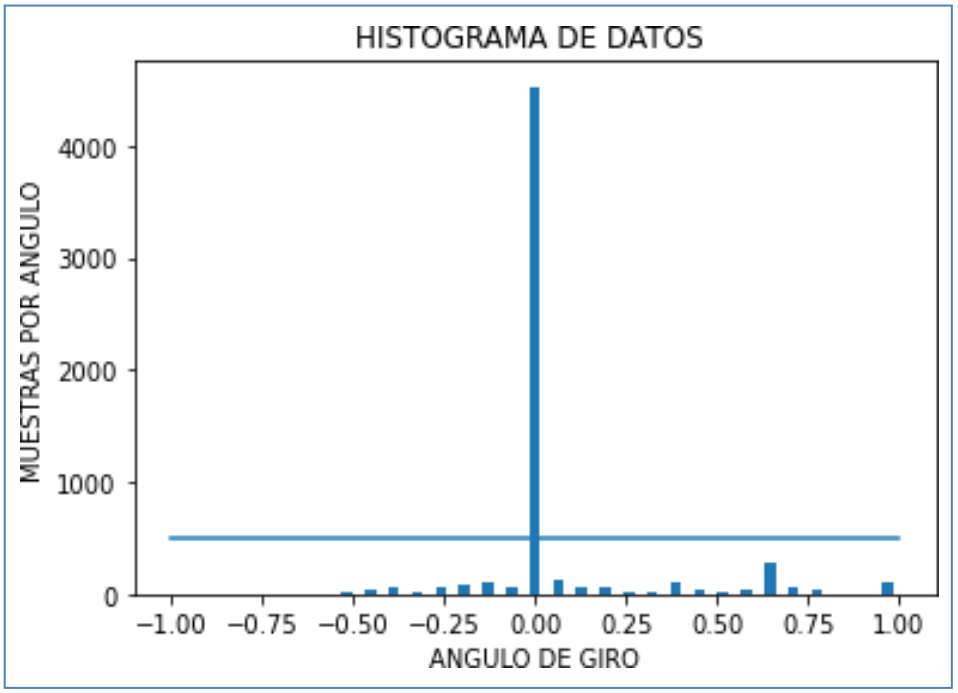
Data Obtenida Desbalanceada con lente de baja distorsión.



Fuente Propia

Figura 45

Data Obtenida Desbalanceada con lente angular.



Fuente Propia

- Oversampling de datos diferentes a 0° grados.

Consistió en aumentar la data minoritaria hasta que iguale a la data mayoritaria. En el caso de este trabajo, no fue tan factible ya que, al contar con mucha información de una clase, se logró también un aumento del dataset y por lo cual hizo más lento el proceso de entrenamiento.

Entonces, habiendo indagado de cada uno de los procedimientos de balanceo se optó por usar el subsampling de la clase mayoritaria, por lo cual se generó un algoritmo el cual colocó un límite de muestras por cada comando de giro producido. Para ello, se implementó una función en el lenguaje de programación Python que tiene como entrada la data suministrada por los entrenamientos (ver la figura 46); luego, de manera aleatoria se seleccionaron solo 500 muestras de todas las clases, y se redujo notoriamente la clase dominante lo cual produjo una distribución de datos más balanceada.

Figura 46

Función de Balanceo en Python.

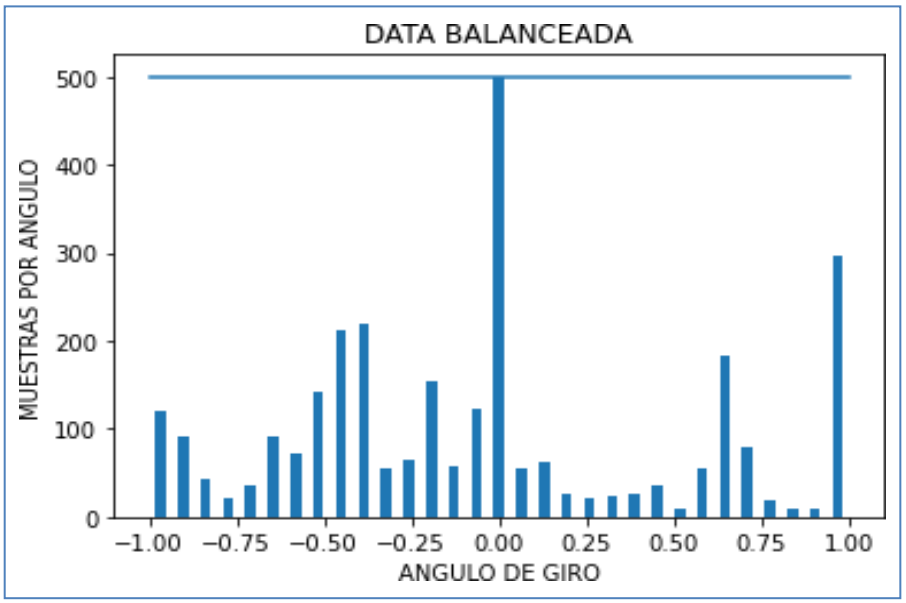
```
def balanceData(data,display=True):
    nBin = 31
    samplesPerBin =  500
    hist, bins = np.histogram(data['Steering'], nBin)
    if display:
        center = (bins[:-1] + bins[1:]) * 0.5
        plt.bar(center, hist, width=0.03)
        plt.plot((np.min(data['Steering']), np.max(data['Steering'])), (samplesPerBin, samplesPerBin))
        plt.title('HISTOGRAMA DE DATOS ')
        plt.xlabel('ANGULO DE GIRO')
        plt.ylabel('MUESTRAS POR ANGULO')
        plt.show()
    removeindexList = []
    for j in range(nBin):
        binDataList = []
        for i in range(len(data['Steering'])):
            if data['Steering'][i] >= bins[j] and data['Steering'][i] <= bins[j + 1]:
                binDataList.append(i)
        binDataList = shuffle(binDataList)
        binDataList = binDataList[samplesPerBin:]
        removeindexList.extend(binDataList)

    print('IMAGENES ELIMINADAS:', len(removeindexList))
    data.drop(data.index[removeindexList], inplace=True)
    print('IMAGENES RESTANTES:', len(data))
    if display:
        hist, _ = np.histogram(data['Steering'], (nBin))
        plt.bar(center, hist, width=0.03)
        plt.plot((np.min(data['Steering']), np.max(data['Steering'])), (samplesPerBin, samplesPerBin))
        plt.title('DATA BALANCEADA')
        plt.xlabel('ANGULO DE GIRO')
        plt.ylabel('MUESTRAS POR ANGULO')
        plt.show()
    return data
```

Fuente propia

Una vez eliminada la data excedente procedemos a guardar esta nueva información en una nueva etiqueta la cual contendrá toda la nueva información lista para ser procesada; a continuación, en las figuras 47 y 48 se muestran los histogramas del conjunto de datos balanceados de los dos tipos de lente.

Figura 47

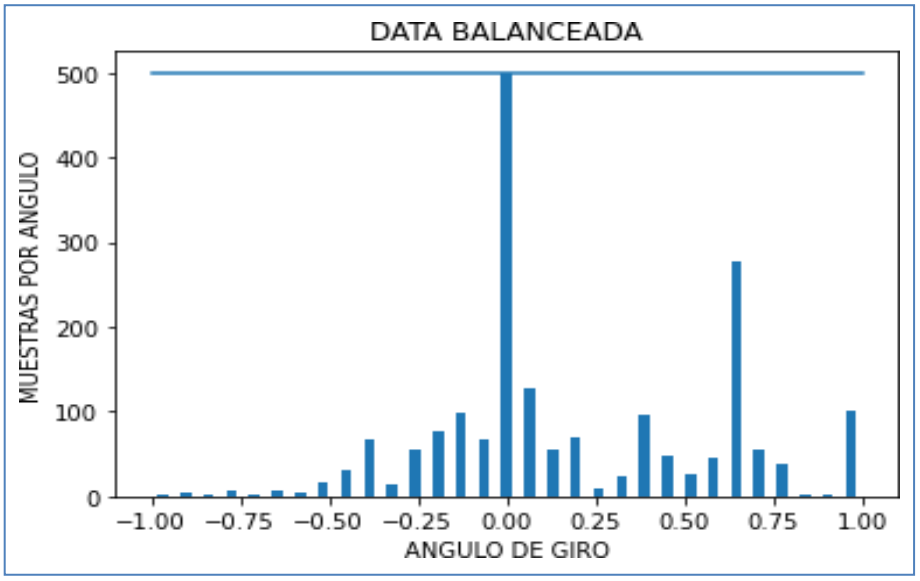
Data Obtenida Balanceada lente baja detorsión.



Fuente propia

Figura 48

Data Obtenida Balanceada lente angular.



Fuente propia

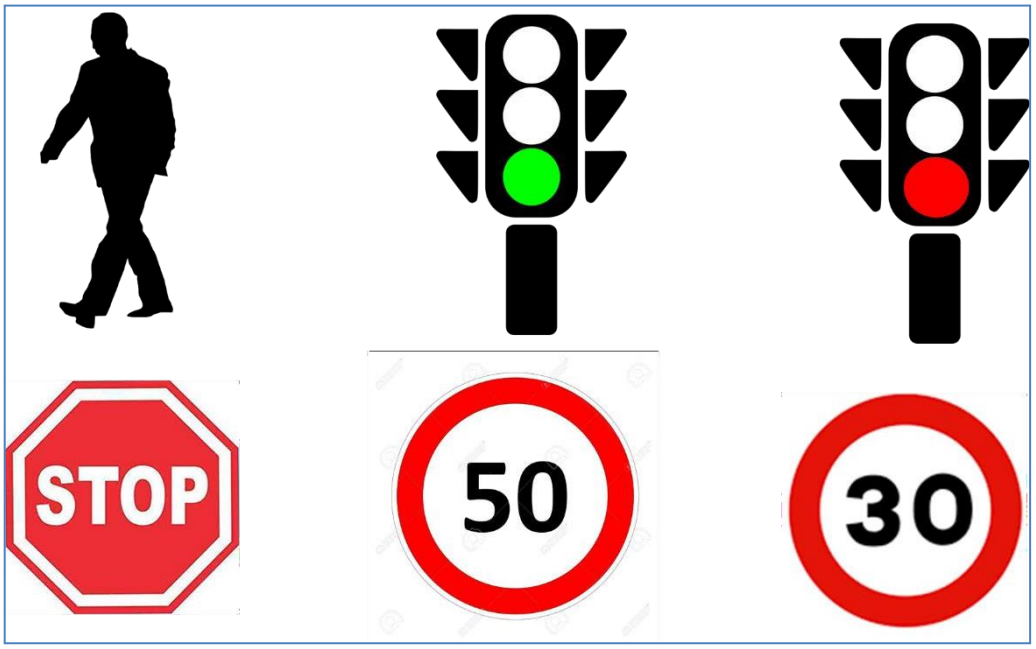
3.3. Desarrollo del algoritmo de visión artificial basado en Haar Cascade para detección de elementos en el circuito

3.3.1. Captura de datos para el entrenamiento del algoritmo de visión artificial

Dado que se trabajó con la técnica de Haar Cascade, lo primero que se realizó fue la implementación de un archivo “xml” para cada clase detectada. En este caso son 6: Luz roja, luz verde, señal de stop, límites de velocidad y peatones; por ello, se desarrolló un código de programación en Python para la Jetson Nano de tal forma que permitarecolectar imágenes con una dimensión de 40 pixeles de alto y 40 de ancho, haciendo untotal de 260 imágenes por clase; en la figura 49 se observa cada una de las clases mencionadas.

Figura 49

Clases a Detectar.



Fuente propia

Además de las imágenes de cada una de las clases, se tuvo que recolectar imágenes donde no aparezcan tales clases. Estas son llamadas negativas, y para eso se realizó la captura con la misma dimensión que las del grupo de positivas cuando se hizo el recorrido del automóvil en el circuito, pero sin ninguna de las clases presentes. A continuación, las figuras 50 y 51 muestran algunos ejemplos obtenidos con los dos tipos de lentes utilizados

Figura 50

Circuito vacío: imagen tomada con lente de baja distorsión.

Fuente propia

Figura 51

Circuito vacío: imagen tomada con lente angular.

Fuente propia

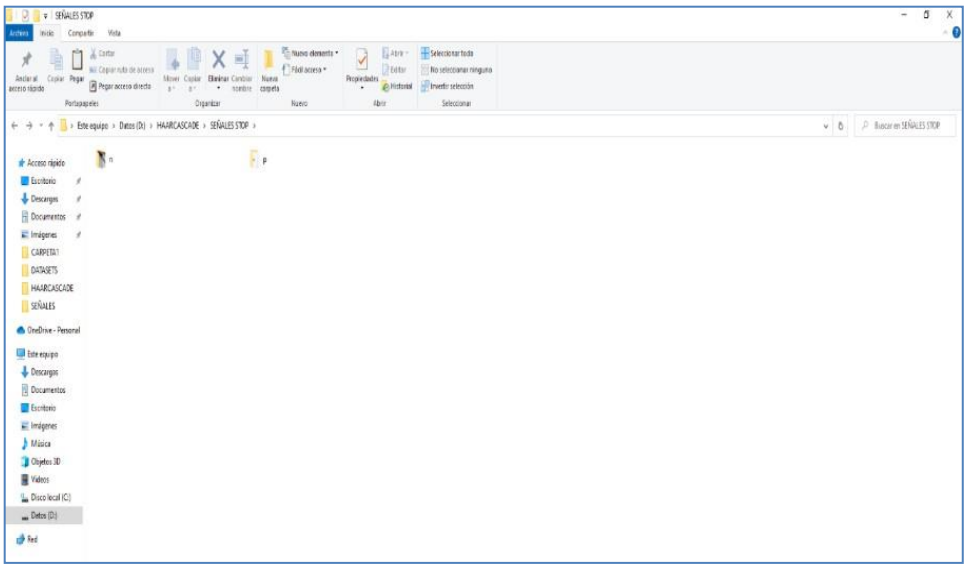
3.3.2. Entrenamiento del contenido de los archivos XML

Para el entrenamiento de los contenidos de cada archivo “xml”, primero se procedió a separar las clases en carpetas diferentes. Luego, las imágenes positivas (aquellas que pertenecen a las clases por clasificar) se almacenaron en una carpeta denominada “p”. Por otro lado, las imágenes negativas fueron almacenadas en una carpeta denominada “n”.

Seguidamente, la figura 52 muestra tal representación.

Figura 52

Separación de Información.

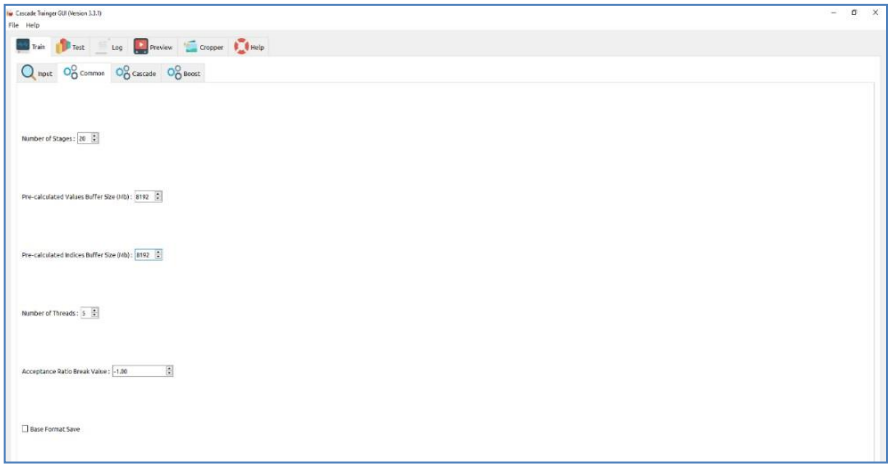


Fuente propia

Posterior a esto y con la ayuda del programa Cascade Trainer Gui, se realizó el entrenamiento de cada uno de los clasificadores. Para lo cual, primero se ingresó al programa, luego se seleccionó la carpeta donde se encuentra la información para el entrenamiento, y posteriormente se dejaron las demás opciones como predeterminadas en esa pestaña. Ver las figuras 53,54,55.

Figura 53

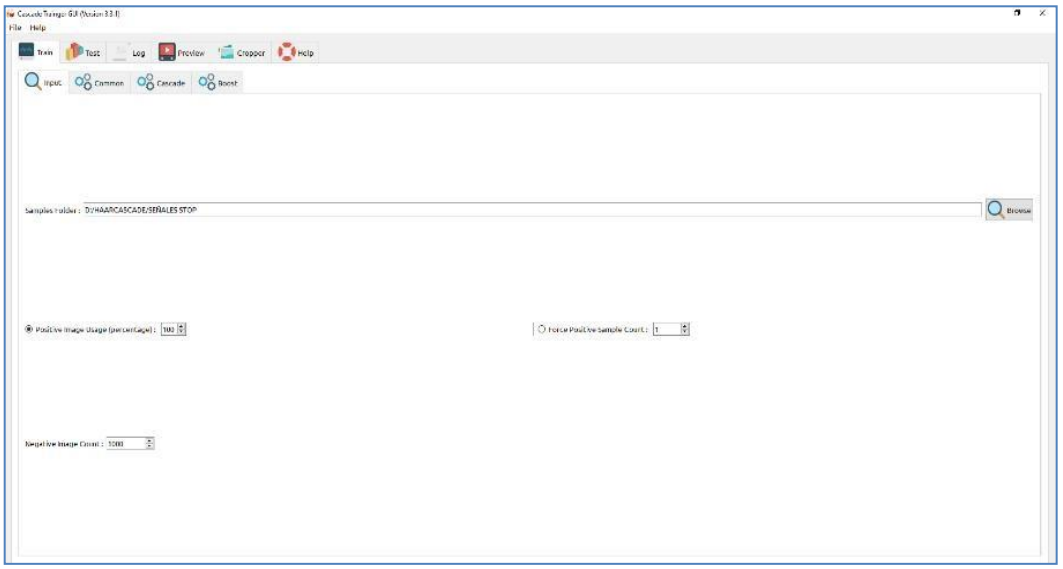
Selección de Información.



Fuente propia

Figura 54

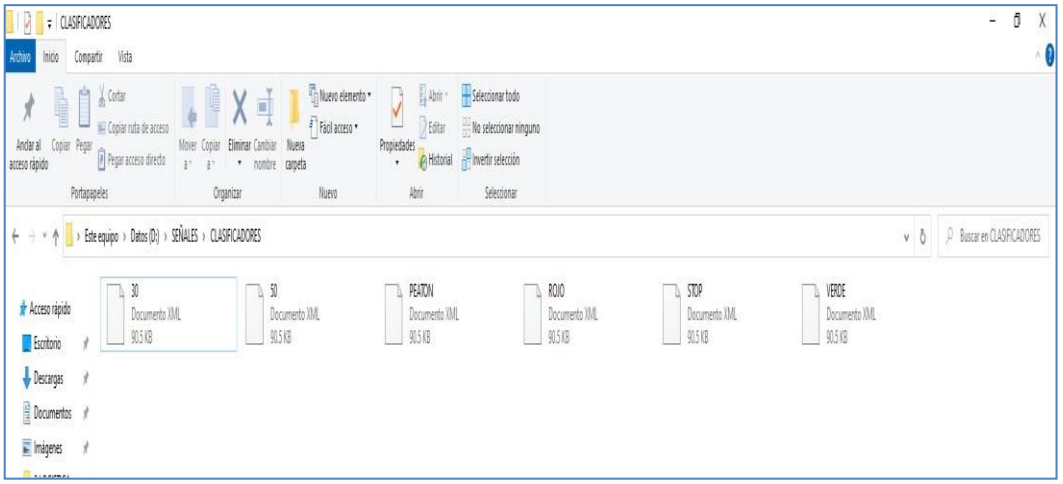
Configuración de parámetros en Common.



Fuente propia

Figura 55

Archivos .xml.



Fuente propia

Luego, en la siguiente etapa se seleccionó la dimensión de las imágenes a 40*40 pixeles. Una vez seleccionado este parámetro se continuó con el entrenamiento del clasificador, el cual generó el archivo denominado “cascade.xml” que corresponde al clasificador de cada clase. Este proceso se repitió en todas las clases existentes de objetos que se deseaban detectar. Una vez logrado el entrenamiento de todos los clasificadores, se prosiguió a exportarlos hacia la tarjeta JETSON-NANO donde fueron evaluados y ejecutados mediante un módulo de reconocimiento de objetos. Posteriormente, haciendo

uso de la librería de aceleración CUDA integrada en OpenCV se obtuvieron mejores resultados a la hora de leer y ejecutar los clasificadores de manera conjunta. Para ello, se tuvo que cargar cada uno de los clasificadores con una variable diferente; luego, se procedió con la transformación de la imagen a una escala de grises, seguido de la realización de la detección de los objetos y finalizando con la devolución del objeto detectado.

3.4. Optimización con (KERAS-TUNER)

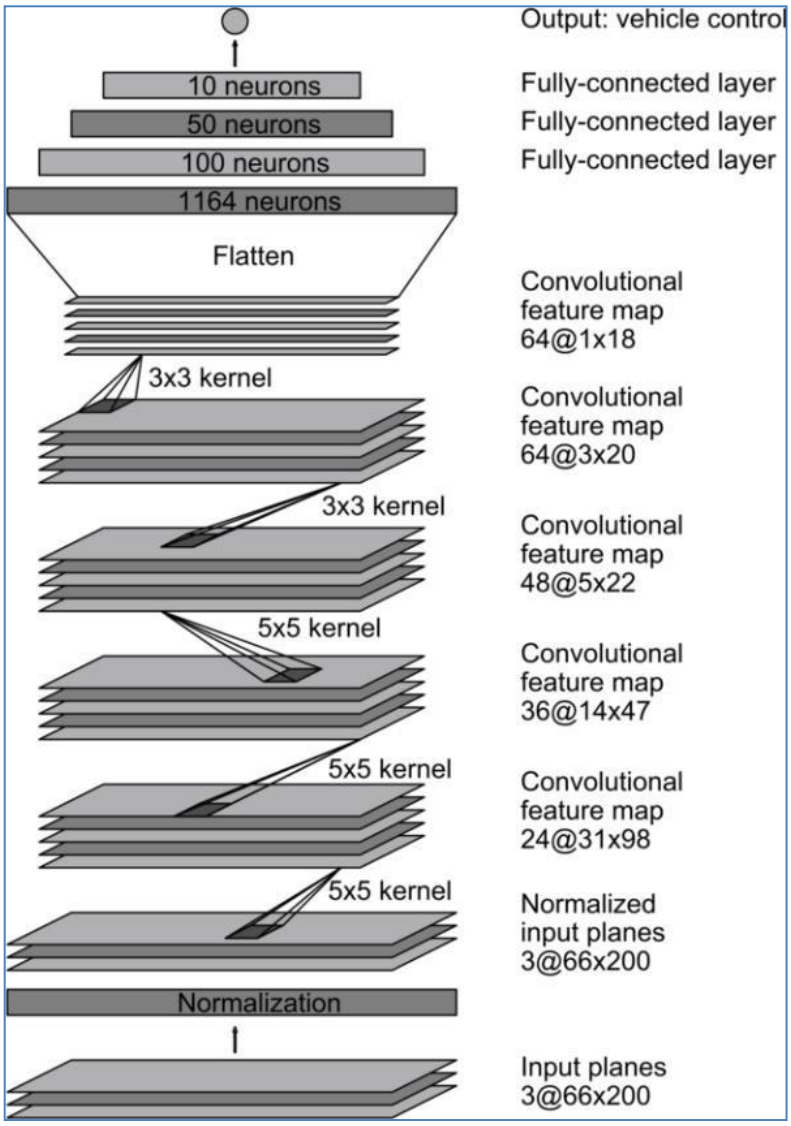
El proceso de optimización se llevó a cabo utilizando el DATASET optimizado, de tal forma que al utilizar el TENSORFLOW GPU se logró el entrenamiento, y a su vez la ejecución del framework TensorRT para mejorar el performance.

3.4.1. Optimización de Red Neuronal Convolucional con Keras Tuner

Como todo proyecto que involucra el uso de las redes neuronales convolucionales, fue preferible utilizar una red ya diseñada. Por ejemplo, si se desea detectar rostros utilizamos la arquitectura FACENET, si se desea detectar objetos, animales, personas, etc. se utilizan las arquitecturas ALEXNET, RESNET, GOOGLENET y diferentes redes convolucionales. Por lo tanto, para este proyecto se decidió por usar la red DAVE-2 creada por la empresa NVIDIA y muy usada en sus proyectos. A continuación, en la figura 56, se muestra la arquitectura y las principales capas de la red neuronal convolucional DAVE-2.

Figura 56

Arquitectura de red DAVE 2.



Fuente https://developer.nvidia.com/blog/deep-learning-self-driving-cars/

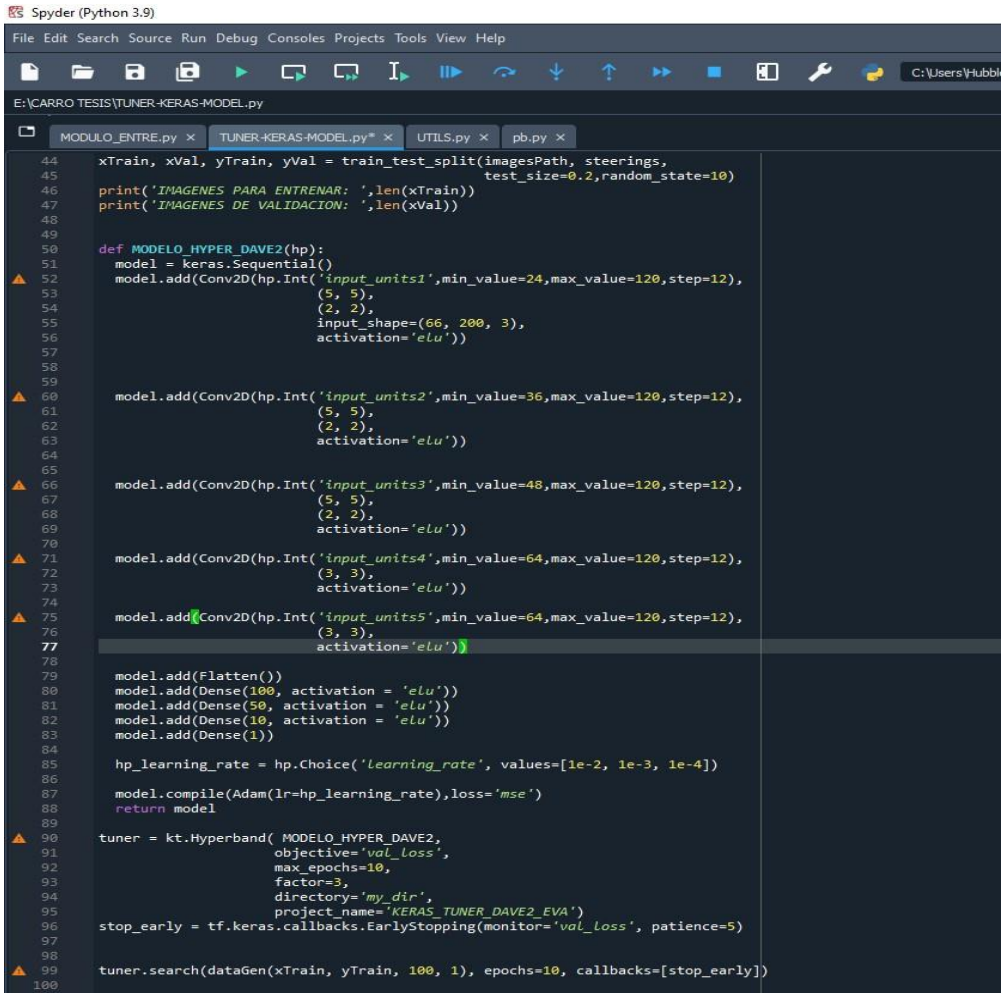
Asimismo, de la figura 56 se observa la presencia de una capa inicial denominada input que tolera imágenes con resolución de 66*200 pixeles. Luego, tales imágenes ingresan a una etapa de normalización, y posteriormente pasan por 5 capas de convolución constituidas por un total de 24 filtros de dimensiones de 5*5, un maxpollingde 2*2, y una función de activación RELU. Posterior a las capas de convolución se continuó con el proceso de flatten el cual sirve como intermediario entre el proceso de convolución y las capas de neuronas. Asimismo, es necesario resaltar que esta red es deltipo Regresión y no de Clasificación; por cual, el resultado será un valor numérico que corresponde al ángulo de giro.

Por lo tanto, surge la siguiente interrogante: ¿Existe alguna manera en la cual se puede mejorar el performance de la red alimentada por el DATASET de 2 tipos de lentes, así como mejorar los resultados obtenidos de la red base?; la respuesta está en el uso de la librería KERAS-TUNER y la manipulación de los hiper-parámetros. Para ello, primero se tuvo que seleccionar que hiper-parámetros se iban a modificar, y estos fueron la cantidad de filtros por capa de convolución y el learning rate en la parte del entrenamiento de la red convolucional. Es así que, para lograr esta mejora se definió un nuevo modelo para un espacio de búsqueda entre un rango de valores; de esta manera, fueron seteados dentro de cada capa con un campo establecido que se denominó "input_units"; entonces, por cada capa de convolución se procedió a la inicialización con valores de 24 hasta 120 números de filtros con un salto de 12 unidades como en la primera capa, y en el caso del learning rate se seleccionaron 3 opciones diferentes [1e-2, 1e-3, 1e-4]. En la figura 56 se muestra el espacio de búsqueda del cual se ha comentado.

Una vez realizado este procedimiento, se continuó con la definición de un sintonizador de hiper-parámetros el cual estuvo conformado de 4 sintonizadores comentados anteriormente; asimismo, se encargaron de seleccionar valores que fueron colocados en los espacios definidos del modelo de búsqueda, además se seleccionó el objetivo a lograr y se eligió la métrica "val_loss" para este sintonizador para luego dar inicio con la sintonización de los hiper-parámetros, y así alcanzar una mejora de las métricas del modelo. Luego de terminar la búsqueda, se invocó el modelo con mejor performance y fue etiquetado para el entrenamiento del modelo optimizado, el cual contó con valores diferentes a los anteriores. Y, tal como también se observa en la figura 57, la cantidad de filtros seleccionados por cada capa fue distinta al modelo original, así como también lo fue el learning rate que es un valor distinto al primero.

Figura 57

Espacio de Búsqueda.



Fuente Propia

3.4.2. Entrenamiento de la Red Neuronal usando Tensorflow GPU

El proceso de entrenamiento de la red neuronal se llevó a cabo usando una unidad de procesamiento gráfico (GPU) para las tareas que más se requirieron. Para ello, fue necesario configurar las dependencias con la finalidad de hacer uso de esta. Es así como, primero se instalaron las dependencias de CUDA TOOLKIT en su versión 11.2 y CUDNN en la versión 8.2. Después de la instalación, se continuó con el entrenamiento de la red neuronal optimizada utilizando un total de 11024 imágenes tomadas con el lente de baja distorsión, y 11029 imágenes capturadas con el lente angular. Estos dos grupos de imágenes fueron divididas a su vez en 2 grupos por tipo de lente: entrenamiento y validación, pero con una repartición de 70% para el primero y 30% para el segundo grupo.

Asimismo, para mejorar los resultados del entrenamiento con las imágenes

obtenidas se tuvo que realizar un procesamiento de estas utilizando la librería OPEN-CV, que representa una imagen como un arreglo multidimensional pero invertido. Por lo cual, al colocar una imagen en el modelo de color RGB se tuvo como salida en un formato BGR; dado esto, se optó por utilizar la librería MATPLOTLIB que representa a la imagen mediante un arreglo no invertido (RGB). De esta manera, teniendo esta indicación y con el uso de la librería IMGAUG, fue posible realizar el procesamiento de cualquier imagen en el modelo de color RGB, y así ayudar a la red neuronal en su procesamiento. Por lo tanto, las acciones de mejoras que fueron realizadas de manera aleatoria por imagen y que representan una probabilidad del 50% de ser ejecutada, se enumeran a continuación:

- Ajuste de brillo

Para poder aumentar la variedad de iluminación en cada imagen se procedió a realizar un aumento y/o disminución de la cantidad de brillo en la imagen de entrada.

- Zoom

La imagen obtenida posee elementos los cuales no son necesarios para realizar la conducción autónoma del vehículo en el carril determinado, por lo cual solo fue necesaria la información proveída por el área de interés de la imagen.

- Giro de imagen

Al dar un sentido opuesto a la imagen se aumentó la variedad de datos, logrando así la obtención de mejores resultados teniendo en cuenta que el ángulo de giro también se cambia al opuesto.

De esta manera, el proceso de entrenamiento duró 102 minutos para las imágenes con lente de baja distorsión; mientras que para las de lente angular fueron 105 minutos. Asimismo, los entrenamientos fueron ejecutados con un total de 40 épocas de entrenamiento y 300 pasos por cada una de ellas; así como también, se contó con una semilla idéntica para los 2 casos. Luego del entrenamiento, se procedió a guardar el modelo en un archivo “. pb” para su futuro procesamiento.

3.4.3. Optimización de la red neuronal usando Tensor-RT

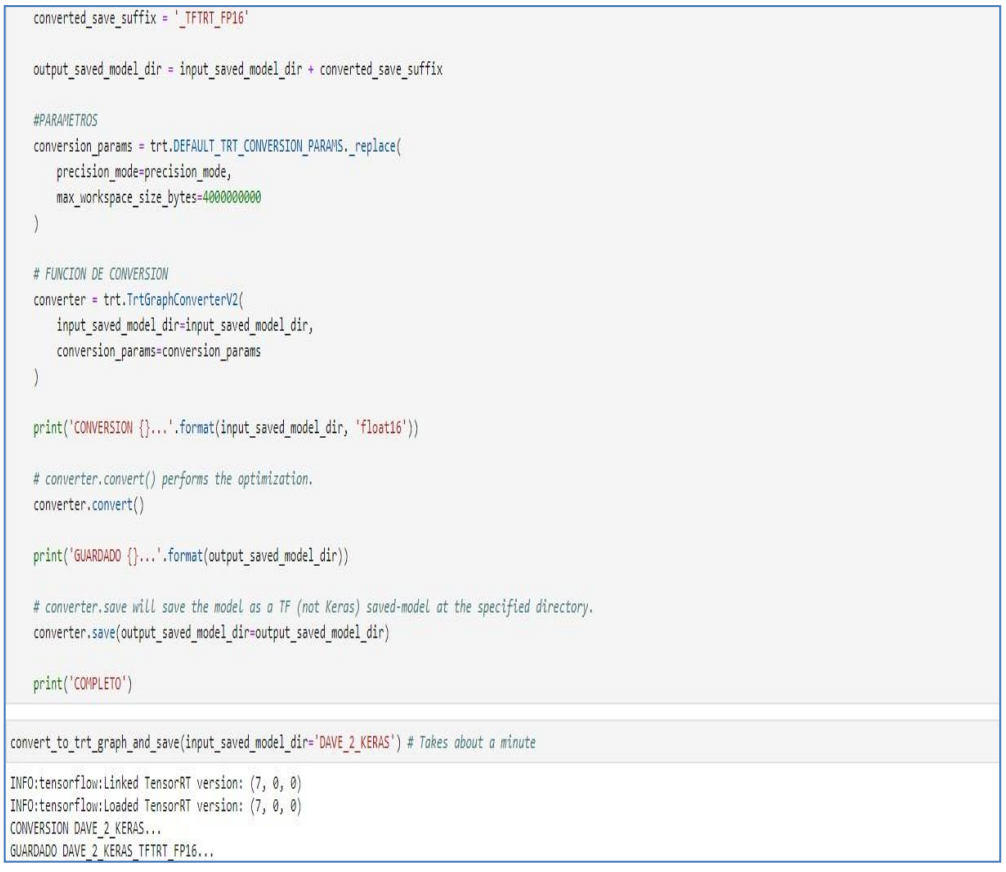
Para mejorar aún más los resultados obtenidos se realizó una optimización con la ayuda de Tensor-RT, el cual mejoró la inferencia de los modelos y optimizó el uso de recursos del sistema. Además, son compatibles con el ecosistema Nvidia en el cual se ha desarrollado este trabajo. De esta manera, como primer paso se procedió a guardar la red neuronal en formato TF, dado que su versión optimizada por Keras Tuner se encuentra en formato “.h5” y requiere ser convertida a un formato “.pb” para que sea compatible

con la optimización de la red neuronal. Luego de contar con el modelo convertido, se pasó a realizar la partición de la red en un operador, el cual unió todos los elementos quesean compatibles con Tensor-RT tales como capas de convolución, funciones de activación, normalización, etc. Todo ello fue realizado dentro de un operador al cual se le denominó TRTEngineOp y fue procesado por el optimizador, mientras que el resto fueejecutado de manera normal con Tensorflow. Luego, para la conversión se necesitó seleccionar los parámetros de esta, tal como el tipo de dato a utilizar que en este caso fuede FP16 para la optimización, así como también la capacidad máxima de memoria de la placa la cual fue 4 Gb. Luego de ello, se procedió a almacenar el modelo optimizado. A continuación, en la figura 58 se muestra el script utilizado para la conversión del modeloa FP16.

Luego de la conversión de la red neuronal, esta fue exportada a la tarjeta Jetson Nano para ser evaluada en el circuito de pruebas. Seguidamente, en la figura 59 se muestra el proceso por etapas para convertir un modelo Keras a Tensor RT.

Figura 58

Optimización con Tensor-RT.



```
    converted_save_suffix = '_TFTRT_FP16'

    output_saved_model_dir = input_saved_model_dir + converted_save_suffix

    #PARAMETROS
    conversion_params = trt.DEFAULT_TRT_CONVERSION_PARAMS._replace(
        precision_mode=precision_mode,
        max_workspace_size_bytes=4000000000
    )

    # FUNCION DE CONVERSION
    converter = trt.TrtGraphConverterV2(
        input_saved_model_dir=input_saved_model_dir,
        conversion_params=conversion_params
    )

    print('CONVERSION {}...'.format(input_saved_model_dir, 'float16'))

    # converter.convert() performs the optimization.
    converter.convert()

    print('GUARDADO {}...'.format(output_saved_model_dir))

    # converter.save will save the model as a TF (not Keras) saved-model at the specified directory.
    converter.save(output_saved_model_dir=output_saved_model_dir)

    print('COMPLETO')
```

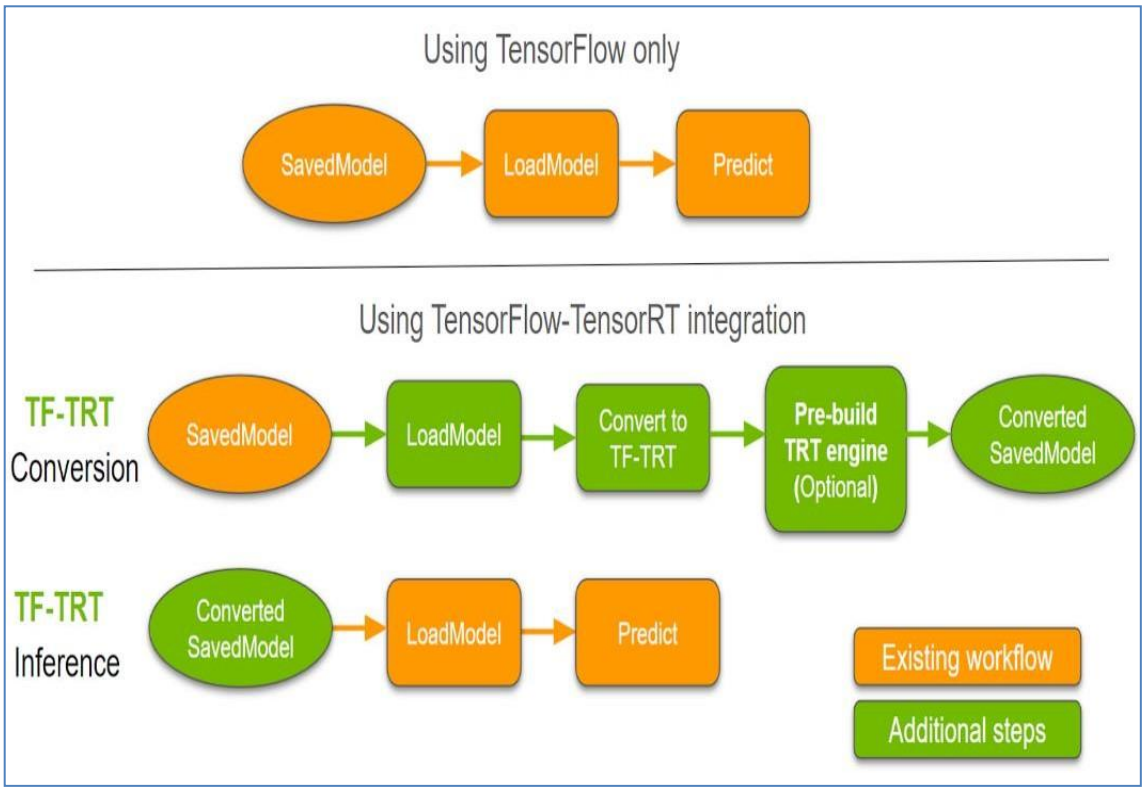
```
convert_to_trt_graph_and_save(input_saved_model_dir='DAVE_2_KERAS') # Takes about a minute
```

```
INFO:tensorflow:Linked TensorRT version: (7, 0, 0)
INFO:tensorflow:Loaded TensorRT version: (7, 0, 0)
CONVERSION DAVE_2_KERAS...
GUARDADO DAVE_2_KERAS_TFTRT_FP16...
```

Fuente propia

Figura 59

Proceso de Conversión de TF A TRT.



Fuente Tensorflow

CAPÍTULO IV: PRUEBAS Y RESULTADOS

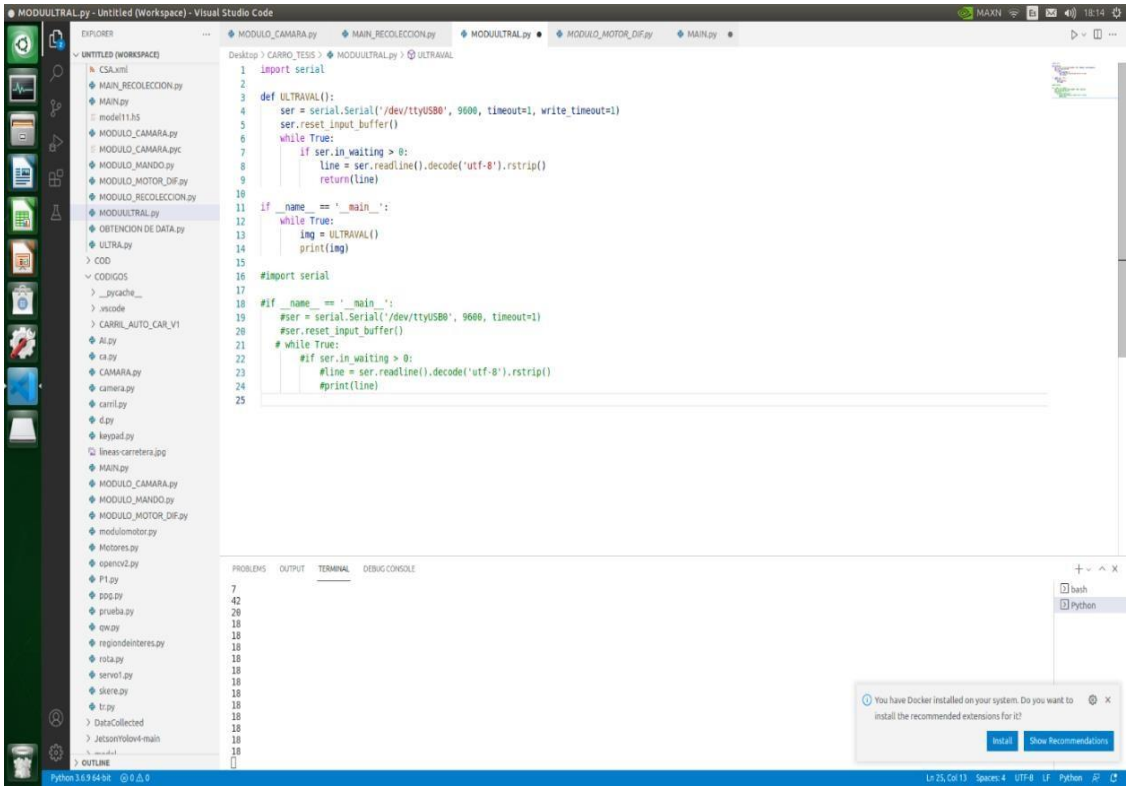
En este capítulo se muestran los resultados de las pruebas realizadas con lo desarrollado en el capítulo anterior, comenzando con las pruebas de comunicación serial entre los 2 dispositivos que se utilizaron para luego comprobar si la información del sensor ultrasonido fue enviada correctamente para detectar objetos frente al sensor; luego, se realizó una prueba del módulo de detección de señales, peatones, semáforos con los 2 tipos de lentes propuestos y usados en este proyecto. Una vez realizadas estas pruebas, se continuó con la obtención de los resultados logrados por KERAS-TUNER donde se comparó el performance del modelo original, así como también de la versión optimizada y un caso no ideal obtenido todo por la sintonización de hiper-parámetros. Después de realizar la optimización, se muestra como TENSOR-RT mejora el tiempo de inferencia y como también la optimización de recursos de la placa de desarrollo; y, para finalizar, se hacen pruebas sintéticas del modelo a través de comparaciones del modelo entrenado con la data de validación, así como las verificaciones en el circuito real de pruebas.

4.1. Prueba de Sensor Ultrasonido con Peatones

Para esta primera prueba se realizaron ensayos de proximidad con la clase de peatones. Para ello, se colocó el vehículo en modo de aceleración, así como también un peatón el cual está representado como una silueta de una persona impresa en cartulina que fue colocada al frente del recorrido. De esta manera, cuando el peatón fue detectadoa una distancia de 10 cm del vehículo, se procedió a la acción de frenado automático logrando así el detenimiento total del vehículo. En las figuras 60 y 61 se muestran la perspectiva del código fuente implementado en la tarjeta Jetson Nano cuando se detecta a un peatón a 10 cm al frente del sensor, así como también una fotografía de la perspectivadel entorno real.

Figura 60

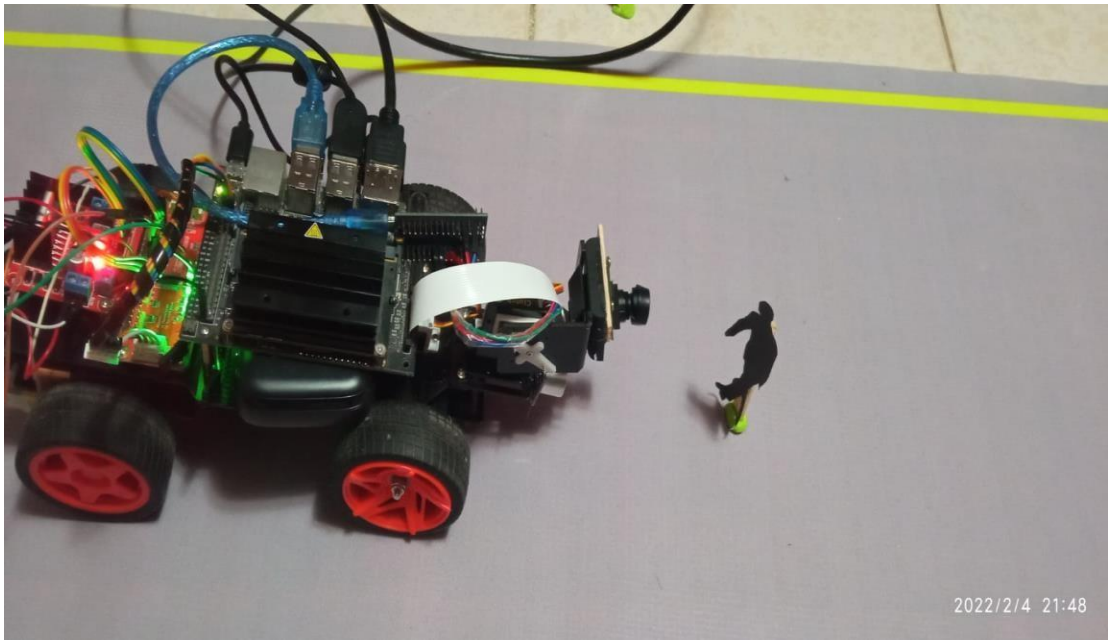
Detección de Peatones implementado en Jetson Nano.



Fuente Propia

Figura 61

Detección de Peatones en perspectiva del entorno real.



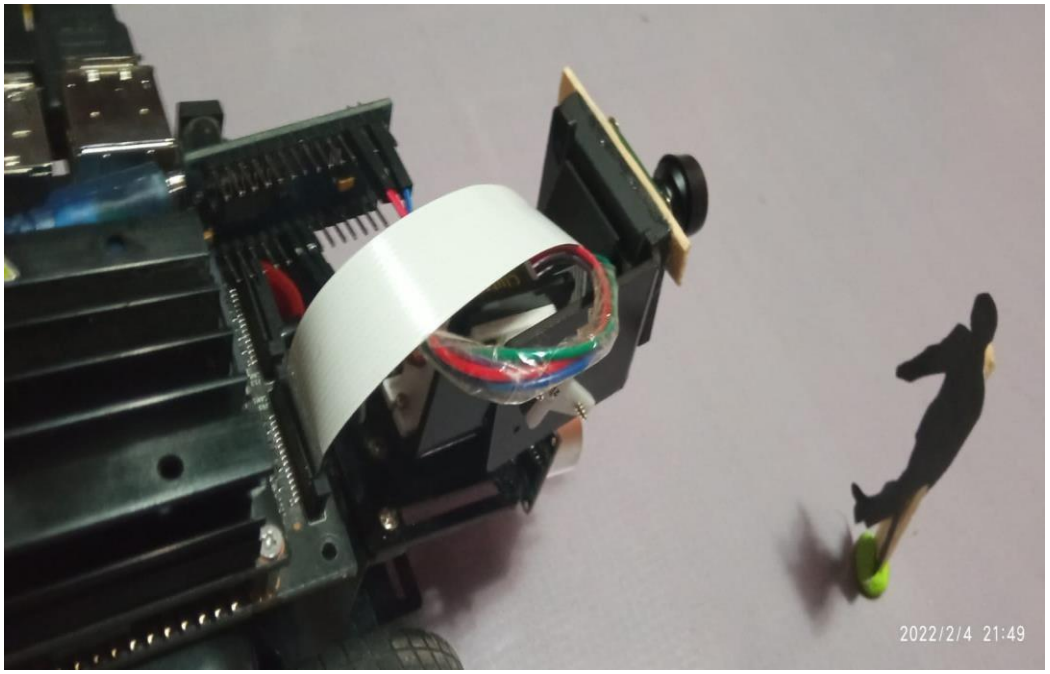
Fuente Propia

De lo obtenido anteriormente, se observó que la comunicación serial entre los dos dispositivos funcionó de manera satisfactoria, debido a que se transmitió de forma idónea la información desde el ATMEGA 328p hacia la tarjeta Jetson Nano. Por lo cual, la detección de objetos fue correcta y a la vez fue posible integrar dicho módulo al proyecto, teniendo la habilidad de detectar objetos que se encuentran en el área sensitiva del sensor.

No obstante, cuando el objeto se encontraba en una esquina del sensor no fue posible detectarlo porque estaba fuera del área de detección; por lo cual, el sensor únicamente brindó una detección apropiada cuando el objeto se encontraba únicamente al frente del sensor. En la figura 62 se comparte un ejemplo donde no es posible detectar al peatón con el sensor.

Figura 62

Detección Fallida de Peatón en perspectiva del entorno real.



Fuente Propia

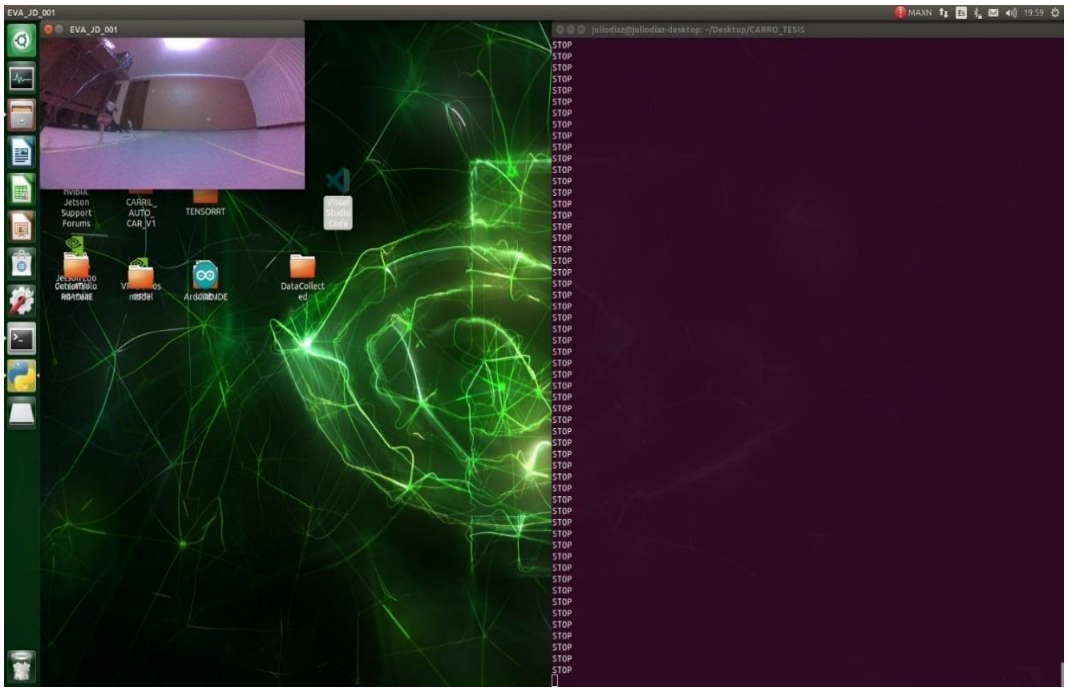
4.2. Detección de señales de tráfico

Una vez que se contó con todos los archivos “xml”, se determinó englobar todo en un único módulo detector de objetos; para ello, se hicieron las pruebas en el circuito de forma independiente por cada señal. A continuación, en las figuras 63 y 64 se muestran las detecciones de señal que se tuvieron en promedio, cuando el automóvil se encontraba

a 15-20 centímetros de las imágenes pertenecientes a algunas de las clases entrenadas.

Figura 63

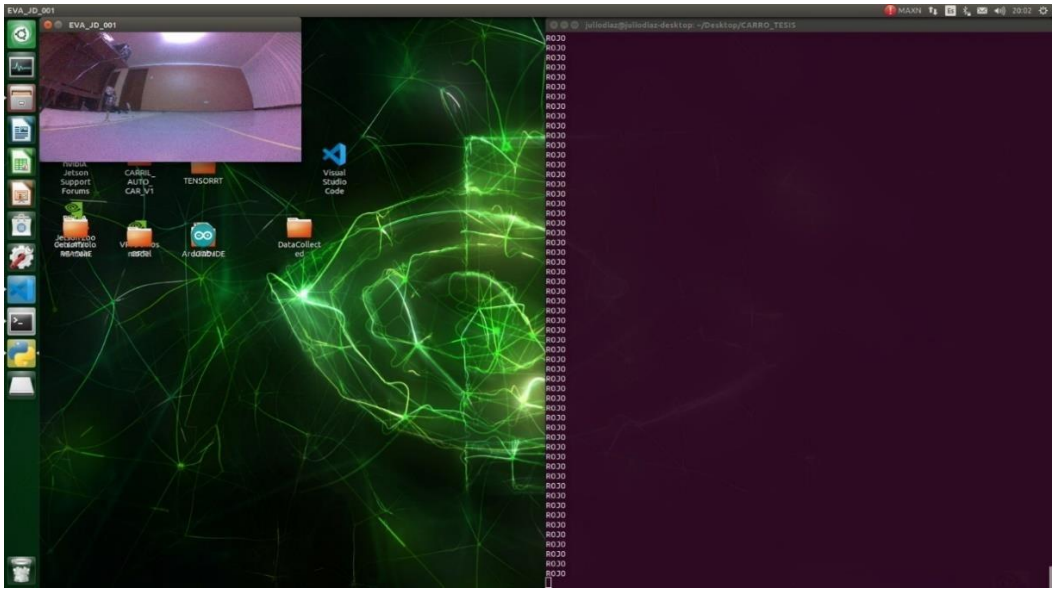
Detección de STOP con la lente angular.



Fuente Propia

Figura 64

Detección de Semáforo en Rojo con la lente angular.



Fuente Propia

Además de lo anterior, se realizaron otras pruebas tales como la de 6 detecciones por clase y por cada tipo de lente utilizado. De esta forma, en las tablas 7 y 8 se muestran tales resultados logrados.

Tabla 7

Resultados de Detección para distintas velocidades.

	No Detección	Detección	No Detección	Detección	No Detección	Detección
	STOP		Límite de Velocidad 30		Límite de Velocidad 50	
Lente Angular	0	6	1	5	0	6
Lente Baja Distorsión	1	5	2	4	1	5

Fuente Propia

Tabla 8

Resultados de Detección para distintos tipos de luces.

	No Detección	Detección	No Detección	Detección	No Detección	Detección
	Peatón		Luz roja		Luz verde	
Lente Angular	0	6	2	4	1	5
Lente Baja Distorsión	2	4	1	5	0	6

Fuente Propia

De esta manera, de los resultados obtenidos, la detección de las clases fue mejor cuando se utilizó una lente angular. Pues, esto permitió obtener en total, para todas las clases, un puntaje de 4 situaciones no detectadas; por lo contrario, la lente de baja distorsión dio un total de 8 situaciones no detectadas y se debió a su mayor campo de

visión, dado que en algunos casos los peatones se situaron fuera del carril de circulación; esto originó que la lente de baja distorsión no los encuentre en su rango de visión. Por lo cual, en el caso de una implementación con mayor número de clases y en un circuito de pruebas más controlado, se recomienda el uso de una lente angular para la realización de esta tarea la cual tiene un consumo de 1.1 gigas de memoria del sistema.

Con lo obtenido en todas las pruebas realizadas durante el proyecto se recomienda al menos el uso de 2 lentes, una para la navegación en el entorno (lente de baja distorsión) mientras que, para la captura y reconocimiento de señales de tráfico, semáforos, peatones haríamos uso de una lente angular, dada las limitaciones del sistema embebido JETSON NANO priorizamos la lente de baja distorsión dado que si se utilizan las 2 cámaras faltarían recursos para poder ejecutar el programa.

4.3. Optimización de Red Neuronal Convolucional usando Keras Tuner

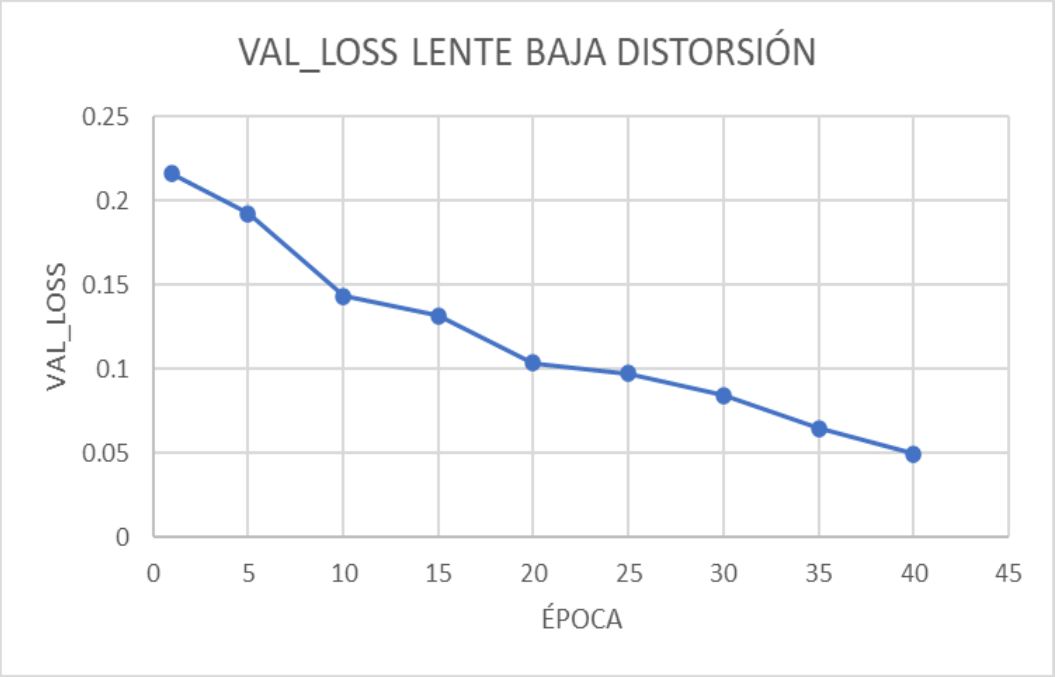
La realización de la búsqueda ideal de los hiper-parámetros demandó un tiempo aproximado de 16 horas, porque se probaron muchas combinaciones donde algunas dieron un mejor resultado de lo esperado y en otros casos los resultados fueron menores al original. Es importante resaltar que esta optimización se hizo con 2 tipos de imágenes obtenidas por los sensores (lente de baja distorsión y lente angular), con el motivo de contar con una base de datos más diversa, así como también tener un solo modelo que sirva para el entrenamiento con 2 tipos de imágenes diferentes. A continuación, en las figuras 65, 66, 67, 68, 69, 70, 71, 72, 73, 74 y 75, se muestran comparativas de los resultados obtenidos del entrenamiento con las métricas “val_loss”y “loss” en función de las 40 épocas realizadas para la red neuronal original, así como también para la red neuronal optimizada, y usando las imágenes del lente de baja distorsión y angular, y asimismo para un caso no ideal de optimización.

Es así como, las figuras 65 y 66, corresponden a la medición del comportamiento de la métrica de entrenamiento “val_loss” de los dos tipos de lentes (baja distorsión y angular), donde se observa el resultado alcanzado con el entrenamiento del modelo original de red neuronal; para el primer tipo se alcanzó un valor de 0.051 mientras que para el segundo un valor de 0.069, presentando una mayor pérdida con la lente de baja distorsión. Siguiendo con los resultados, a continuación, se muestran en las figuras 67 y 68, los resultados obtenidos con la métrica “loss"; donde, para el primer caso se alcanzó un valor de 0.05899 y para el segundo 0.07484.

Seguidamente, en las figuras 69 y 70, se muestran los resultados obtenidos para la métrica “val_loss” de los dos tipos de lentes (baja distorsión y angular), y logrados con la mejor sintonización de la red neuronal optimizada; es así como, se alcanzó un valor de 0.01211 para la lente de baja distorsión, mientras que para el lente angular se obtuvo un valor de 0.04900. Luego, se cambió de métrica a “loss" y se obtuvieron los resultados que se muestran en las figuras 71 y 72, en las cuales se observa que la lente de baja distorsión marca una pérdida de 0.02611, mientras que la angular una pérdida de 0.05; de esta manera.

Figura 65

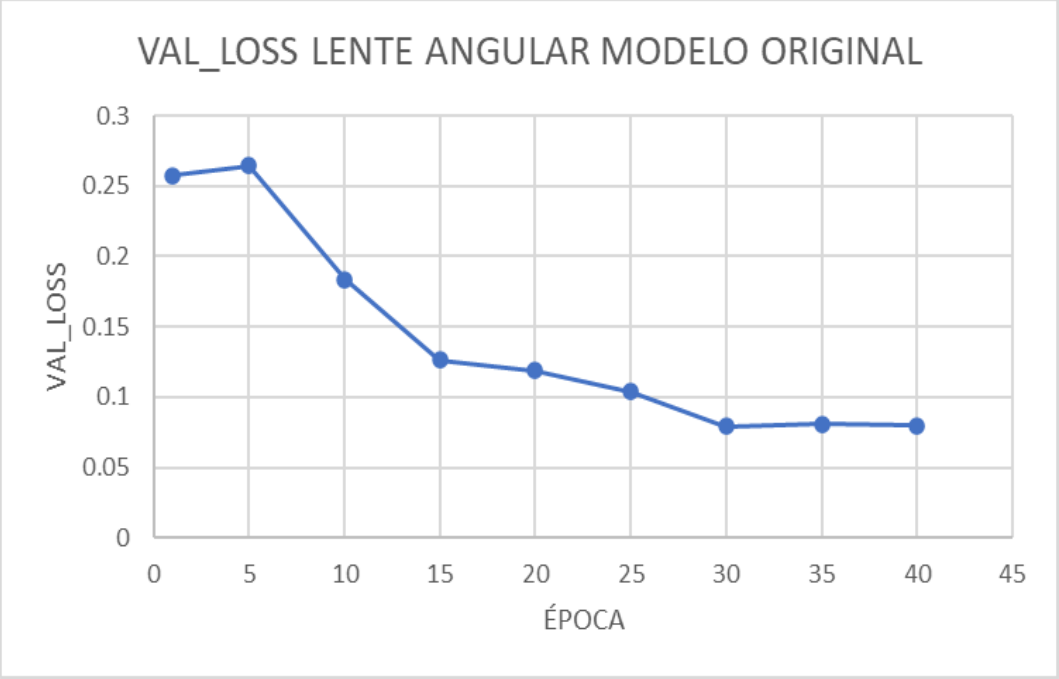
Métrica de VAL_LOSS para lente de baja distorsión con el modelo original.



Fuente Propia

Figura 66

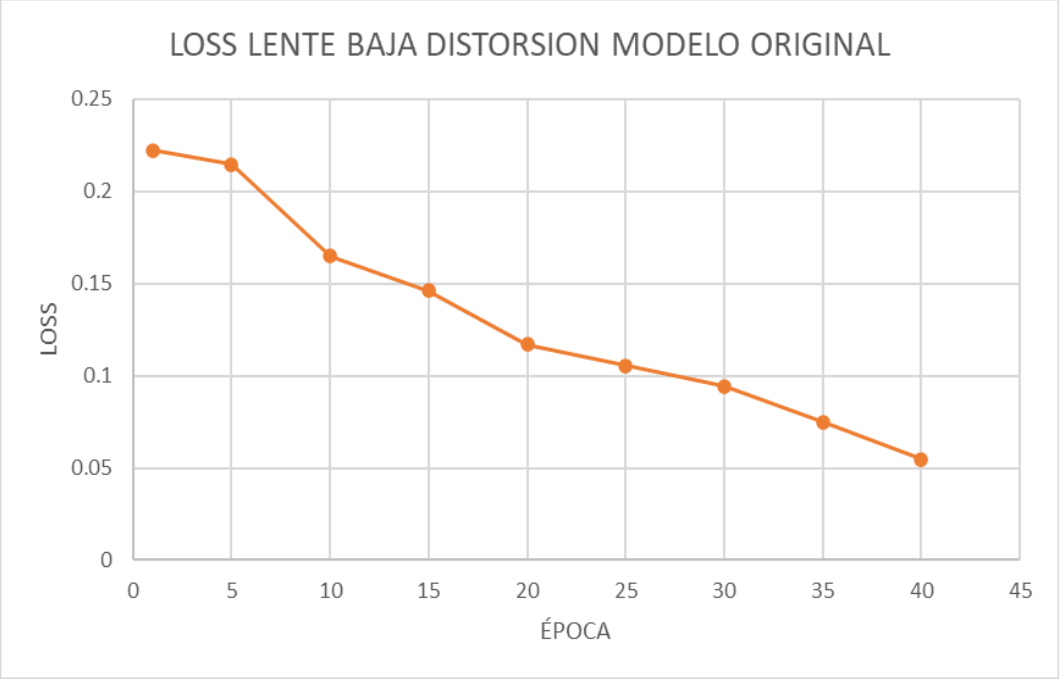
Métrica de VAL_LOSS para lente angular con el modelo original.



Fuente Propia

Figura 67

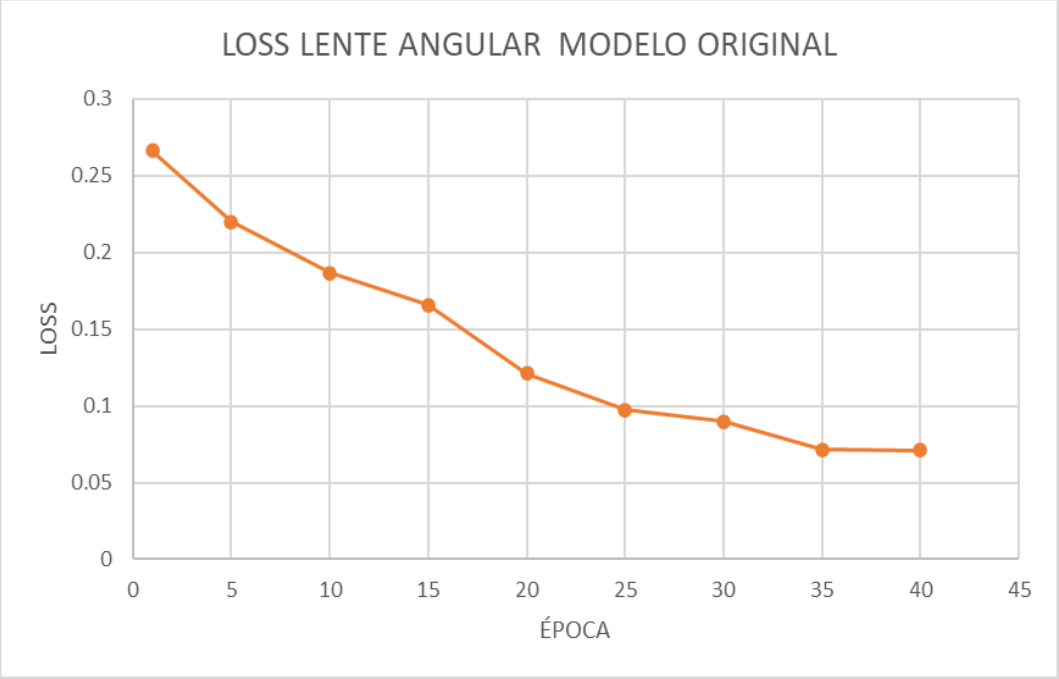
Métrica de LOSS para lente de baja distorsión con el modelo original.



Fuente Propia

Figura 68

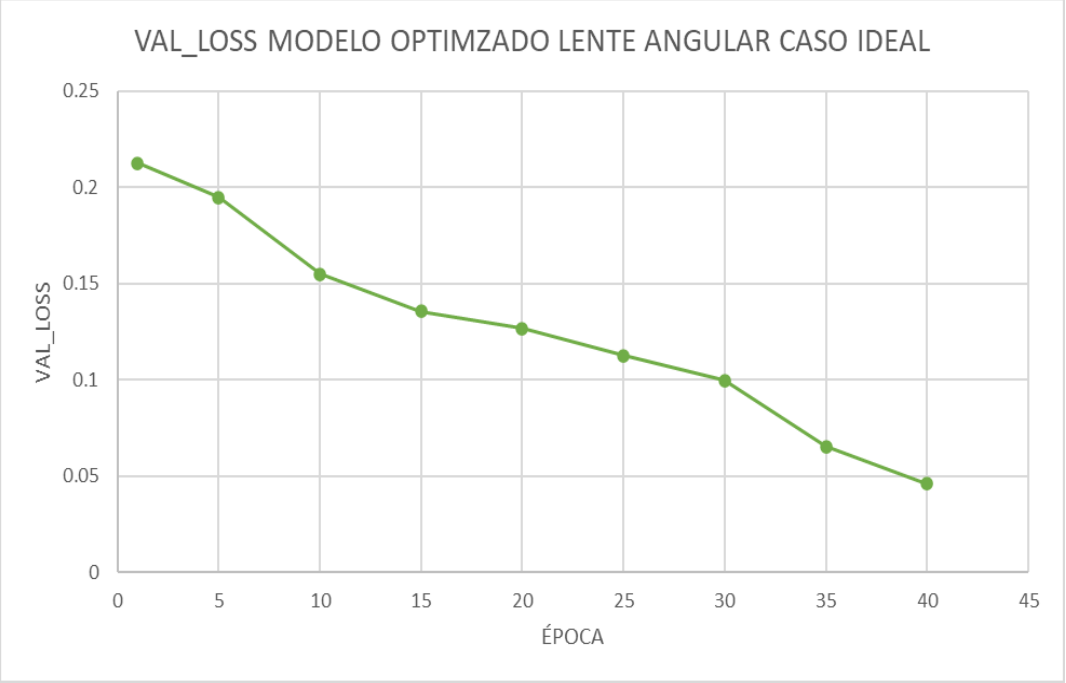
Métrica de LOSS para lente angular con el modelo original.



Fuente Propia

Figura 69

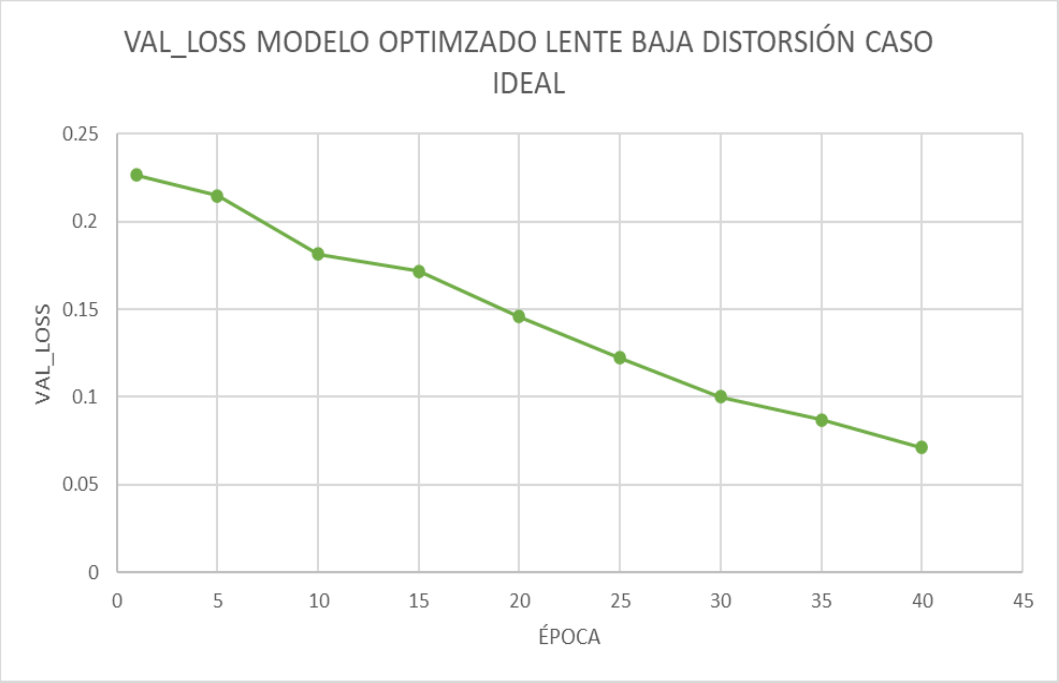
Métrica de VAL_LOSS para lente angular con el modelo optimizado caso IDEAL.



Fuente Propia

Figura 70

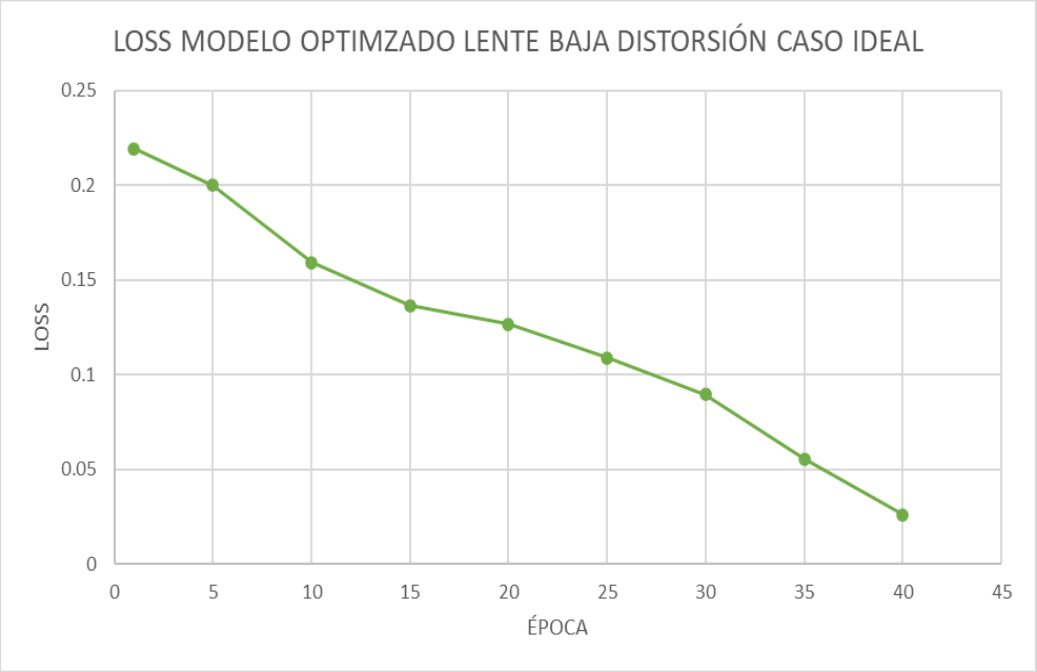
Métrica de VAL_LOSS para lente de baja distorsión con el modelo optimizado caso IDEAL.



Fuente Propia

Figura 71

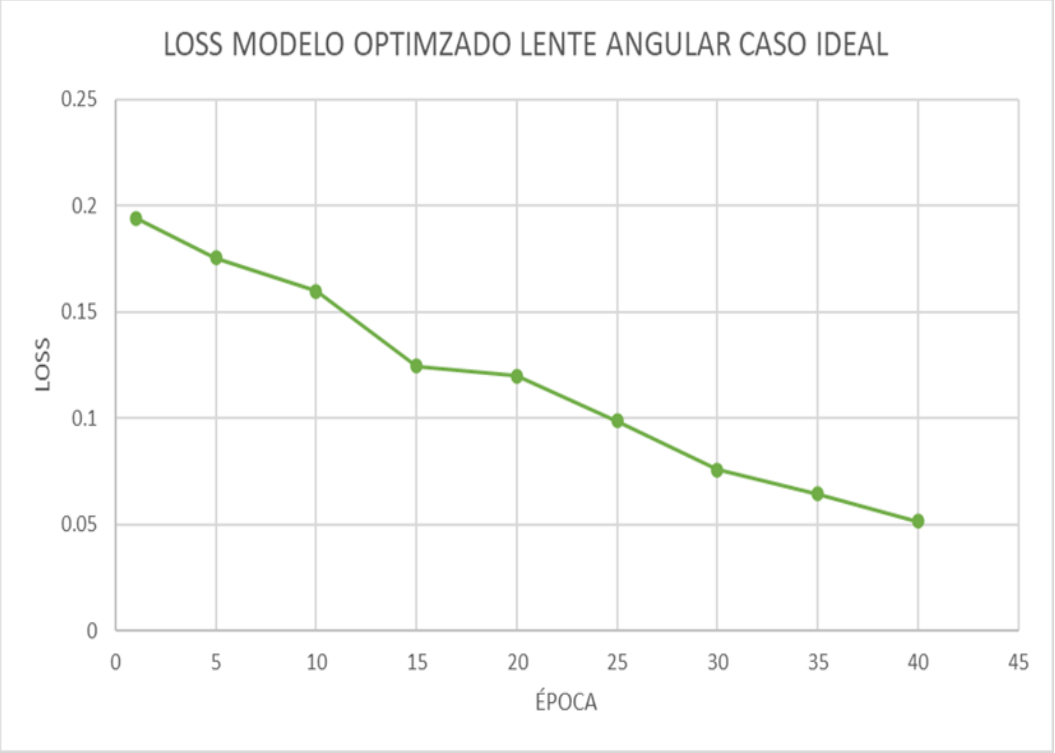
Métrica de LOSS para lente baja distorsión con el modelo optimizado caso IDEAL.



Fuente Propia

Figura 72

Métrica de LOSS para lente angular con el modelo optimizado caso IDEAL.



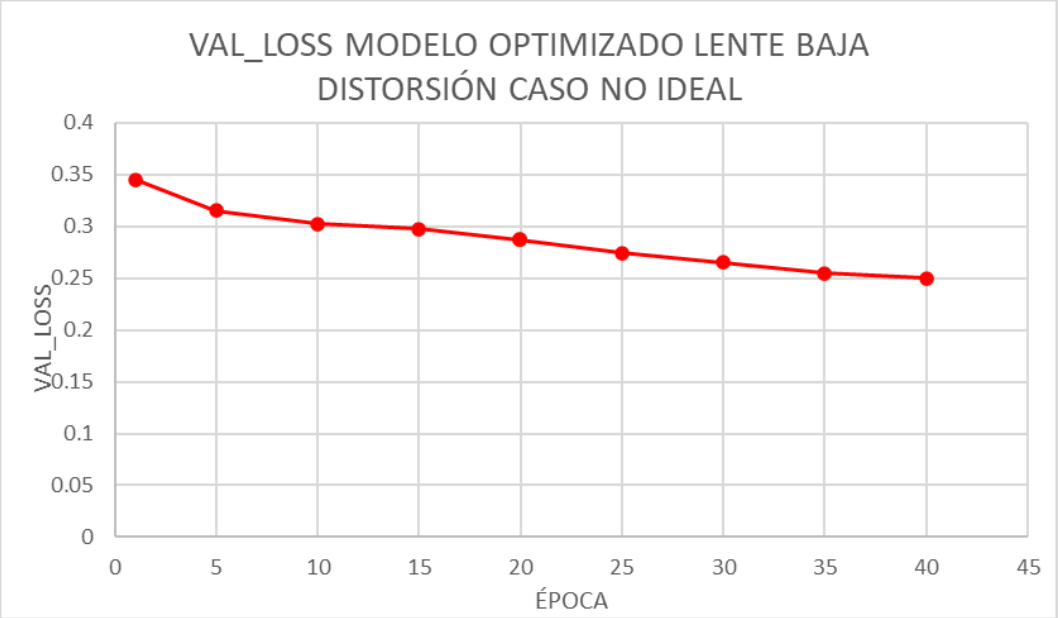
Fuente Propia

Ahora como todo proceso de sintonización, algunas veces se obtienen resultados que no son esperados tales como los expuestos en las figuras 73, 74, 75 y 76, los cuales indican que, aunque el modelo haya sido sintonizado con algunos valores estos en ocasiones no son los adecuados para el entrenamiento, de esta forma, entregan valores que no son idóneos y de menor calidad comparándolos con el modelo original. Tal como se presenta en este caso, donde la métrica de entrenamiento "val_loss" no logra bajar de una marca de 0.21 en ambos casos, mientras que en los modelos originales como ideal si se logró bajar aún más este valor, y así como también sucede con la métrica "loss" que igualmente no bajan de 0.2 en promedio.

De los resultados de entrenamiento alcanzados, se llega a la conclusión que la lente de baja distorsión es la indicada para este proyecto, la cual presenta una ventaja al momento de tener menos pérdida y de esta forma genera un ángulo de giro más certero a la hora de evaluar el automóvil en el circuito de pruebas controlado.

Figura 73

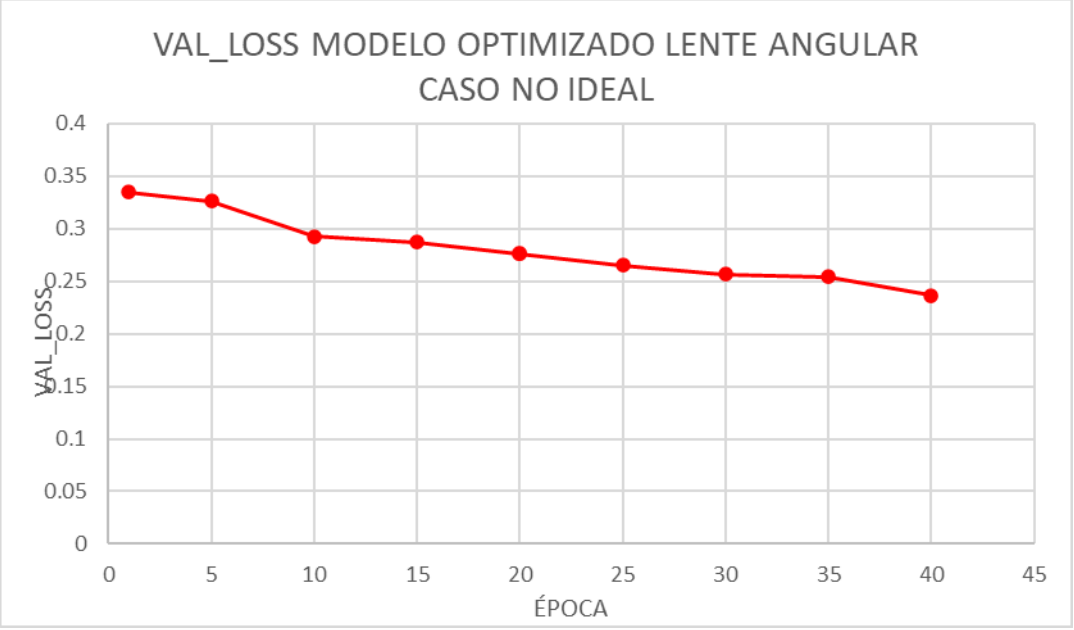
Métrica de VAL_LOSS para lente baja distorsión con el modelo optimizado caso NO IDEAL.



Fuente Propia

Figura 74

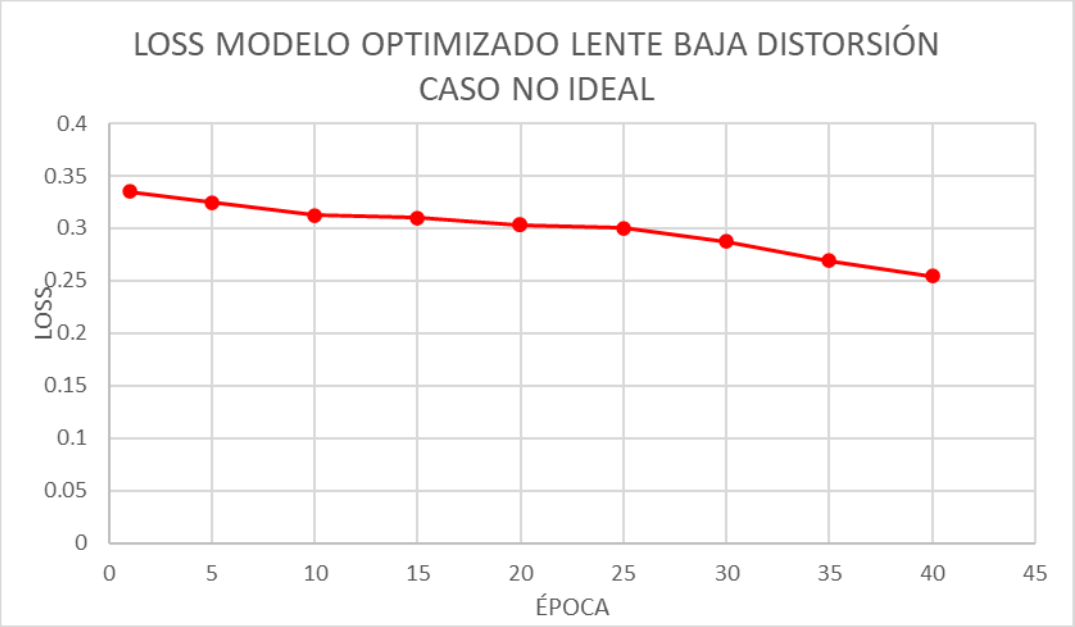
Métrica de VAL_LOSS para lente angular con el modelo optimizado caso NO IDEAL



Fuente Propia

Figura 75

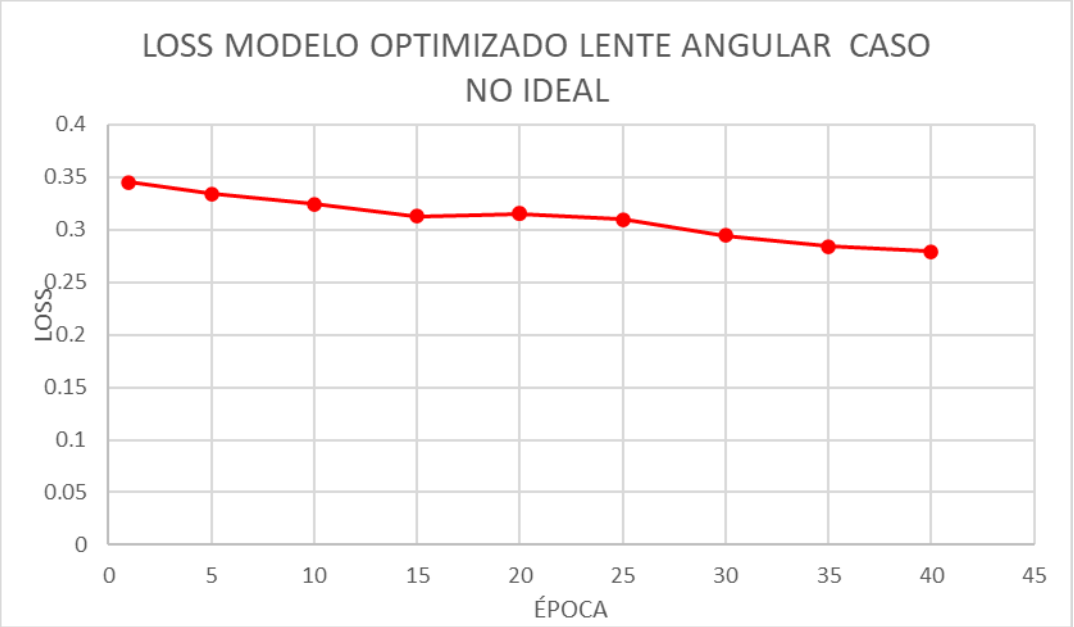
Métrica de LOSS para lente baja distorsión con el modelo optimizado caso NO IDEAL.



Fuente Propia

Figura 76

Métrica de LOSS para lente angular con el modelo optimizado caso NO IDEAL.

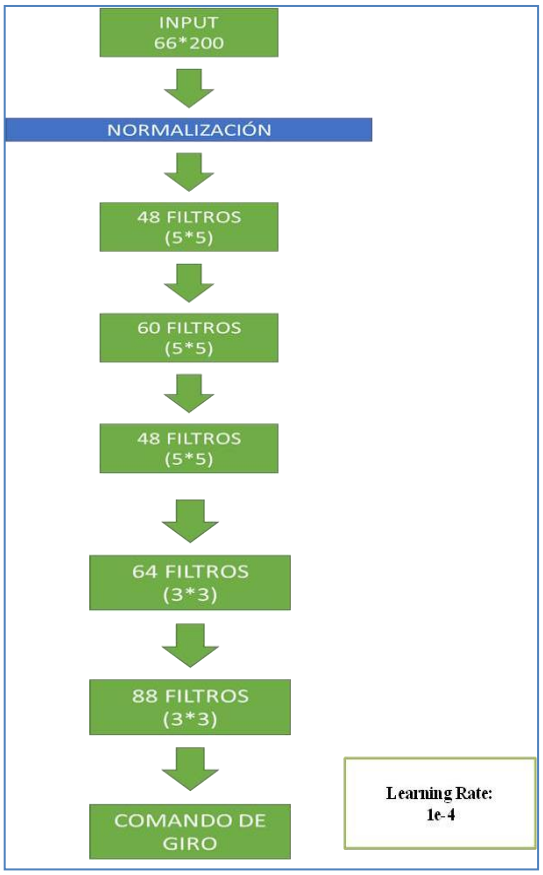


Fuente Propia

La arquitectura final de la red con los hiper-parámetros seleccionados con KERAS-TUNER, posteriormente se observará en la figura 77.

Figura 77

Red Neuronal Optimizada con KERAS-TUNER.



Fuente Propia

4.4. Resultados de la optimización con Tensor-RT

Para realizar las pruebas con Tensor-RT, lo primero que se hizo fue una prueba de inferencia en el circuito de pruebas controlado usando el modelo KERAS-TUNER sin optimizar con Tensor-RT, la prueba consistió en medir el tiempo que demora la red neuronal en dar una inferencia en campos de 10 en 10; además de esto, se midió el uso de recursos de la tarjeta Jetson Nano obteniendo los siguientes gráficos (ver la figura 78). En tales figuras se observa el uso de recursos como la memoria RAM, la cual representó casi el 84% de su totalidad; así como un uso de CPU elevado en el núcleo 2, pasando al tiempo de inferencia, se observa una media de 0.2 segundos de procesamiento por imagen; lo cual es satisfactorio.

Figura 78

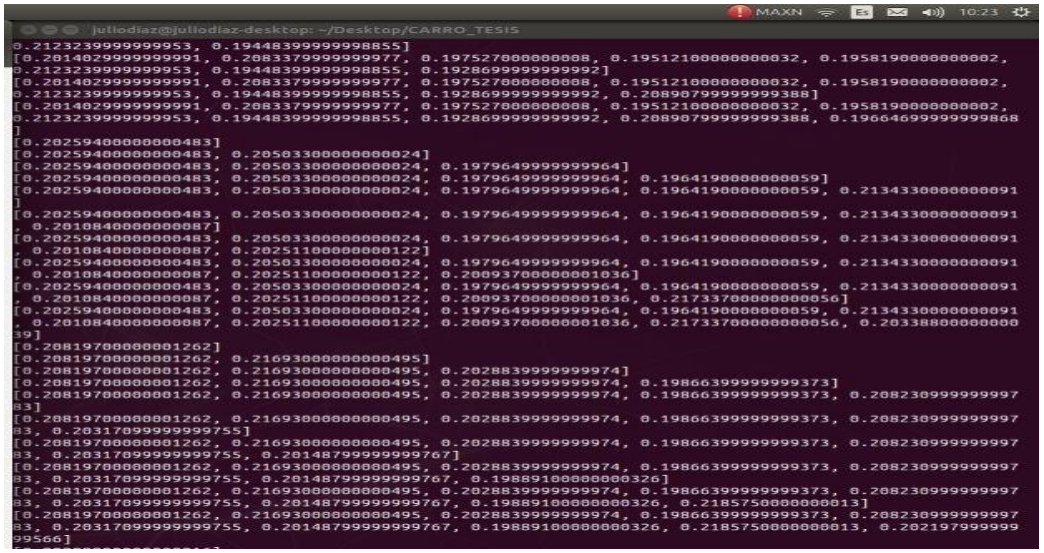
Uso de recursos del modelo no optimizado con Tensor-RT.

Fuente Propia

Sin embargo, a costa de un uso excesivo de recursos como es la memoria RAM, se procedió a realizar la misma prueba, pero esta vez utilizando el modelo optimizado con Tensor-RT. Por lo cual, se obtuvo el siguiente resultado (ver la figura 79).

Figura 79

Tiempo de inferencia no optimizado con Tensor-RT.



Fuente Propia

De esta manera, en las figuras 80 y 81 se observa que el uso de recursos respecto a la memoria RAM fue alrededor del 64% de su totalidad y un uso de CPU elevado en el núcleo 2, pasando al tiempo de inferencia se observó una media de 0.089 segundos de procesamiento; esto dio como resultado una mejora de inferencia de 2.247 veces más rápida que el modelo sin optimizar, y a la vez se contó con un ahorro del 20% de recursos en cuanto a la memoria RAM; por lo cual, se comprobó que si existe una mejora en el modelo cuando se hizo un cambio en el tipo de dato a FP16, lo cual ahorra recursos valiosos que pueden ser empleados en otra tarea. Con respecto al uso de los CPU es casi idéntico en los 2 casos, esto es debido a que la red se ejecuta en la GPU de la tarjeta que a su vez genera un uso idéntico en el CPU. Luego, en las figuras 82 y 83 se muestra una comparación entre el tiempo de inferencia y el uso de los recursos.

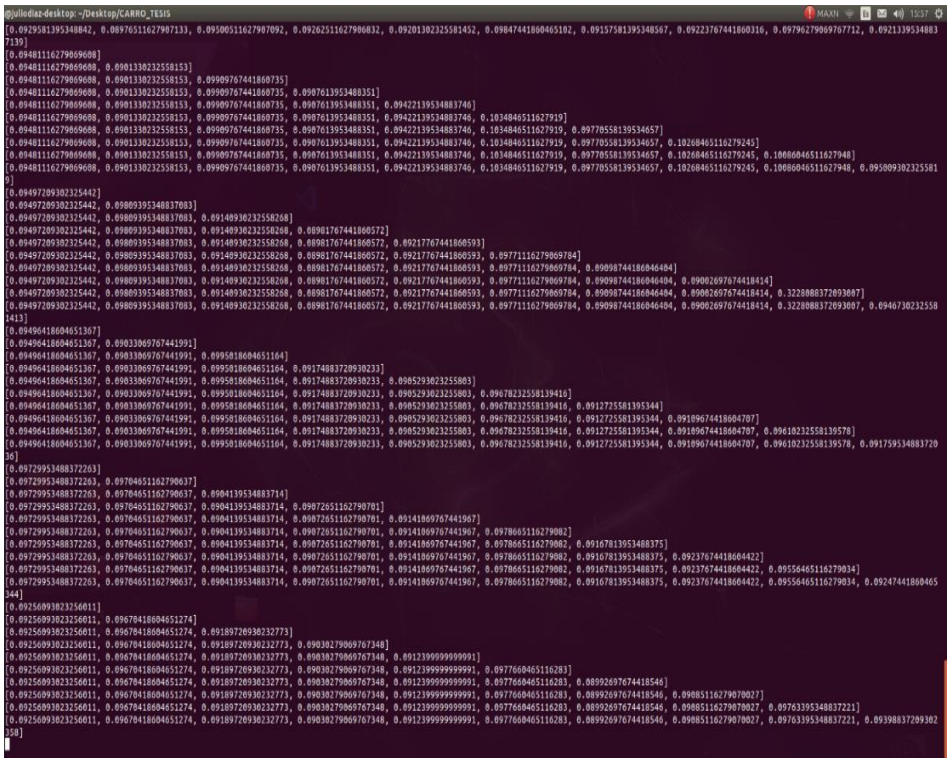
Figura 80

Uso de recursos modelo optimizado con Tensor-RT.

Fuente Propia

Figura 81

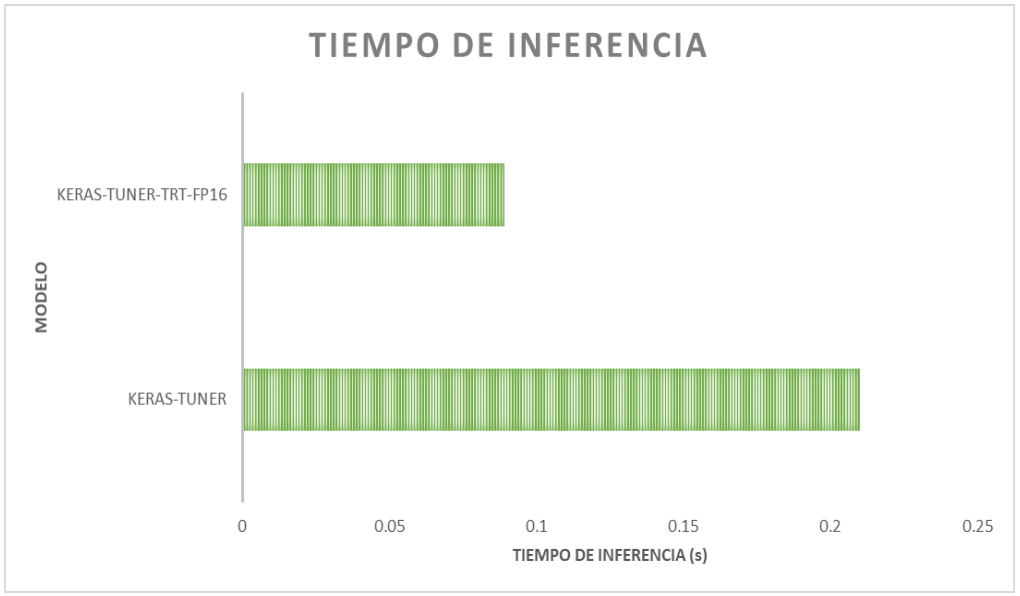
Tiempo de inferencia modelo optimizado con Tensor-RT.



Fuente Propia

Figura 82

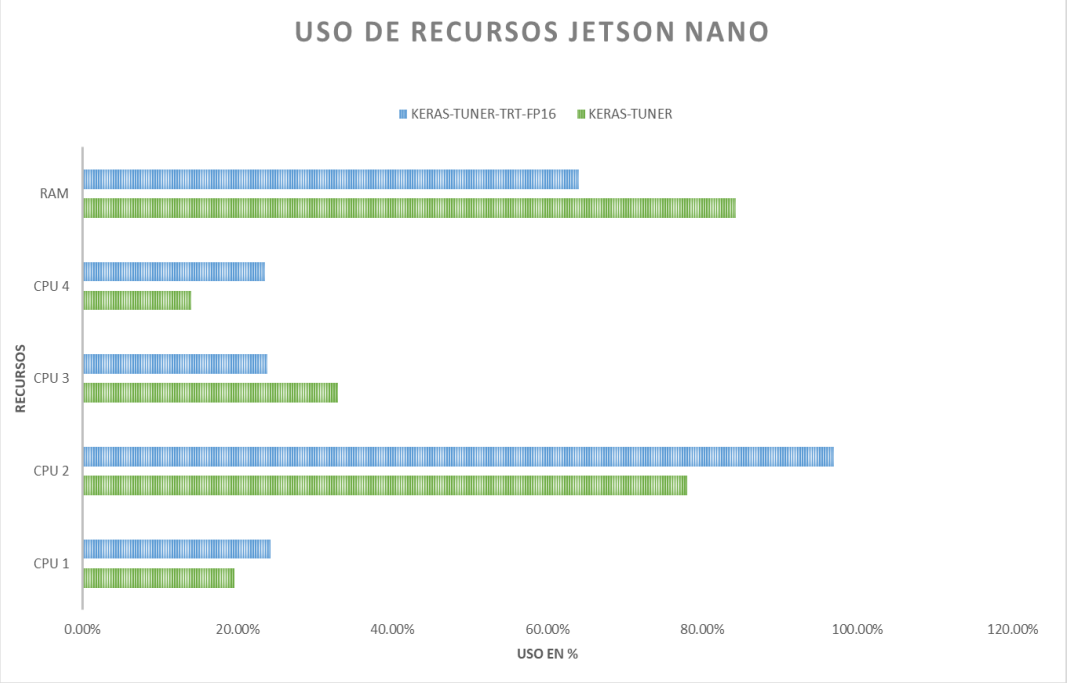
Comparación entre tiempos de inferencia.



Fuente Propia

Figura 83

Comparación de Recursos.



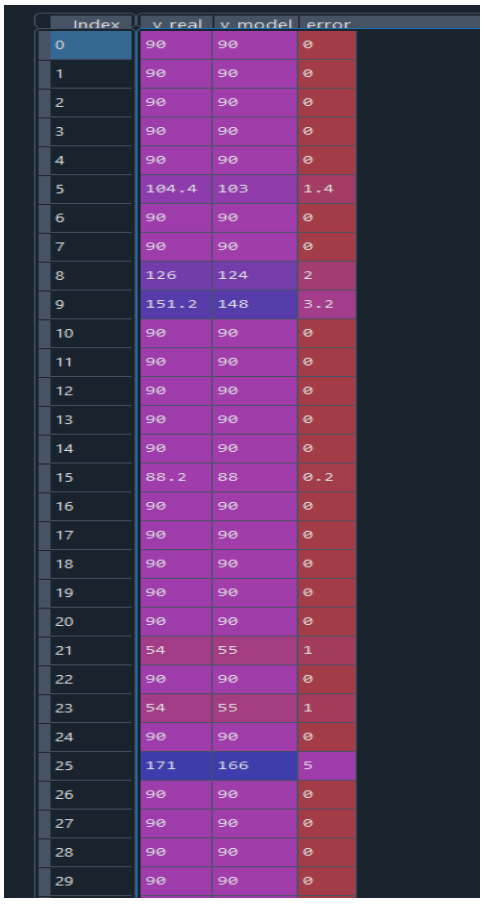
Fuente Propia

4.5. Pruebas de conducción sintéticas con data de entrenamiento en el Circuito Real

Para las pruebas sintéticas se utilizó el modelo con todas las optimizaciones realizadas, así como también la integración de todos los módulos trabajados anteriormente; para ello, en un archivo de cabecera denominado “MAIN.py”, se invocaron todos, comenzando con una prueba de inferencia con las imágenes de entrenamiento en el circuito de conducción controlado sin la presencia de las señales, luego comparando el valor real, el predicho y la diferencia entre estos valores. Todo esto dio como resultado lo mostrado en la figura 84.

Figura 84

Comparación de valor real y predicho.



Index	y_real	y_model	error
0	90	90	0
1	90	90	0
2	90	90	0
3	90	90	0
4	90	90	0
5	104.4	103	1.4
6	90	90	0
7	90	90	0
8	126	124	2
9	151.2	148	3.2
10	90	90	0
11	90	90	0
12	90	90	0
13	90	90	0
14	90	90	0
15	88.2	88	0.2
16	90	90	0
17	90	90	0
18	90	90	0
19	90	90	0
20	90	90	0
21	54	55	1
22	90	90	0
23	54	55	1
24	90	90	0
25	171	166	5
26	90	90	0
27	90	90	0
28	90	90	0
29	90	90	0

Index	y_re	y_model	error
1423	78.3	78	0.3
1845	78.3	78	0.3
2152	78.3	78	0.3
2179	78.3	78	0.3
2609	78.3	78	0.3
3247	78.3	78	0.3
3443	78.3	78	0.3
3568	78.3	78	0.3
3877	78.3	78	0.3
4006	78.3	78	0.3
4055	78.3	78	0.3
4117	78.3	78	0.3
161	76.5	77	0.5
241	76.5	77	0.5
334	76.5	77	0.5
541	76.5	77	0.5
610	76.5	77	0.5
1029	76.5	77	0.5
1078	76.5	77	0.5
1085	76.5	77	0.5
1209	76.5	77	0.5
1253	76.5	77	0.5
1634	76.5	77	0.5
1708	76.5	77	0.5
1785	76.5	77	0.5
1786	76.5	77	0.5
1803	76.5	77	0.5
1889	76.5	77	0.5
1951	76.5	77	0.5
1977	76.5	77	0.5

Fuente Propia

De lo obtenido, se observa una predicción perfecta en los casos de un comando de giro de 90°. Esto significa que el automóvil está centrado; pero, cuando se trata de predecir valores de giro a izquierda o derecha, se obtuvo un error que va desde 0.2° a 5.4°grados, y en el caso menos óptimo un error medio absoluto de 0.554. Además de eso, se hizo una prueba de correlación de variables que dio como valor la cantidad de 0.9742, loque corresponde a un valor confiable para la predicción de comandos de giro.

4.6. Pruebas de conducción en el Circuito Real

Para esta prueba se realizó una vuelta por el circuito real, siguiendo las siguientes reglas:

- La velocidad de desplazamiento es del 70% del duty cycle de los motores.
- Cuando se detecte una señal de STOP el automóvil parará 10 segundos para luego retomar su trayecto conservando su velocidad.

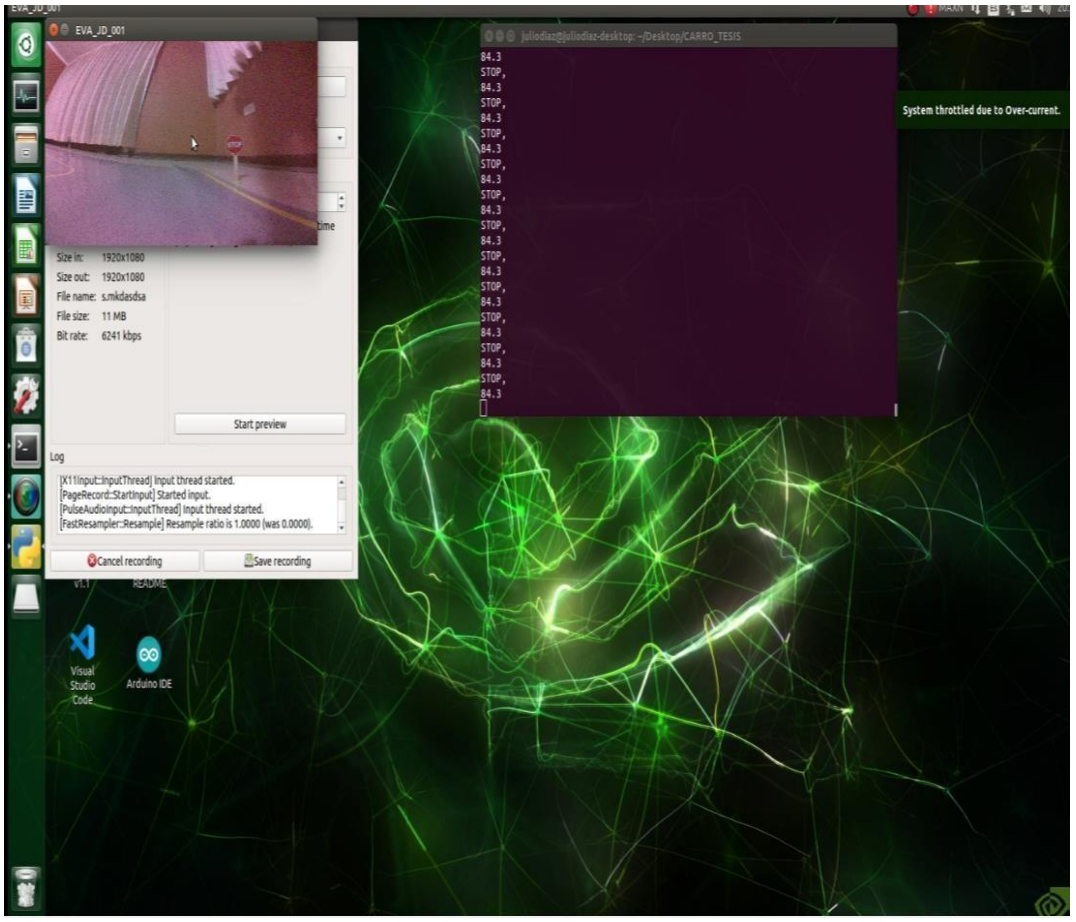
- Cuando se detecte un Semáforo en Rojo, el automóvil parará 30 segundos. Después de este tiempo, continuará su trayectoria con la misma velocidad.
- Cuando se detecte a un Peatón, el automóvil se detendrá de inmediato siempre y cuando el sensor ultrasonido detecte que se encuentra a 10 centímetros del sensor.
- Al detectar una señal de límite de velocidad, se realizará un cambio de velocidad a lo que indique la señal; por lo cual, esta será expresada en el valor de duty cycle de los motores en ese momento.

Teniendo las reglas establecidas anteriormente, se procedió a probar el sistema y observar su comportamiento.

En las siguientes figuras 85, 86 y 87, se observa a través de la cámara del automóvil, la situación en la que se encuentra este respecto a la perspectiva de todo el sistema planteado. Es decir, en la figura 85 el vehículo está ejecutando la red neuronal como también los demás módulos en el circuito de pruebas dentro del carril delimitado; además, también se observa en la figura la impresión en el prompt del sistema el comando de giro predicho por la red neuronal, el cual en ese momento fue de 84.3° grados y asimismo la detección de la señal de tráfico que en ese instante fue STOP. En la siguiente figura 86, se observa que el ángulo predicho cambia a 88.4° además de la detección del semáforo en verde, dando un resultado correcto al momento de detectar la señal de tráfico mientras que se tiene un ligero error de 1.6°, al momento de predecir el ángulo de giro dado que en ese instante el automóvil debería estar centrado con un ángulo de 90°.Y, en la figura 87 se tiene el mismo ángulo de giro predicho que en la figura anterior, y una detección correcta del límite de velocidad que en ese caso es del 50% del duty cycle de la velocidad de los motores.

Figura 85

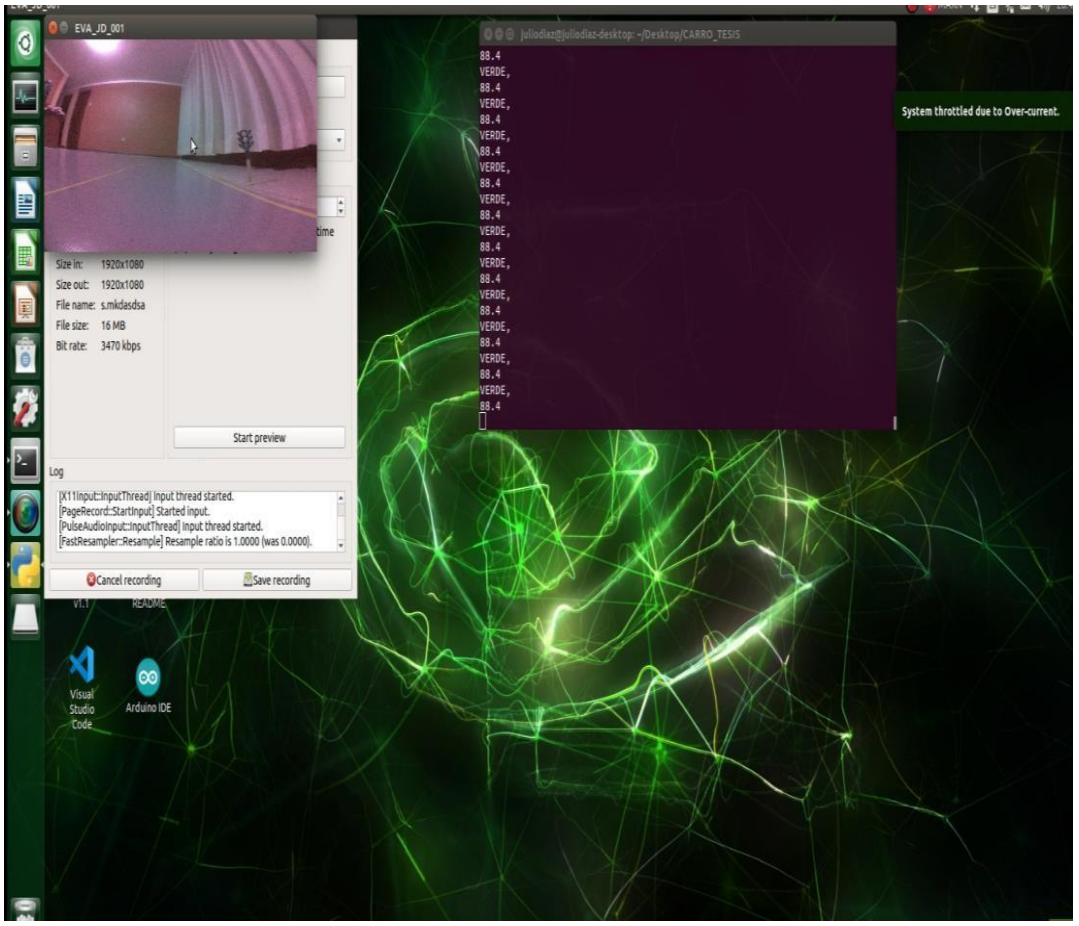
Detección de Señal STOP y comando de GIRO.



Fuente Propia

Figura 86

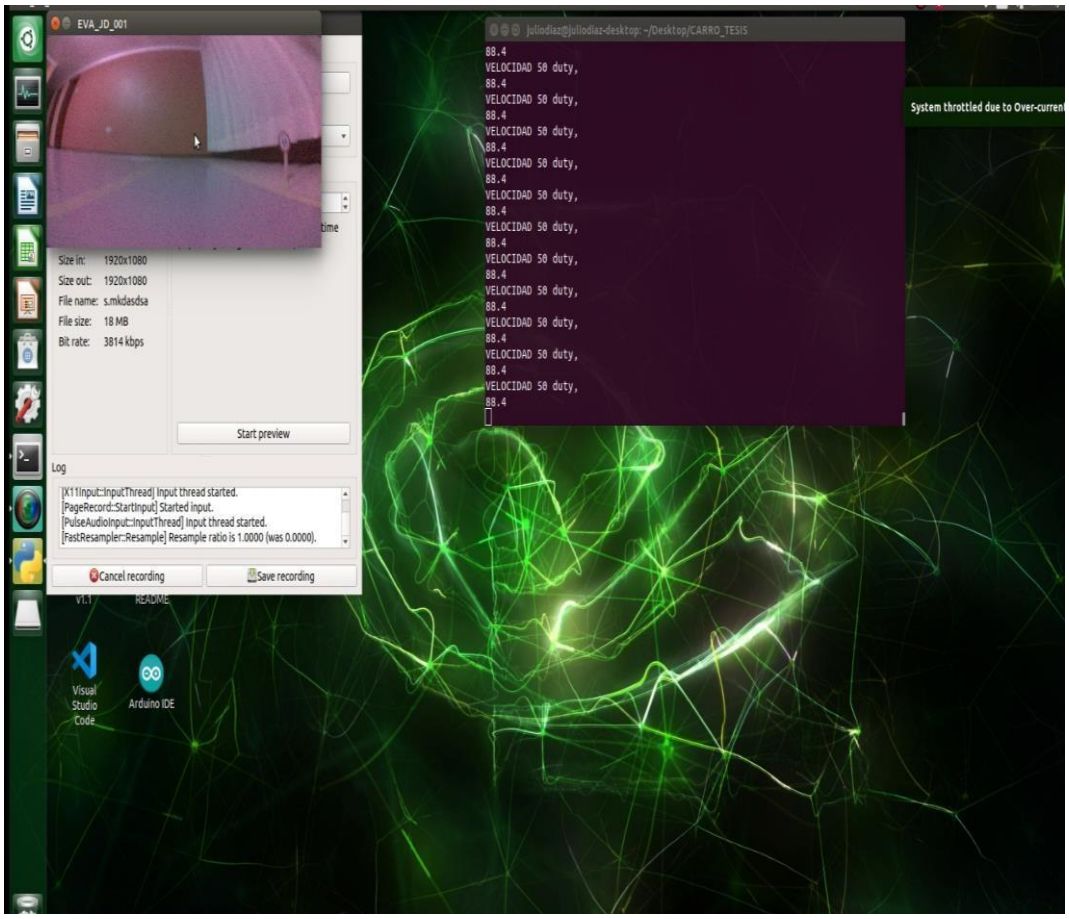
Detección de Semáforo en Verde y comando de giro.



Fuente Propia

Figura 87

Detección de Semáforo en Verde y comando de giro.



Fuente Propia

CONCLUSIONES

a. Se implementó un automóvil a escala basado en 2 sistemas, uno de alto nivel usando una Jetson Nano conformado por módulos escritos en lenguaje Python y que fue mostrado en la figura 27, donde se encargó de controlar el automóvil realizando una predicción; y, un segundo sistema de bajo nivel mostrado en la figura 31, que se encargó de recolectar la información de un sensor ultrasonido la cual fue enviada al sistema de alto nivel para su debido procesamiento, dando como resultado una comunicación entre los 2 dispositivos, y logrando un funcionamiento de manera correcta; por lo cual, los datos del sensor ultrasonido son recibidos y no poseen ningún error en la información; sin embargo, si se coloca el objeto muy próximo a una esquina del sensor, este no logra ser detectado.

b. Se logró recolectar información del recorrido del automóvil en el circuito de pruebas mostrado en la figura 41, donde se capturaron los atributos necesarios para el entrenamiento (imagen y ángulo de giro) con apoyo de los módulos implementados, y así como también con el contenido del archivo .CSV donde se tiene toda la información necesaria para la tarea de entrenamiento. Asimismo, esta información fue balanceada a partir de un número de muestras máximas, lo que ayudó a contar con un DATASET más balanceado tal como se mostró en las figuras 47 y 48.

c. El algoritmo de visión artificial para la detección de señales de tráfico, peatones y semáforos mediante la técnica Haar Cascade funcionó de manera correcta detectando la señal y procediendo a interactuar con lo que esta indica, tal como se explicó en la sección 4.6 definida como pruebas de conducción en el circuito real; sin embargo, para un mayor número de señales a detectar se recomendaría entrenar una nueva red neuronal convolucional y el uso de una lente angular para ampliar el rango de visión del sistema, con lo cual se obtuvieron mejores resultados que fueron compartidos en las tablas 7 y 8.

d. Según lo comparado entre los mejores modelos obtenidos de redes neuronales convolucionales, se observa que el modelo optimizado con KERAS-TUNER posee el valor de pérdida más bajo en comparación con los demás modelos, dicho valor tuvo como resultado 0.01211 lo cual da a concluir que el uso de un sintonizador de

hiper-parámetros, como lo es el KERAS-TUNER, mejora notablemente los resultados a la hora del entrenamiento, pero es posible mejorar aún más haciendo uso por ejemplo de un campo de búsqueda más extenso o agregando hiper-parámetros a la búsqueda, lo que generaría mejores resultados a costa de un mayor tiempo de búsqueda. Luego de la sintonización, se procedió con el entrenamiento del sistema usando TENSORFLOW GPU dando una media de tiempo de 100 a 110 minutos de entrenamiento, además en las pruebas de inferencia, se determinó que el modelo optimizado usando FP16 tuvo un tiempo promedio de inferencia de 0.089 segundos, mientras que la prueba del modelo original sin optimización tuvo un tiempo de 0.2 segundos lo que demuestra la mejora de tiempos. De esta forma, se permitió obtener una mejora del 2.247 con respecto al otro modelo, por lo cual todos estos resultados se apreciaron en las figuras 82 y 83.

e. Según vemos de las figuras, 85, 86 y 87, que corresponden a la prueba en el circuito real, no se tiene un ángulo de giro como arrojaban las pruebas sintéticas, esto puede ser por varios motivos y el más evidente es la iluminación del momento o pequeñas variaciones en el circuito de pruebas; sin embargo, el automóvil se mantiene en el carril establecido y cumple la detección de señales de tráfico, así como también responde al entorno donde está navegando, lo que otorga una autonomía de nivel 4, y con un uso de recursos finales de 2.5 gigas de memoria mostrados en la figura 80 que consume la red neuronal convolucional y otros 1.1 gigas usados por el algoritmo de visión artificial dando un consumo total de 3.6 gigas , algo muy positivo sabiendo de todo el proceso que tiene que realizar.

RECOMENDACIONES

1. Tener un circuito de pruebas con delimitadores de carril, de preferencia, usar cinta de color para delimitar el área; así, el automóvil puede detectar de mejor manera la zona por donde va a transitar.

2. Colocar la cámara del automóvil en una altura donde esta pueda captar el carril como parte del entorno.

3. Realizar un entrenamiento de mínimo 10 vueltas en una dirección y otras 10 en la dirección contraria.

4. Calibrar el automóvil antes de iniciar el recorrido, para esto es necesario colocar a 0° el ángulo de giro.

5. Realizar el entrenamiento de la red en una computadora de gama alta, de preferencia tener una capacidad de cómputo al menos de 7.5 para así acortar tiempos de entrenamiento.

6. Para futuros trabajos es recomendable el uso de 2 lentes, una de baja distorsión para la navegación del automóvil y otra lente angular para poder captar más elementos del entorno; además, se recomienda el uso de un sistema más potente de la marca NVIDIA como puede ser la NVIDIA JETSON XAVIER NX que tiene el doble de capacidad de memoria RAM, con el cual se hace posible ejecutar más tareas y por ende hacer uso de 2 cámaras en simultaneo.

REFERENCIAS BIBLIOGRÁFICAS

Almenara, J. P. L. (2019). Paro de colectivos y por qué Lima es la tercera ciudad del mundo con más tráfico vehicular | #NoTePases. *El Comercio Perú*. https://elcomercio.pe/lima/transporte/lima-tercera-ciudad-mundo-congestion-vehicular-400-noticia-ecpm-642900-noticia/

Atmel (2021). *8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash*. Microchip.com

Automated vehicles for safety. (s/f). Nhtsa.gov. Recuperado de https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

Bechtel, M. G., McEllhiney, E., Kim, M., & Yun, H. (2017). DeepPicar: A low-cost deep neural network-based autonomous car. arXiv [cs.OH].2

Bejerano, P. G. (2014). *Historia de los coches autónomos: esa gran desconocida*. Blogthinkbig.com. https://blogthinkbig.com/historia-de-los-coches-autonomos

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). End to end learning for self-driving arXiv *[cs.CV]*. https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

Castro, P. C. (2016). La imprudencia y alta velocidad causan mayoría de accidentes. El Comercio Perú. https://elcomercio.pe/lima/imprudencia-alta-velocidad-causan-mayoria-accidentes-155836-noticia/

CUDA Zone. (2021, 26 julio). NVIDIA Developer. https://developer.nvidia.com/cuda-zone

David E. Rumelhart; James L. McClelland, "A General Framework for Parallel Distributed Processing," in Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations, MIT Press, 1987, pp.45-76.

Jetson Nano Developer Kit. (2019). Nvidia.com. https://developer.nvidia.com/embedded/jetson-nano-developer-kit

Keras Team. (2021). *KerasTuner*. Keras.io. Recuperado de https://keras.io/keras_tuner/

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016). Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv [cs.LG]*.

Manual De Dispositivos De Control Del Transito Automotor Para Calles Y Carreteras. (2016). MTC. Recuperado de. https://portal.mtc.gob.pe/transportes/caminos/normas_carreteras/documentos/manuales/mANUAL%20de%20DISPOSITIVOS%20de%20CONTROL%20del%20TRANSITO%20FINALIZADO_24%20MAYO_2016.pdf

Mena, F. G. (2018). Usuarios pierden hasta 12 años de su vida por congestión vehicular en Lima. *Gestión*. https://gestion.pe/economia/usuarios-pierden-12-anos-vida-congestion-vehicular-lima-251738-noticia/

N. Deepika and V. V. Sajith Variyar, "Obstacle classification and detection for vision-based navigation for autonomous driving," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017, pp. 2092-2097.

National Highway Traffic Safety Administration. (2018). *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*. NHTSA. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812506#:~:text=Sleep%20was%20the%20most%20common,1.9%25)%20of%20the%20drivers.

Nvidia (2016). GPU-Based Deep Learning Inference: A Performance and Power Analysis [White paper]. https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf

NVIDIA TensorRT. (2021, 2 diciembre). NVIDIA Developer. https://developer.nvidia.com/tensorrt

OpenCV (2021). OpenCV. Recuperado de https://opencv.org/

P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 2001, pp. I-I.

Rojas, E. (2018). *Los coches autónomos podrían reducir hasta un 90% los accidentes de tráfico*. MuyComputer PRO. Recuperado de: https://www.muycomputerpro.com/2015/03/05/coches-autonomos-accidentes-

trafico

TensorFlow. (2021). Tensorflow.org. Recuperado de https://www.tensorflow.org/

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, *59*(236).

W. Vijitkunsawat and P. Chantngarm, "Comparison of Machine Learning Algorithm's on Self-Driving Car Navigation using Nvidia Jetson Nano," 2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2020, pp. 201-204.

Y. Han and E. Oruklu, "Traffic sign recognition based on the NVIDIA Jetson TX1 embedded system using convolutional neural networks," 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017, pp. 184-187.

Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.