Implementación de una Red Neuronal Convolucional embebida en un automóvil a escala para la conducción autónoma en entornos reales controlados

Implementation of a Convolutional Neural Network embedded in a scale automobile for autonomous driving in real controlled environments.

Julio Díaz Villanueva, estudiante de pregrado
Universidad Ricardo Palma, Perú, julio.diaz@urp.edu.pe

***Abstract*– In this research work, a tool from the KERAS-TUNER library was used to improve an existing neural network used for autonomous driving, in addition to that, the neural network was optimized with the hyperparameters modified with TENSOR-RT using the FP16 data type. All this was embedded in a NVIDIA JETSON-NANO development board where an autonomous driving system was implemented as well as a data collection system. The results were an improvement in validation loss and 2.24 faster inference than the non-optimized model.**

En este trabajo de investigación se utilizó una herramienta de la librería KERAS-TUNER para mejorar una red neuronal existente usada para la conducción autónoma, además de eso se optimizó dicha red neuronal con los hiperparámetros modificados con TENSOR-RT usando el tipo de dato FP16. Todo ello fue embebido en una tarjeta de desarrollo NVIDIA JETSON-NANO donde se implementó un sistema de conducción autónoma como también de recolección de datos. Los resultados fueron una mejora en la pérdida de validación y una inferencia 2.24 más rápida que el modelo no optimizado.

Keywords—Self driving, Jetson-Nano, Keras-Tuner, Tensor-RT.

I. Introducción

Actualmente, el desarrollo de automóviles autónomos es muy común en los países desarrollados y pueden ayudar con la reducción de 1/3 de los accidentes producidos por errores humanos según el IIHS [1]. Lamentablemente, el desarrollo de este tipo de tecnología es costosa y requiere una inversión elevada para desarrollarla de manera eficiente. Es así que, la empresa Tesla [2] es uno de los principales impulsores, porque fabrica automóviles eléctricos y autónomos que cuentan con cámaras, sensores de proximidad, entre otros accesorios distribuidos en todo el automóvil, permitiendo el desplazamiento autónomo y la toma de las mejores decisiones en circunstancias normales y difíciles.

Pero ¿Cómo una máquina puede predecir y tomar decisiones correctas? La respuesta se encuentra en el uso de inteligencia artificial, y específicamente en el uso de una red neuronal convolucional [3].

En el año 2020, en la investigación desarrollada por Wuttichai Vijitkunsawat y Peerasak Chantngarm [4], se implementaron 3 tipos de arquitecturas de redes neuronales para conducción autónoma, siendo una de ellas la red neuronal convolucional recurrente de largo plazo que tuvo el mejor porcentaje de asertividad en el momento de las pruebas porque alcanzó un 88.7%.

Por otro lado, en el año 2016, la empresa NVIDIA [5] 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 a su vez tienen la función de capturar fotos del trayecto; además, se obtuvo el grado de giro del timón el cual unido con la imagen capturada en ese momento, permitió generar una data para entrenar al sistema y logrando obtener un 90% de autonomía en las pruebas realizadas.

Luego, en [6], se implementó un automóvil autónomo de bajo costo usando la red neuronal implementada en el proyecto End-to-End Deep Learning for Self-Driving Cars [5] en el cual se logró desarrollar un sistema de conducción autónoma de bajo costo el cual funcionaba obteniendo una imagen de entrada la cual tenía que ser modificada para tener una dimensión de 200*66 pixeles para ser compatible con la red neuronal usada, con esta imagen se procederá a realizar el proceso de inferencia con la red neuronal lo cual arroja un comando de giro que va hacia los motores para cambiar la dirección del automóvil luego de esto se hicieron pruebas como el tiempo de inferencia del modelo lo cual arrogó un tiempo promedio de 23.74 ms para completar todos los procesos requeridos para la conducción haciendo uso de los 4 núcleos del Cortex-A53 de la Raspberry Pi [7]; a la misma vez se realizaron pruebas también en un sistema embebido Intel UP y otro de la marca NVIDIA Jetson TX2; de los resultados obtenidos claramente podemos ver 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, pero viendo el otro extremo vemos la gran diferencia entre una TX2 usando su GPU dando un tiempo de ejecución de 6 ms aproximadamente frente a los 23.74ms teniendo una mejora de casi un 400% de menor tiempo de inferencia, lo que comprueba lo dicho en [9]; para finalizar

podemos ver que el rendimiento es sorprendente el costo de una TX2 ronda los $600 mientras que una Raspberry Pi 3 (B) cuesta $35, para un modelo a escala el costo sería muy elevado por lo cual se debe buscar un equilibrio entre costo rendimiento.

Asimismo, en [8], el objetivo fue clasificar obstáculos (peatones, automóviles, etc) usando técnicas como la segmentación de la imagen, delimitar una región de interés usando la técnica CONVEX-HULL, encerrar un posible candidato para obstáculo y dar a conocer si este se encuentra dentro o fuera del área de interés usando el famoso POINT POLYGON TEST. Por lo cual, los resultados obtenidos fueron aceptables; por ejemplo, para un peatón dentro de la ROI se tuvo un 86.38% de confianza, mientras que para un carro dentro de la ROI se tuvo un 96.67% de confianza; estos resultados fueron obtenidos usando un sistema embebido NVIDIA JETSON TX1 donde se ejecutó el algoritmo. Algo resaltante de este trabajo fue que el autor dio a conocer que es posible obtener mejores resultados usando un sistema con mayor potencia.

En el whitepaper de NVIDIA [9] nos comenta cuales son las mejoras en términos de performance y consumo energético usando una GPU. Teniendo en cuenta que en el año 2012 con el desarrollo de la red neuronal ALEXNET [10] fue 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 dejando a lado el lento proceso de una CPU. Todo esto originó los optimizadores para trabajar con estas tecnologías como puede ser librerías como CUDNN basada de la arquitectura CUDA [11] y nuevos tipos de datos como el FP16 lo que amplió la mejora en términos energéticos y de memoria en comparación con el FP32.Ademas se hicieron pruebas entre el proceso de inferencia de una red neuronal el cual es simplemente dar una entrada nueva a una red neuronal entrenada la cual tiene que deducir el resultad final, para esto es recomendable primero entrenar la red neuronal con una amplio BATCH SIZE para luego testear la red con un variado grupo de imágenes para ver 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 de inferencia 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 dispositivos de mayor costo

Por las razones anteriormente señaladas, en este trabajo se determinó utilizar una tarjeta de desarrollo JETSON-NANO [12] para implementar una red neuronal convolucional embebida y mejorada, utilizando adicionalmente KERAS-TUNER [13] y el motor de inferencia TENSOR-RT [14] para sintonizar los hiper-parámetros, y así lograr una conducción autónoma de un automóvil a escala en entornos reales controlados.

A continuación, en la segunda sección se aborda la metodología y el desarrollo de este proyecto, haciendo énfasis en la implementación del vehículo autónomo, la obtención de datos y el procesamiento, el procesamiento de las imágenes obtenidas y el respectivo entrenamiento, así como la optimización utilizando TENSOR-RT. Luego, en la tercera sección se mencionan los resultados alcanzados en las diversas pruebas realizadas; y, en la última sección, las conclusiones correspondientes a este tema desarrollado.

Digital Object Identifier: (only for full papers, inserted by LACCEI).
ISSN, ISBN: (to be inserted by LACCEI).
DO NOT REMOVE

II. METODOLOGÍA Y DESARROLLO

Para explicar la metodología empleada en el desarrollo de este trabajo, inicialmente se hace la aclaración que 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.

Los hiper-parámetros se pueden seleccionar de manera manual; pero, existe una librería con la cual es posible encontrar los mejores hiper-parámetros para el modelo propuesto. Esta librería lleva el nombre de KERAS-TUNER [11] y fue utilizada para sintonizar los mejores hiper-parámetros. Para ello, como primer paso, se tuvo que establecer un hiper-modelo de tal forma que se comportara como un campo de búsqueda y sintonización del hiper-parámetro. De esta forma, los hiper-parámetros fueron la cantidad de números de filtros y la learning rate. En la figura 1 se muestra el diagrama representativo de este proceso, comenzando con la configuración de los hiper-parámetros, continuando con el entrenamiento, luego la validación, y finalmente el modelo con el mejor resultado.

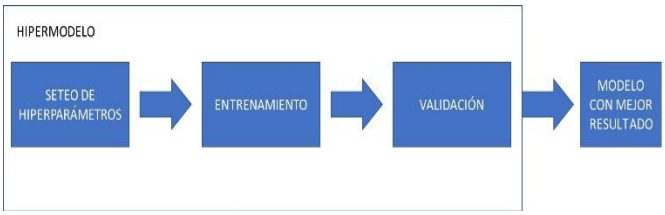

Fig. 1 Diagrama representativo para Keras Tuner.

Una vez establecido el lugar de la búsqueda y los valores de los hiper-parámetros, se continuó con el método de búsqueda; asimismo, en este trabajo se utilizó la búsqueda aleatoria para los valores de los hiper-parámetros. Estos a su vez, tienen un valor mínimo y máximo, o como también una selección específica de valores, por ejemplo:

- Número de filtros
  Para este caso en particular se utilizó la clase INT.
  La sintaxis utilizada fue de la siguiente manera:
  hp.Int('Filtros', min_value = 24, max_value = 256, step = 16)
- Learning Rate (lr)

Para escoger un lr correcto se utilizó la clase CHOICE en la cual se escoge un valor comprendido entre N selecciones.
La sintaxis utilizada fue de la siguiente manera:
hp.Choice('lr', values = [0.00001, 0.0001])

De este modo, una vez ejecutada la búsqueda se obtuvieron los diferentes entrenamientos con los diversos parámetros, y se prosiguió con la selección del que mejor convenga, por lo cual hubo una demora aproximada de 4 horas en obtener tales resultados.

A. Implementación del vehículo autónomo

Según [5], se utilizó un tipo de red neuronal convolucional artificial profunda de aprendizaje supervisado, con una considerable cantidad de capas ocultas. Asimismo, para la simulación en un entorno real, se optó por utilizar la tarjeta de desarrollo o también conocido como hardware embebido JETSON NANO, porque cuenta con un CPU ARM Cortex-A57 MPCore de 4 núcleos capaz de proporcionar 472 gigaflops de potencia, además posee una GPU NVIDIA Maxwell con 128 núcleos CUDA (capaz de ejecutar la librería de procesado de datos CUDA-X AI), y como también 4 Gb de RAM [8]. Todo ello contribuyó en el momento de realizar la predicción del ángulo de giro de una manera más eficiente, en conjunto con una cámara USB, un controlador PCA 9685 que mediante una comunicación I2C recibió el comando de la tarjeta de desarrollo JETSON-NANO para realizar un arranque a los motores traseros, así como también manipular la dirección mediante el servomotor; esto se logró gracias a las ondas PWM las cuales tienen un valor mínimo de 0 y un máximo de 0XFFFF. Entonces, si se desea obtener un arranque para adelante se debe colocar el valor 0XFFFF en un PIN PWM conectado en IN1, y un 0 en un PIN PWM conectado a IN2 del módulo L298N para poder realizar la acción. Luego, para controlar la dirección del automóvil mediante el servomotor se utilizó la librería del módulo PCA 9685 invocando la sentencia kit.servo[*] = θ, donde * significa el PIN PWM del módulo y θ el ángulo que se desea; en este proyecto se delimitó a 90° la posición central, de 0° a 90° como giro a la izquierda y finalmente de 90° a 180° como giro a la derecha. Asimismo, para seleccionar la velocidad del automóvil se utilizaron los pines ENABLE A y B del L298N con los cuales se controló la variable velocidad ingresando el porcentaje de la misma como un valor máximo del valor permitido (0XFFF); por ejemplo, si se desea una velocidad del 75% se deben realizar las operaciones definidas en (1), (2), (3) y (4).

$$0XFFFF\ (hexadecimal)\ = 65535\ \text{(decimal)} \quad (1)$$

$$valor_{velocidad} = 65535 * 0.75 = 49151 \quad (2)$$

$$49151\ (decimal) = 0XBFFF\ (hexadecimal) \quad (3)$$

$$valor_{velocidad} = 0XBFFF \quad (4)$$

A continuación, la figura 2 muestra una representación general del diagrama de bloques del proyecto propuesto. De esta manera, todo el proyecto fue controlado por módulos, los cuales son apreciados en la propia figura. Además, cada módulo tiene una función particular como es la de capturar la imagen a través de la cámara y comunicado con la tarjeta de desarrollo JETSON NANO, así como también los módulos para controlar los motores y el servomotor, y la existencia de un archivo de cabecera MAIN.PY para invocar a los módulos y así lograr realizar una conducción autónoma del automóvil a escala, según la propuesta de este trabajo.

Luego, en la figura 3, se muestra una fotografía de la implementación física del automóvil a escala que fue utilizado en este trabajo. Y, tal como se aprecia en la figura, el automóvil cuenta con la integración de la tarjeta JETSON-NANO, los módulos PCA 9658, L298N, la cámara, motores y una batería externa para alimentar el sistema, estos en conjunto forman un sistema de conducción autónoma eléctrico en el cual podemos probar diferentes redes neuronales para realizar una conducción autónoma.

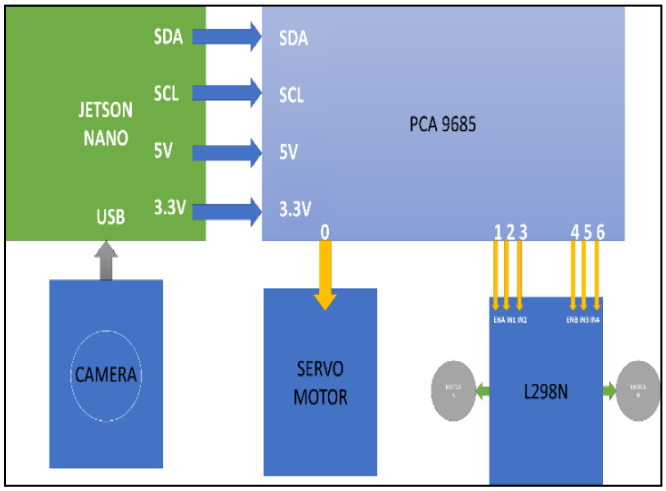

Fig. 2 Diagrama general de bloques.

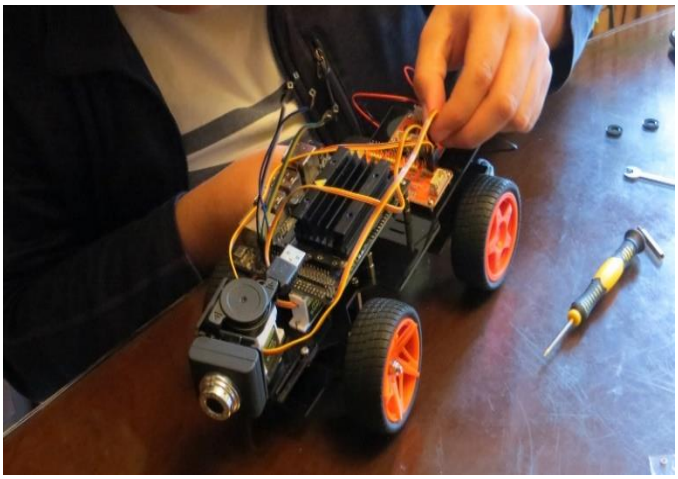

Fig. 3 Fotografía del automóvil a escala.

B. Obtención de datos y procesamiento

Para generar la data necesaria para el entrenamiento de la red neuronal artificial propuesta, se utilizaron 2 principales elementos:

- Circuito de Pruebas. Lugar donde el automóvil a escala fue entrenado para realizar una conducción y prueba autónoma, el carril donde el automóvil va a transitar está delimitado por las franjas negras como se puede ver mediante una fotografía en la figura 4 y 5

Fig. 4 Circuito 1 real para pruebas

Fig. 5 Circuito 2 real para pruebas

- Gamepad. Encargado de producir uno de los elementos necesarios para el entrenamiento (ángulo de giro) gracias a los joysticks que posee, con estos podemos generar información para poder entrenar el sistema y a su vez controlar el automóvil.

Fig. 6 Fotografía del Gamepad

Además, estos elementos fueron procesados por módulos los cuales forman parte de la tarjeta de desarrollo JETSON-NANO, y tienen las siguientes funciones:

- Módulo de Cámara. Captura la imagen de entrada que proviene de la cámara conectada a la tarjeta de desarrollo JETSON NANO. De esta manera, la captura de imágenes se realizó cada 30 mili segundos durante el trayecto del automóvil.
- Módulo JOYSTICK. Lee el valor proveniente del joystick del gamepad el cual tiene valores de -1 a 1 para el control del automóvil; además, estos son enviados hacia el módulo de control de los servomotores donde a través de una fórmula son convertidos a comandos en forma de ángulo de rotación del servomotor. Ver la expresión en (5).

$$\theta = \left(\frac{V_{JOYSTICK} + 1}{2}\right) * 180 \quad (5)$$

Donde:
θ = Ángulo deseado (de 0° a 180°)
$V_{JOYSTICK}$ = Valor dado por el joystick (de -1 a 1)

- Módulo de Motores. Controla la velocidad de los motores traseros encargados de 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.

- Módulo de Recolección. Realiza las funciones para capturar los valores de imagen y del joystick analógico del gamepad.

- Módulo MAIN_RECOLECCION. Este módulo con apoyo de los anteriores se encarga de capturar el valor del JOYSTICK y de la cámara, los cuales se almacenan en una carpeta con las imágenes. Asimismo, cada sesión de entrenamiento se almacena en un archivo .csv, en el cual se encuentra el nombre de la imagen, así como también el valor del joystick analógico.

De esta manera, tendiendo la información necesaria se procedió a la extracción de los datos obtenidos de la conducción en los circuitos 1 y 2 pasaríamos a exportar esta data hacia una Workstation y, se procedió a conocer la cantidad de datos o información disponible. Sin embargo, la base de datos resultó desbalanceada porque cuando se conduce un vehículo, este se mantiene casi siempre en una posición centrada. Por lo tanto, la figura 7 representa un gráfico de histograma donde se aprecia la cantidad de imágenes correspondientes a imágenes sin algún giro (representadas por 0), las imágenes de giro hacia la izquierda (comprendidas entre -1 a 0) y las de giro hacia la derecha (comprendidas entre 0 a 1).

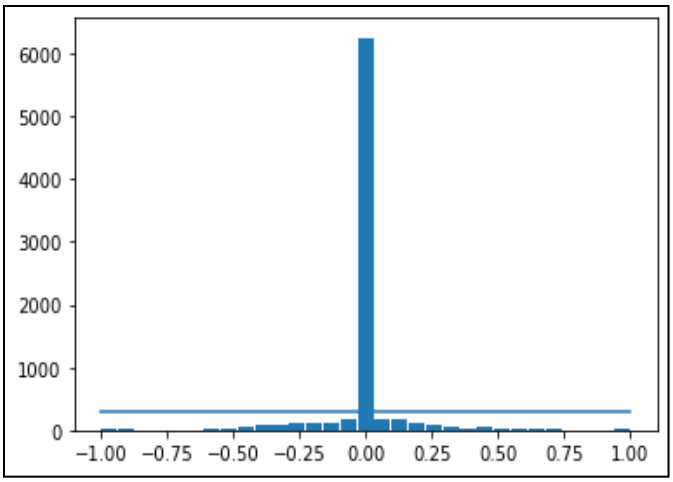

Fig. 7 Datos Desbalanceados

Por lo cual, debido a la distribución de los datos de una forma desbalanceada se presentó un problema asociado a la pérdida de tiempo durante el entrenamiento de la red neuronal, por la cantidad de datos innecesarios; de esta manera, se optó por balancear tales datos utilizando técnicas de BIG DATA, esto es importante ya que puede producir un overfitting en el momento del entrenamiento y a la misma vez retrasar el tiempo de entrenamiento de la red neuronal, para eso se colocó un número de muestras máximas para la data con un ángulo de 0° y se obtuvo los resultados mostrados en la figura 8.

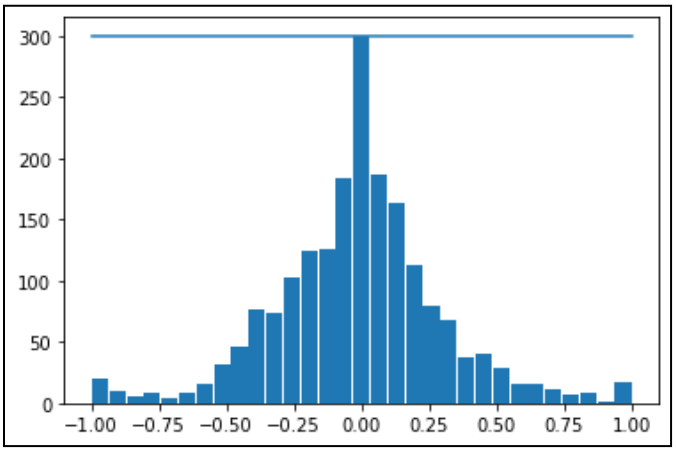

Fig. 8 Datos balanceados

C. Procesamiento de las imágenes obtenidas y entrenamiento

Para mejorar los resultados del entrenamiento con las imágenes obtenidas se tuvo que realizar un procesamiento de estas, para ello al principio del proyecto se utilizó la librería OPEN-CV [14], pero esta librería 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, se enumeran a continuación.

- Ajuste de brillo
  Usado para poder simular diferentes condiciones de brillo en el entorno real.
- Zoom
  La imagen obtenida posee elementos los cuales no son necesarios para realizar la conducción autónoma del vehículo, por lo cual solo se requirió una imagen del carril donde se desplaza el automóvil.
- Giro de imagen
  En la base de datos obtenida la imagen puede tener una tendencia a apuntar a un lado del automóvil; por lo cual, el giro de la imagen ayudó a diversificar de la mejor manera el entrenamiento.

Luego de las mejoras obtenemos los siguientes resultados como se pueden observar en la figura 9.

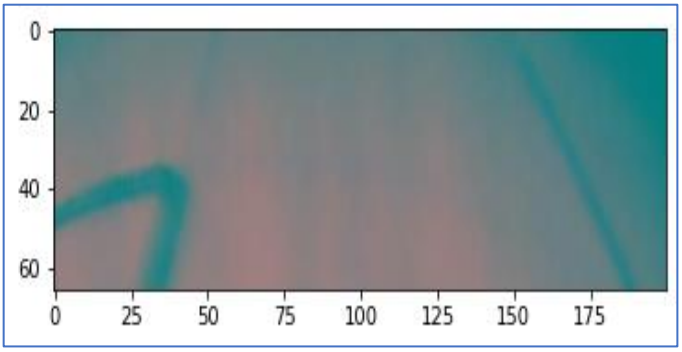

Fig. 9 Imagen con Mejoras

Una vez procesada la información se continuó con el entrenamiento de la red neuronal utilizando un total de 4512 imágenes entre los 2 circuitos mostrados en las figuras 4 y 5 divididas en dos grupos. El primero con 3609 imágenes utilizadas para el entrenamiento propiamente dicho, y otras 902 imágenes para la validación; asimismo, se tuvo un apoyo del framework TENSORFLOW-GPU para hacer uso de una GPU (RTX 2080 SUPER) y así acelerar el proceso de entrenamiento. En promedio, se utilizaron de 50 a 63 minutos de entrenamiento con 8 épocas, y con 300 pasos por cada una. Al finalizar el entrenamiento guardamos el modelo obtenido en un archivo con extensión. pb en el cual encontramos la red neuronal y los pesos del modelo.

D. Optimización con Tensor-RT

Con el modelo entrenado y guardado, se tuvo la opción de ejecutar una mejora con el optimizador TENSOR-RT [9]. Por lo cual, para ejecutar este optimizador primero se tuvo que

realizar una función en la cual se almacenó toda la configuración necesaria para la optimización del modelo, como por ejemplo el tipo de datos de precisión (FP32, FP16, INT8) utilizado para realizar las operaciones del tensor; luego, se creó otra función para realizar la partición de la red neuronal en pequeñas redes, las cuales dependiendo de su compatibilidad con el TENSOR-RT¨[14] se fusionaron en un bloque llamado "TRTEngineOP", con lo cual se logró mejorar el uso de memoria GPU dependiendo del tipo de dato de precisión que se seleccionó, y como también mejorar el performance de la red neuronal a la hora de ser ejecutada en el sistema embebido. Finalmente se procedió con el almacenamiento de la red neuronal optimizada en un archivo con extensión. pb el cual se ejecutó en la tarjeta de desarrollo JETSON-NANO [12].

III. Resultados

En cuanto a los resultados obtenidos, la figura 7 muestra el desempeño del entrenamiento de la red neuronal usando modelos optimizados con KERAS-TUNER [13] y su comparativa con el modelo original, donde se observa también la pérdida del modelo por época, siendo 8 épocas el total y 300 pasos por cada una de estas.

Además, según lo comparado entre los mejores modelos obtenidos, se observa que el modelo "MODEL_KERAS1" posee el valor de pérdida más bajo en comparación con los demás modelos, dicho valor de resultado fue igual a 0.03725747. Por lo cual, este modelo contó con una cantidad de filtros entre 56 a 216 y una tasa de aprendizaje lr igual a 0.00001; entonces de los resultados obtenidos en esta sección se puede concluir que el uso de un sintonizador de hiperparametros 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 hiperparametros a la búsqueda lo que generaría mejores resultados a costa de un mayor tiempo de búsqueda.

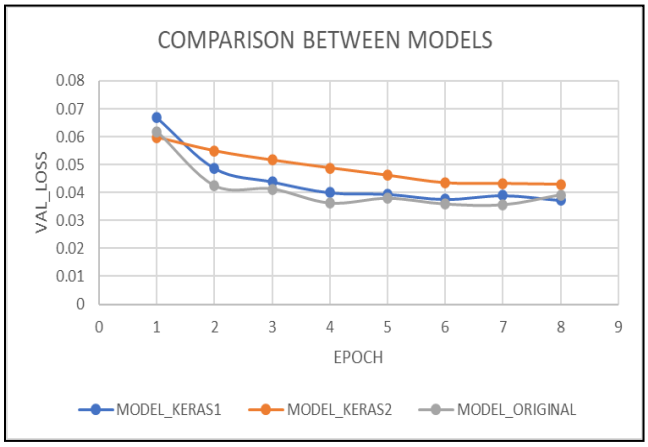

Fig. 10 Comparativa de los modelos, sin KERAS TUNER y con hiper-parámetros sintonizados en función de pérdida por época.

Así como también, se realizó una comparación del modelo optimado con KERAS-TUNER [11] usando TENSOR-RT [14] y sin usar el optimizador; por lo cual, los resultados logrados indican el tiempo que demoran en realizar la inferencia a la hora de efectuar las pruebas.

Según lo obtenido en las pruebas de inferencia sintéticas, se determina que el modelo optimizado usando FP16 posee un tiempo promedio de inferencia de 0.641455212489 segundos, mientras que la prueba del modelo original sin optimizaciones tuvo un tiempo de 1.4421267935 segundo lo que demuestra la mejora de tiempos usando un optimizador para sintonizar los hiper-parametros la red y otro para mejorar el performance de la CNN al momento de probarla lo que permitió obtener una mejora del 2.24 con respecto al otro modelo; la optimización con TENSOR-RT nos permite obtener una mejora sustancial para el tipo de proyecto realizado, cabe recalcar que una respuesta rápida y certera es esencial en este proyecto ya que deseamos que el automóvil actúe rápido en circunstancias adversas.

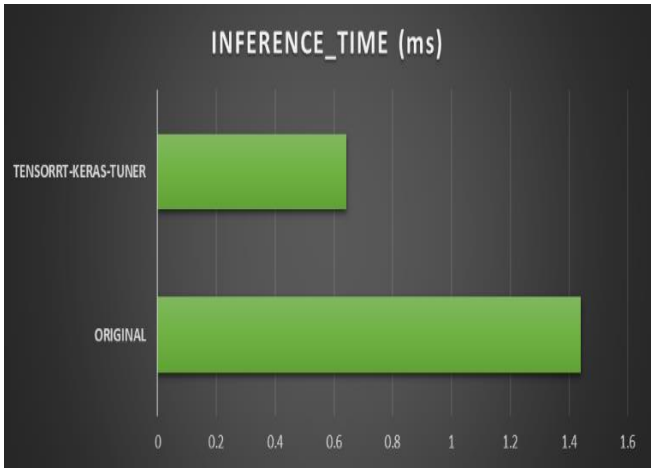

Fig. 11 Comparativa del modelo optimizado usando TENSOR-RT y sin optimizar.

Además de eso se hizo una comparativa del uso de CPU y RAM de la Jetson Nano al momento de ejecutar las pruebas, los resultados arrojaron que el modelo sin optimizar consume más memoria RAM, mientras que el modelo optimizado con KERAS-TUNER [13] y TENSORRT [14] posee menos uso de memoria gracias a que el optimizar la CNN se cambió de tipo de dato (de FP32 A FP16) lo que permitió al sistema a consumir menos memoria y consumir menos recursos, esto se ve reflejado en la figura 13 donde observaremos las diferencias.

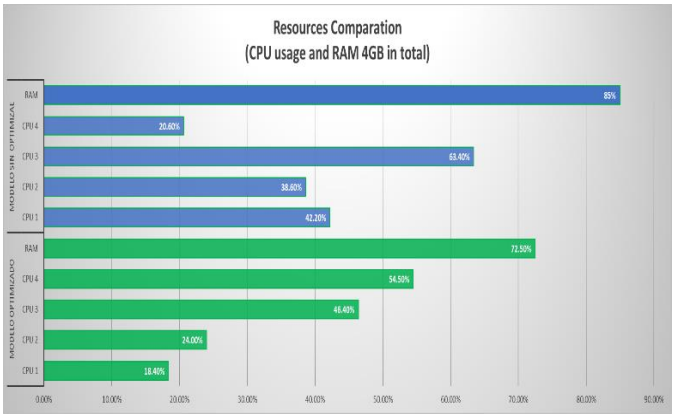

Fig. 12 Comparativa de uso de CPU y RAM entre los 2 modelos

De lo obtenido podemos ver que el uso de los núcleos de la JETSON-NANO [12] es menor en el modelo optimizado como también podemos observar que el uso de RAM es de 10% menor comparándolo con el modelo sin optimizar, aunque esto varía según las circunstancias donde se están realizando las pruebas, en este caso la velocidad seteada fue de 60%, además el uso de CPU es similar en las 2 pruebas debido a que el código netamente corre en la GPU del sistema embebido.

Por último, se puso a prueba el modelo con todas las optimizaciones y el original en el entorno real, y se midió la autonomía según la velocidad a la que funcionó. Sin embargo, se necesitó de intervención humana para no salir del carril, para esto se utilizó la fórmula de [5] representada en (6), pero teniendo en cuenta que para realizar una intervención con el analógico se demoró un aproximado de 1 segundo.

$$Autonomy = \left(1 - \frac{\#Number\ of\ interventions * 1}{Elapsed\ Tine}\right) * 100 \qquad (6)$$

A continuación, la Tabla I muestra los resultados alcanzados para las tres pruebas realizadas en el circuito 1. Asimismo, tales resultados corresponden al porcentaje de velocidad y autonomía, así como al tiempo en segundos y la cantidad de intervenciones del modelo optimizado.

TABLA I
MÉTRICAS DE VELOCIDAD Y AUTONOMÍA DEL MODELO EN CIRCUITO 1

	VELOCITY (%)	TIME (SEC)	INTERVEN-TIONS	AUTONOMY (%)
TEST 1	50 %	20.23	0	100
TEST 2	70 %	15.78	2	87.32
TEST 3	100 %	12.42	4	67.79

De los resultados obtenidos podemos ver una relación directa entre pérdida de autonomía e incremento de velocidad del automóvil, como podemos ver en la prueba 3 tenemos una pérdida de casi 33% cuando colocamos una velocidad de 100% en lo motores traseros encargados del arranque, en la figura 14 podremos ver el automóvil recorriendo el circuito 1.

Fig. 13 Automóvil a escala en Circuito 1

Ahora con la misma red se probó en otro entorno el cual tiene mayor longitud, los resultados obtenidos serán presentados en la Tabla II.

TABLA II
MÉTRICAS DE VELOCIDAD Y AUTONOMÍA DEL MODELO EN CIRCUITO 2

	VELOCITY (%)	TIME (SEC)	INTERVEN-TIONS	AUTONOMY (%)
TEST 1	50 %	43.23	1	97.68
TEST 2	70 %	32.84	3	90.86
TEST 3	100 %	27.87	3	89.23

De los resultados obtenidos podemos ver una relación directa entre pérdida de autonomía e incremento de velocidad del automóvil al igual que en las pruebas del circuito 1, sin embargo, el automóvil respondió de mejor manera a máxima velocidad que en la prueba anterior esto se ve evidenciado en los resultados del TEST 3, en la figura 15 podremos ver el automóvil recorriendo el circuito 2.

Fig. 14 Automóvil a escala en Circuito 2

En las pruebas de autonomía del automóvil observamos que se tiene menos autonomía en el primer circuito, dado que en la prueba con los motores funcionando al 100% de su capacidad obtenemos un 67.79% de autonomía, mientras que en el circuito 2 se tiene 89.23%. Esto se debe a que se obtuvo más datos del segundo circuito en la recopilación de datos, debido a que este tiene mayor dimensión que el primero; una solución para este problema sería tener la misma cantidad de datos obtenidos en los dos circuitos para que pueda desempeñarse de mejor manera en ambos.

Con lo obtenido , se concluye que la tarjeta de desarrollo JETSON-NANO soporta de buena manera el propósito del trabajo; no obstante, si existiera un interés de implementar dicho trabajo en un automóvil a escala real, se necesitaría de un hardware más robusto para tal ejecución, otro aspecto importante es la data de entrenamiento la cual si es tratada correctamente con algoritmos de BIG DATA nos daría un mejor resultado en el momento del entrenamiento como también a la hora de ejecutar la red neuronal, además se comprueba la importancia de usar optimizadores los cuales ayudan enormemente a la hora de ejecutar el programa en un sistema embebido, además queda comprobada que el uso de una GPU es esencial para este tipo de trabajos.

TRABAJOS FUTUROS

Y, como futuras mejoras, se tiene planeado el uso de un controlador digital de señales Dspic33fj128mc802 para incorporar un algoritmo de control PID o de lógica difusa que permita controlar la velocidad de los motores traseros y el de la dirección; además, se tiene planificado el uso de una comunicación tipo UART entre la tarjeta de desarrollo JETSON-NANO [12] y la tarjera de control de motores con el Dspic33fj128mc802. Y como también, utilizar una red neuronal convolucional recurrente para la obtención del ángulo de giro ya que usando una celda LSTM es posible predecir el siguiente estado en función de uno anterior a este.

AGRADECIMIENTOS

Agradezco al profesor Pedro Huamani Navarrete por su gran apoyo en la redacción de este artículo, por los alcances dados y su ayuda incondicional a este proyecto; asimismo, mis agradecimientos van hacia la Universidad Ricardo Palma por dotarme conocimientos científicos a través de su plana docente.

VIDEOS DEL AUTOMOVIL EN EJECUCCIÓN

En esta sección se hará presente links de videos tomados del vehículo en ejecución en los circuitos 1 y 2.

PRUEBA 1 y 2:

- https://www.youtube.com/watch?v=4OtW9RFMEA0
- https://www.youtube.com/watch?v=1JFG7a6US3E

REFERENCIAS

[1] Self-Driving Vehicles Could Struggle To Eliminate Most Crashes - Fleet Management Weekly", Fleet Management Weekly, 2020. [Online]. Available: https://www.fleetmanagementweekly.com/self-driving-vehicles-could-struggle-to-eliminate-most-crashes/#:~:text=According%20to%20a%20new%20study,drive%20too%20much%20like%20people. [Accessed: 20- Dec- 2020].

[2] Electric Cars, Solar & Clean Energy | Tesla", Tesla, 2021. [Online]. Available: https://www.tesla.com/. [Accessed: 20- Dec- 2020].

[3] J. Donahue et al., "Long-term Recurrent Convolutional Networks for Visual Recognition and Description", arXiv.org, 2021. [Online]. Available: https://arxiv.org/abs/1411.4389. [Accessed: 20- Dec- 2020].

[4] Comparison of Machine Learning Algorithm's on Self-Driving Car Navigation using Nvidia Jetson Nano", Ieeexplore.ieee.org, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9158311. [Accessed: 07- Jan- 2021].

[5] M. Bojarski et al., "End to End Learning for Self-Driving Cars", arXiv.org, 2016. [Online]. Available: https://arxiv.org/abs/1604.07316. [Accessed: 07- Jan- 2021].

[6] M. Bechtel, E. McEllhiney, M. Kim and H. Yun, "DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car", arXiv.org, 2017. [Online]. Available: https://arxiv.org/abs/1712.08644. [Accessed: 07- Jan- 2021].

[7] R. Pi, "Teach, Learn, and Make with Raspberry Pi", Raspberry Pi. [Online]. Available: https://www.raspberrypi.org/. [Accessed: 18- Jun- 2021].

[8] Obstacle classification and detection for vision-based navigation for autonomous driving", Ieeexplore.ieee.org, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8126154?denied=. [Accessed: 07 -Jun- 2021].

[9] Nvidia.com, 2015. [Online]. Available: https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf. [Accessed: 08- Jun- 2021].

[10] J. Wei, "AlexNet: The Architecture that Challenged CNNs", Medium, 2020. [Online]. Available: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951. [Accessed: 20- Apr- 2021].

[11] M. Heller, "What is CUDA? Parallel programming for GPUs", InfoWorld, 2018. [Online]. Available: https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html. [Accessed: 03- Apr- 2021].

[12] Jetson Nano Developer Kit", NVIDIA Developer, 2021. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Accessed: 16- Jul- 2020].

[13] K. Team, "Keras documentation: KerasTuner", Keras.io. [Online] A vailable: h ttps://keras.io/keras_tuner/. [Accessed: 10- Jan- 2021].

[14] Developer Guide: NVIDIA Deep Learning TensorRT Documentation" Docs.nvidia.com, 2021. [Online].