UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

SISTEMA DE CONTROL PARA DISTRIBUCIÓN DE COMBUSTIBLES MEDIANTE LA IDENTIFICACIÓN DE PLACAS EN VEHÍCULOS UTILIZANDO VISIÓN ARTIFICIAL

Trabajo de titulación previo a la obtención del
Título de Ingeniero en Electrónica y Automatización

Autores: Rony Andrés Anasi Nasimba
German Andrés Martínez Arellano
Tutor: Carmen Johanna Celi Sánchez

Quito – Ecuador
2023

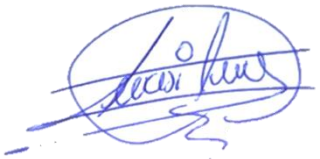
CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Rony Andrés Anasi Nasimba con documento de identificación N° 1722663760 y German Andrés Martínez Arellano con documento de identificación N° 0401765748 manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 10 de marzo del año 2023.

Atentamente,



Rony Andrés Anasi Nasimba
1722663760

German Andrés Martínez Arellano
0401765748

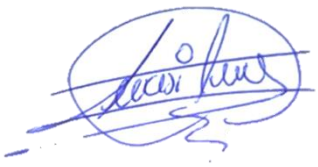
CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, Rony Andrés Anasi Nasimba con documento de identificación N° 1722663760 y German Andrés Martínez Arellano con documento de identificación N° 0401765748, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico: "Sistema de control para distribución de combustibles mediante la identificación de placas en vehículos utilizando visión artificial", el cual ha sido desarrollado para optar por el título de: Ingenieros en Electrónica y Automatización, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, 10 de marzo de 2023

Atentamente,



Rony Andrés Anasi Nasimba
1722663760

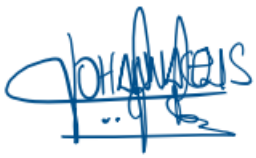
German Andrés Martínez Arellano
0401765748

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Carmen Johanna Celi Sánchez con documento de identificación N° 1717437808 docente de la Universidad Politécnica Salesiana Sede Quito, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: SISTEMA DE CONTROL PARA DISTRIBUCIÓN DE COMBUSTIBLES MEDIANTE LA IDENTIFICACIÓN DE PLACAS EN VEHÍCULOS UTILIZANDO VISIÓN ARTIFICIAL, realizado por: Anasi Nasimba Rony Andrés con documento de identificación N° 1722663760 y German Andrés Martínez Arellano con documento de identificación N° 0401765748 obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 10 de marzo de 2023

Atentamente,



Ing. Carmen Johanna Celi Sánchez Mgtr.
1717437808

Tabla de contenido

Índice de Tablas

Índice de Figuras

RESUMEN

El reconocimiento de placas de vehículos por sus siglas en ingles ALPR es un avance importante en el campo de la visión artificial, para identificar y leer los caracteres de una matrícula, las aplicaciones en base a esta tecnología se aplican en diversos campos como la seguridad y control.

El presente trabajo desarrolla un sistema de identificación y control de vehículos en el abastecimiento de combustible por medio de la tarjeta de desarrollo Jetson Nano 4GB con el uso de redes neuronales convolucionales y visión artificial, para la creación de la red neuronal se apoyó en la CNN Mobilnet v2 modificándola y reentrenándola en Google Colab disminuyendo la complejidad de la red optimizando los procesos de inferencia para posteriormente enviar el modelo a la tarjeta Jetson Nano que contiene una serie de librerías propias de Nvidia como es Jetson Utilies el cual está integrado por TensorRT, CUDA, VPI entre otras. El proceso de control se realiza creando archivos con extensión .CVS en el cual almacena la placa del vehículo en formato texto para enviar la información a un servidor que se encarga de comprobar si existe repitencia y de haberlo enviar una alerta vía correo electrónico.

La implementación del sistema en ambientes controlados se comprobó su eficacia con resultados mayores al 90% en la detección y control de vehículos.

ABSTRACT

The recognition of vehicle plates by its acronym in English ALPR is an important advance in the field of artificial vision, to identify and read the characters of a license plate, the applications based on this technology are applied in various fields such as security and control.

The present work develops a system of identification and control of vehicles in the fuel supply by means of the Jetson Nano 4GB development card with the use of convolutional neural networks and artificial vision, for the creation of the neural network was supported by CNN Mobilnet v2 modifying and retraining it in Google Colab reducing the complexity of the network optimizing the inference processes to later send the model to the Jetson Nano card that contains a series of Nvidia's own libraries such as Jetson Utilies which is integrated by TensorRT, CUDA , VPI among others. The control process is carried out by creating files with a .CVS extension in which the vehicle license plate is stored in text format to send the information to a server that is in charge of checking if there is repetition and, if there is, sending an alert via email.

The implementation of the system in controlled environments, its effectiveness was verified with results greater than 90% in the detection and control of vehicles

INTRODUCCIÓN

La visión artificial analiza imágenes o interpreta escenas, es el proceso de extraer información del mundo real a partir de imágenes utilizando ordenadores como herramientas. Los componentes principales de un sistema de visión artificial son un sensor de imagen y un digitalizador. Los sensores de imagen son dispositivos físicos sensibles en el espectro de energía electromagnética que generan.

El presente proyecto pretende realizar un sistema de identificación y control de combustible mediante el reconocimiento de placas de vehículos en ambientes controlados (luminosidad, distancia entre la cámara y el vehículo), proporcionando imágenes de entrada de calidad para que la red neuronal pueda procesar la información de forma correcta.

Para desarrollar este proyecto, la documentación se desglosa de la siguiente manera.

El Capítulo 1, analiza el estado de la tecnología, antecedentes del proyecto técnico, en el cual se detalla el problema de estudio, justificaciones de la implementación y objetivos

El Capítulo 2 describe el marco de referencia, que es la información recopilada para el desarrollo del sistema de reconocimiento y control de las placas vehiculares.

El Capítulo 3 describe el diseño e implementación de un sistema de reconocimiento de matrículas que utiliza redes neuronales artificiales y computadoras de inteligencia artificial para el reconocimiento.

El Capítulo 4 describe las pruebas y los resultados obtenidos del reconocimiento y control de matrículas. Finalmente se presentan las conclusiones y recomendaciones obtenidas durante la interacción con la aplicación.

CAPÍTULO 1

ANTECEDENTES

1.1. Planteamiento del problema

El contrabando de combustibles provoca pérdidas millonarias al estado ecuatoriano, desde el 22 de octubre del 2021 los precios están congelados y se establece Extra y Ecopaís en $2,55 y el Diesel en $1,90. El Diésel mantiene una diferencia por galón de alrededor $0,70 centavos (Andrade, 2021).

Un reporte de la Policía Nacional informa que en el segundo trimestre del 2021 se efectuaron 33 operativos en Carchi dejando 20 detenidos y 4850 litros de combustibles decomisados. Se ha detectado una modalidad de contrabando denominada **"Hormiga"**, es decir el paso de combustible desde Tulcán a Ipiales en pocas cantidades, pero varias veces al día (Ecomex360, 2019).

La provincia del Carchi cuenta con 12 estaciones de servicio para la distribución del hidrocarburo, 10 pertenecientes a Petroecuador, la normativa vigente en esta provincia desde septiembre del 2019 permite cargar de combustible una sola vez al día a los vehículos de placas ecuatorianas (Benalcazar, 2020). Para su control se implementa un software que lleva un registro con capturas y fotografías del vehículo que accedieron al combustible, mientras tanto los despachadores validan el número de placas y cedula del conductor en el sistema. Sin embargo, este sistema es ambiguo y poco eficiente si bien se cuenta con un "sistema funcionando en tiempo real" las fotografías no son procesadas en ese instante, el parámetro principal es el ingreso de la placa (Endara & Cabezas, 2021).

Anagnostopoulos, (Anagnostopoulos, Anagnostopoulos, Loumos, & Kayafas, 2006) Plantea un algoritmo que identifica matriculas de distintos tamaños y posiciones. También su método es capaz de identificar más de una matrícula a partir de una sola imagen. Su algoritmo se cimenta en una "nueva técnica de segmentación de imágenes adaptables" y una red neuronal probabilística (PNN). Obteniendo un 96,5% de precisión en la detección de la matricula y un 89,1% de reconocimiento.

1.2. Justificación

Con el aumento del parque automotor, los sistemas de reconocimientos automático de matrículas (ALPR) se vuelve importantes en maniobras de aplicación de

la ley, las cualidades de estos sistemas son variadas por su versatilidad, por ejemplo, ejecutarse en dispositivos portátiles o servidores en la nube u operar en condiciones adversas.

Un sistema ALPR captura una imagen o segmento de un video como una entrada al sistema, si en la imagen existe integrados por una cámara que detecta el entorno y un software que realiza el procesamiento (Shashirangana, Padmasiri, Meedeniya, & Perera, 2021).

En este trabajo explora una posible solución de distribución diaria del hidrocarburo en las estaciones de servicio de la provincia del Carchi, utilizando un sistema embebido capaz de procesar y tratar las imágenes capturadas utilizando algoritmos de Visión Artificial con parámetros de precisión y altas velocidades, una cámara capaz de capturar el vehículo con su matrícula a diferentes distancias y en condiciones de visibilidad variables. El sistema se encuentra conectado a internet a través de Wifi donde envía al servidor únicamente la información de interés en formato texto para una carga rápida. Si un vehículo intenta cargar nuevamente combustible en otra gasolinera el sistema lo detecta alertando al operario con un correo electrónico negando así el combustible al vehículo cumpliendo con la normativa vigente de la Agencia de Regulación y Control Hidrocarburífero (ARCH) controlando la fuga de combustible hacia el país de Colombia que provoca pérdidas millonarias al estado

1.3. Objetivos

1.3.1. Objetivo General.

- Implementar un sistema de detección de placas del vehículo mediante Visión e Inteligencia Artificial utilizando software libre para el aviso de distribución de combustible diario.

1.3.2. Objetivos Específicos.

- Investigar el estado de la tecnología en hardware y software para la selección de materiales, equipos y programas que serán necesarios para la solución, a través de bases de datos y artículos científicos.

- Desarrollar un sistema electrónico utilizando una tarjeta de desarrollo y sensores de visión para la integración de software.
- Desarrollar un algoritmo de visión e inteligencia artificial para la detección de las placas del vehículo y la comunicación a la nube para generar un sistema de avisos utilizando un software libre.
- Realizar las pruebas de funcionamiento del prototipo para la validación de los resultados bajo un entorno controlado en un prototipo de prueba.

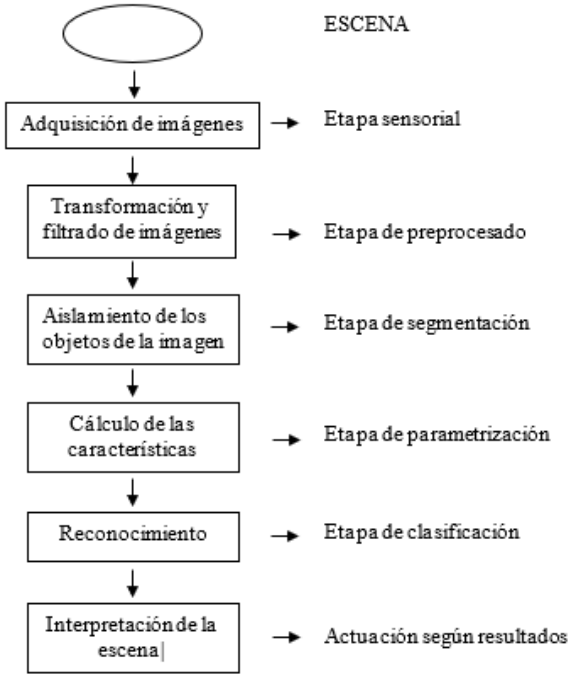
CAPÍTULO 2

MARCO TEÓRICO

2.1. Visión Artificial

Es una variedad de técnicas de procesamiento de imágenes que permiten que una computadora "vea" una imagen. Estas técnicas incluyen Deep Learning, reconocimiento de patrones, procesamiento de imagen digital, análisis de contenido y análisis de objetos. Estas técnicas se usan para identificar objetos, descubrir patrones y realizar análisis más profundos de las imágenes (Rosado, Figueras, Planas, & Reverter, 2015).

Figura 1:Diagrama de bloques de un sistema de visión artificial.



Elaborado por Anasi. R & Martínez. G

En los sistemas de visión artificial cuenta con sistemas de procesamiento que toman imágenes y realizan funciones dependiendo del tipo de análisis a realizar.

Las mejores aplicaciones para el procesamiento de imágenes incluyen:

- **El análisis de objetos:** Es el proceso que ayuda a comprender el entorno para identificar y describir el objeto de una imagen.

- **El reconocimiento de patrones:** es el procesamiento de visión por computadora la cual usa algoritmos en un entorno controlado con múltiples iteraciones en el procesamiento de imágenes (Vega, 2019).
- **Reconstrucción:** Mediante la inteligencia artificial se puede procesar un conjunto de imágenes que contengan el objeto deseado para detectar.
- **El análisis de contenidos:** Es el proceso de utilizar técnicas de aprendizaje automático para extraer información de imágenes y videos la cual permite detectar características específicas (Álvarez Durán, 2014).
- **La detección de objetos:** Consiste en localizar y clasificar objetos en una imagen con la que se puede realizar con una variedad de algoritmos para una determinada información (Romero Gómez, 2021).

2.2. Edge Computing

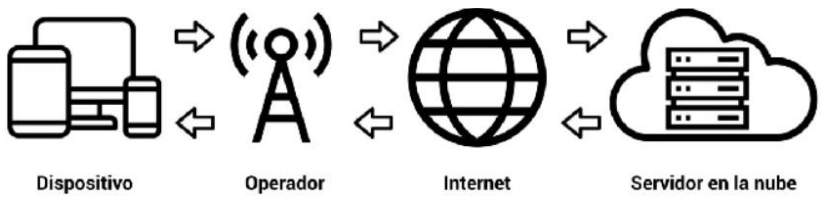
Edge Computing se basa en mover parte de la carga informática al borde de la red para utilizar la potencia informática actualmente infrautilizada de los nodos finales, como estaciones base, enrutadores y conmutadores.

Para entender Edge Computing se necesita comprender que es Cloud Computing, cada vez que se accede a un correo electrónico, redes sociales, se utiliza un servicio en la nube, lo cual el uso de la nube consiste en interactuar con información que se encuentre en un servidor y acceder desde internet. (Satyanarayanan, 2017).

El procedimiento es el siguiente:

El dispositivo se conecta a Internet, el Proveedor responsable de transferir datos desde el dispositivo al servidor de destino, utilizando la dirección IP (por ejemplo, Gmail, Dropbox) para determinar el sitio al que se enviará la información, y luego el servidor procesa la solicitud y la envía al dispositivo (Rebato, 2022).

Figura 2: Arquitectura de Cloud Computing.

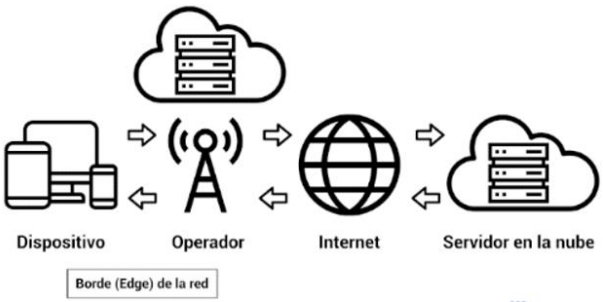


Fuente: (Rebato, 2022)

Ventajas del Edge Computing

La mejor definición para entender Edge Computing es “llevar la potencia informática lo más cerca posible del lugar de donde se generan los datos” para visualizar las funciones del servidor y habilitar la potencia del procesamiento con esto la velocidad se dispara, la latencia se reduce y las posibilidades se multiplican. De modo que se obtiene calidad, seguridad, baja latencia del procesamiento de la PC, flexibilidad, disponibilidad, escalabilidad, y eficiencia que ofrece la nube (Varghese, 2016).

Figura 3: Arquitectura de Edge Computing.



Fuente: (Rebato, 2022)

2.3. Edge Computing y Machine Learning

Los algoritmos de Machine Learning suelen funcionar “entrenando” a la IA con miles de imágenes. Una vez que se crea el modelo, generalmente se carga en un servidor en la nube, con Edge Computing mejora significativamente este proceso. En cualquier caso, en lugar de ir a un servidor en la nube, se crear una copia (virtual o reducida) del modelo de Machine Learning que se encuentra en el borde de la red. Aquí es donde realmente se generan los datos (Rebato, 2022).

Figura 4: Arquitectura Cloud y Edge Computing.

Fuente: (Rebato, 2022)

2.4. Edge Computing y Deep Learning con NVIDIA Jetson Nano

El Deep Learning es una rama de aprendizaje automático que se enfoca en resolver problemas complejos utilizando redes neuronales profundas que han demostrado ser adecuadas para los sistemas de visión artificial (Rentana Ribeiro, 2020).
El proceso de Deep Learning avanzado generalmente se divide en dos etapas:

- **Entrenamiento:** las imágenes indexadas junto con las etiquetas se envían a la red neuronal para que la red reconozca los atributos de todas las imágenes con la misma etiqueta
- **Inferencia:** el proceso de alimentar imágenes no indexadas a una red neuronal para determinar individualmente en que categoría encaja.

Los procesos de entrenamiento e inferencia necesitan de hardware y software especializado para realizar las tareas porque son computacionalmente costosos. El entrenamiento de las redes neuronales generalmente se realiza desde cero (lo que puede llevar horas, días) en grandes centros de datos equipados con GPU.

Sin embargo, la operación de inferencia (clasifica las imágenes en una red entrenada) es relativamente menos exigente y Edge Computing juega un papel en este proceso de inferencia. Edge Computing se refiere al procesamiento de datos localmente sin conexión a un centro de datos externo (Rentana Ribeiro, 2020).

2.5. Redes Neuronales

Utiliza un aprendizaje controlado y una variante de perceptrón multicapa para tomar imágenes de entrada y recuperar la percepción de la imagen, es similar al campo de visión del ojo humano. Puede describirse simplemente como una red neuronal, lo que le permite constar de múltiples neuronas y expresar informáticamente grandes patrones con una cantidad relativamente pequeña de parámetros (Bonilla Carrión, 2020).

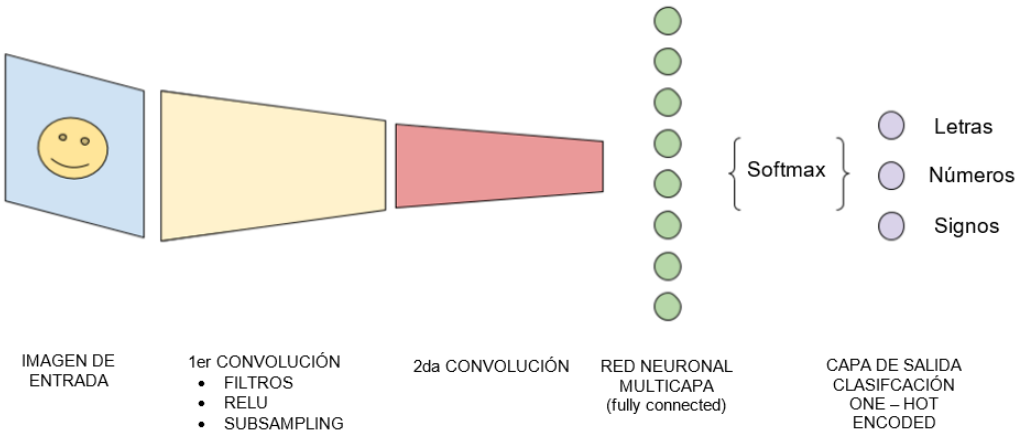
2.5.1. Redes Neuronales Convolucionales

Una red neuronal convolucional, cada neurona procesa datos solo de su región de análisis y los organiza para presentar el panorama general. También, tanto los sistemas

de visión como las redes neuronales convolucionales presentan una jerarquía de capas para extraer más y más características a lo largo del tiempo (Bonilla Carrión, 2020).

En otras palabras, las primeras capas de la red neuronal pueden identificar caracteres sencillos como líneas, bordes a medida que las capas se vuelven más profundas, las capas tienen la capacidad de reconocer objetos complejos como personas, rostros (Domínguez Pavón, 2019).

Figura 5: Arquitectura de una red neuronal convolucional



Elaborado por Anasi. R & Martínez. G

2.5.2. MobilenetV2

Se basa en convoluciones separables en profundidad, introducido para aplicaciones de imágenes integradas y móviles que usan modelos previamente entrenados evita la sobrecarga computacional innecesaria y utiliza los pesos asignados sin perder las funciones aprendidas previamente. MobileNet son básicamente una clase de redes neuronales convolucionales profundas y livianas, mucho más pequeñas y poderosas que muchos otros modelos populares, que utilizan las siguientes capas y funciones (GHOURY, SUNGUR, & DURDU, 2019).

- **Capa Convolucional.** Combina matemáticamente dos funciones para obtener una tercera función. Uno de los cuales es la matriz de imagen de la entrada A y el otro es el núcleo de convolución B que da la salida C. Funciona con

un mecanismo que extraer características de una imagen expresada en la siguiente ecuación (Preeti, y otros, 2021).

$$C(T) = (A * B)(X) = \int_{-\infty}^{\infty} A(T) * B(T - x)dT \qquad (1)$$

- **Capa de agrupación:** La aplicación de operaciones de grupo puede reducir el tamaño de la matriz de entrada sin perder muchas características, lo que acelera el cálculo.
- **Capa de abandono:** Esto reduce la redundancia de páginas que puede ocurrir durante el entrenamiento al eliminar aleatoriamente las neuronas sesgadas del modelo. Estas neuronas pueden formar parte tanto de capas ocultas como de capas visibles.
- **Capa no lineal:** Estas capas suelen seguir capas convolucionales. Las funciones no lineales más utilizadas tienen varios tipos de unidades lineales.

$$\boldsymbol{Relu}: f(x) = \max(0, x) \qquad (2)$$

$$\boldsymbol{Leaky\ Relu}: f(x) = \max(0.1x, x) \qquad (3)$$

$$\boldsymbol{ELU}: f(x) = \begin{cases} x\ x \geq 0 \\ a(e^x - 1)x < 0 \end{cases} \qquad (4)$$

- **Capa totalmente conectada:** Se agregan al modelo y se conectan completamente a la capa de activación. Estas clases ayudan a clasificar la imagen dada en un clasificador multiclase o binario.
- **Cuellos de botellas lineales:** Las funciones de activación no lineal como ReLU se pueden aplicar a las redes neuronales para eliminar fácilmente varias diferencias. Esto le permite construir redes neuronales multicapa (Preeti, y otros, 2021).

2.6. Clasificación de imágenes y Detección de objetos

En la visión artificial, la clasificación de imágenes toma una imagen y predice objetos en la imagen, mientras que la detección de objetos no solo adivina, sino que también ubica en función de un cuadro delimitador (ArcGIS API for Python, 2021)

Las imágenes son capturadas por sensores dentro de la cámara. Estos sensores son elementos sensibles a la luz que ajustan las señales eléctricas en función de la intensidad de la luz visible (Garcia Santillan, 2008).

Figura 6: Diferencia entre clasificación y detección de objetos

Fuente: (ArcGIS API for Python, 2021)

Para ilustrar, suponiendo que una imagen contiene como máximo una clase y un objeto, la salida del modelo de detección de objetos debe consistir en:

- Probabilidad de objeto
- Altura del cuadrado delimitador
- Anchura del cuadrado delimitador
- Coordenada horizontal del punto central del cuadro delimitador
- Coordenada vertical del punto central del cuadro delimitador

2.7. Detector de Disparo Único Single-Shot-Detection

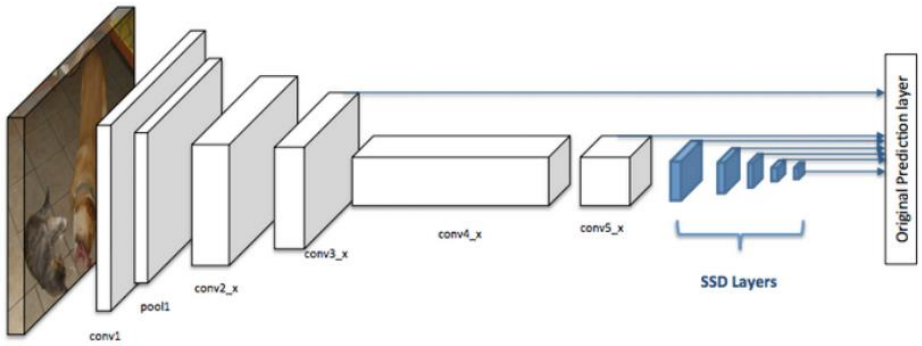
Su función principal es un buen equilibrio ente velocidad y precisión, SSD funciona realizado una red convolucional en la imagen de entrada después de calcular el mapa de características, y luego realiza un pequeño 3x3. Se utiliza un núcleo convolucional para estimar el cuadrado delimitador y probabilidades de clasificación en SSD. Después de varias capas de plegado, cualquiera puede adivinar quien tiene la ventaja y la capacidad de percibir objetos a diferentes escalas. SSD tiene 2 componentes:

- **RED TRONCAL:** Una red de clasificación de imágenes que normalmente se entrena previamente como un extractor de características. Por lo general, una red

como RESNET se entrena en ImageNet y la última capa de clasificación agregada se elimina por completo. Esto deja con una red neuronal profunda que puede extraer la semántica de las imágenes de entrada (ArcGIS API for Python, s.f.).

- **CABEZAL SSD:** Se trata simplemente de una o mas capas convolucionales agregadas a la red troncal, que genera un cuadro delimitador y una clase de entidad interpretada como la ubicación espacial de la última activación.

Figura 7: Arquitectura de una red neuronal convolucional con un detector SSD.

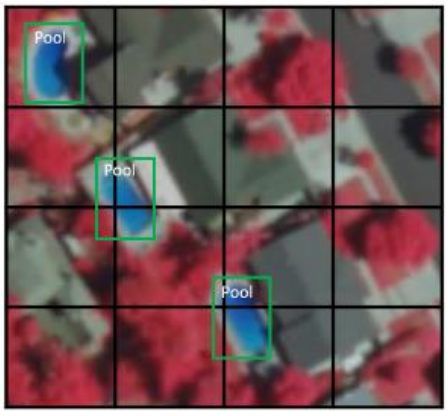


Fuente: (ArcGIS API for Python, s.f.)

2.8. Celda de Cuadricula

En lugar de usar ventanas flotantes, SSD divide la imagen en una cuadricula, donde cada celda de la cuadricula es responsable de detectar objetos en esa ventana gráfica. Si no hay ningún objeto, se ignora la ubicación de ese cuadrante.

Figura 8: Ejemplo de una cuadrícula 4x4.

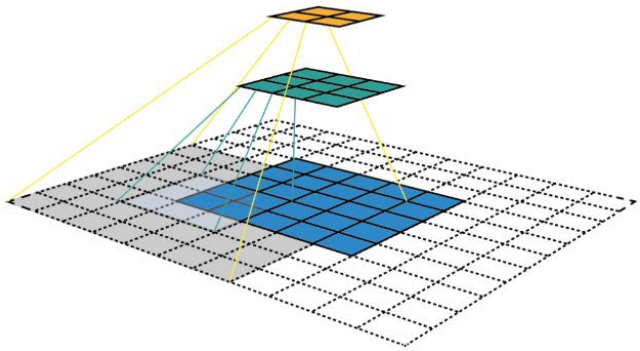


Fuente: (ArcGIS API for Python, s.f.)

2.9. Campo Receptivo

Se define como una región en el espacio de entrada que está enfocada (es decir, afectada por) una característica particular de la CNN. Por lo general los términos "características" y "activación" indistintamente como si fuera una combinación lineal de la capa anterior en este punto (a veces se usa una función de activación después de agregar no linealidad). Debido al proceso de convolución, los elementos de diferentes capas representan diferentes tamaños de campo en la imagen de entrada. Las características en el mismo mapa de características comparten el mismo campo de percepción y busca el mismo patrón en diferentes ubicaciones. Esto da como resultado la invariancia espacial de ConvNet (ArcGIS API, 2021).

Figura 9: Visualización de mapas de características de CNN y campo receptivo.



Fuente: (ArcGIS API, 2021)

2.10. Sistemas Embebidos

Es un sistema electrónico que está especialmente diseñado para la realizar una función específica y generalmente es parte de un sistema más grande. Además, el software generalmente está integrado en la ROM (Read Only Memory), por lo que no necesita un almacenamiento secundario como una computadora (Úbeda Miñarro, 2009).

Los sistemas integrados deben tener restricciones en tiempo real. Un sistema en tiempo real debe responder a los estímulos del objeto controlado dentro de un periodo de tiempo definido por el entorno (Valvano, 2003).

2.11. Hardware

Se trata de un módulo electrónico colocado en un sistema más grande "host" que realiza tareas como procesar datos generados por sensores y controlar ciertos actuadores.

En el ámbito al que se refiere Visión Artificial requiere una GPU (Unidad de Procesamiento Gráfica) potente, es un coprocesador dedicado al procesamiento de imágenes y operaciones de punto flotante. Básicamente al ser un procesador más añadido, su trabajo es de aligerar la carga del procesador así aumentando el rendimiento del ordenador (Domínguez Pavón, 2019).

2.11.1. NVIDIA Jetson Nano

NVIDIA Jetson Nano®™ puede brindar capacidades nuevas e increíbles a millones de sistemas de inteligencia artificial pequeños y de bajo consumo. (NVIDIA Corporation, 2020).

Jetson Nano es una potente microcomputadora que puede ejecutar varias redes neuronales en paralelo para aplicaciones como distribución de imágenes, reconocimiento de objetos, segmentación y procesamiento de voz (NVIDIA Corporation, 2020).

Tabla 1: Especificaciones del equipo Jetson Nano

Componente	***Características***
GPU	GPU con arquitectura NVIDIA Maxwell de 128 núcleos
CPU	Procesador ARM Cotrx – A57 MPCore de 4 núcleos
Memoria RAM	LPDDR4 de 4 GB y 64 bits
Almacenamiento	MicroSD
Redes	Gigabit Ethernet, M.2 Key E
Cámara	2 conectores de cámara MIPI CSI-2 de 15 clavijas y 2 canales
Especificaciones	100 x 79 x 30.21 mm

Elaborado por Anasi. R & Martínez. G

2.11.2. Cámara – AI

Las cámaras utilizadas en visión artificial requieren de un conjunto de funciones que permitan controlar el bolso de la cámara para capturar las piezas que pasan por delante en la posición deseada. Son más avanzadas que las cámaras tradicionales, porque deben controlar por completo: tiempos, señales, velocidad de obturación, sensibilidad, etc.

2.11.3. Protocolo RTSP

El protocolo RTSP se deriva del acrónimo inglés Real Time Streaming Protocol y, como su nombre indica, es un protocolo de transmisión de datos en tiempo real. Su función es establecer y controlar uno o más flujos de datos síncronos (audio o video). Para ver una transmisión de video en vivo con un reproductor multimedia compatible con RTSP, primero debe obtener la URL de la transmisión RTSP de su cámara IP, configurar el reproductor multimedia y establecer un enlace (Moya García, 2018).

2.12. Software

El software es parte de un sistema embebido, que puede entender como un subsistema de procesamiento electrónico programado para realizar una o varias funciones para lograr un objetivo específico (Pérez, 2009).

La regla principal del software embebido es interactuar con el mundo físico, pero el desarrollo de aplicaciones comerciales intenta olvidar el mundo físico y centrarse en abstracciones como entidades de información (Úbeda Miñarro, 2009).

2.12.1. Jet Pack 4.6

Esta es la última versión de producción y es compatible con todos los módulos Jetson. JetPack 4.6 incluye soporte para nuevas versiones de Triton Inference Server, CUDA, cuDNN y TensorRT, VPI 1.1 con soporte para nuevos algoritmos de procesamiento de imágenes y enlaces de Python, actualizaciones inalámbricas, características de seguridad. Herramienta para flashear medios internos o externos adjuntos a Jetson (NVIDIA DEVELOPER, 2020).

2.12.2. TensorRT

Es una biblioteca C++ que permite la inferencia de alto rendimiento en las unidades de procesamiento de gráficos (GPU) de NVIDIA. Está diseñado para complementar marcos de capacitación como TensorFlow, Caffe, PyTorch y MXNet (Jímenez Varela, 2021). En particular, se enfoca en ejecutar de manera rápida y eficiente redes ya entrenadas en GPU para producir resultados (un proceso llamado evaluación, detección, regresión o inferencia en varios lugares) (NVIDIA DEVELOPER, 2020).

2.12.3. CUDA

Es una plataforma de computación paralela y un modelo de programación de propósito general en las GPU NVIDIA para resolver muchos problemas computacionales complejos de manera más eficiente que las CPU (NVIDIA Corporation, 2022).

CUDA viene con un software que permite a los desarrolladores usar C++ como lenguaje de programación de alto nivel. Este kit de herramientas incluye compiladores para GPU NVIDIA, bibliotecas matemáticas y herramientas de depurar y ajustar el rendimiento de las aplicaciones (NVIDIA DEVELOPER, 2020).

2.12.4. Open CV

Cuenta con un conjunto de herramientas para la visión avanzada y procesamiento flexible de imágenes. Este escrito en C++ y ejecuta interfaces en lenguajes como Python, C++ y Java lo que permite una fácil integración con varios sistemas y algoritmos de procesamiento de imágenes y videos diferentes, incluidos algunos aprendizaje automático y visión artificial (Vargas Maisa, 2019).

CAPÍTULO 3

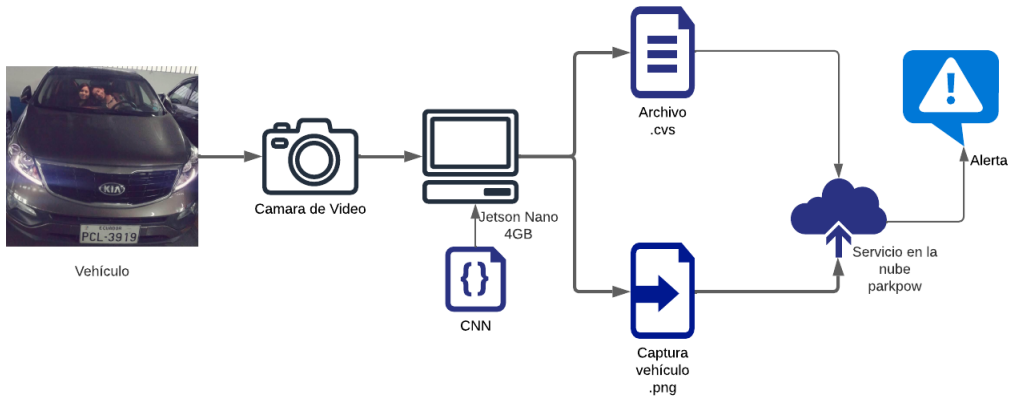
DISEÑO E IMPLEMENTACIÓN

Este capítulo aborda el proceso que se realizó para la implementación del proyecto, se describe el acondicionamiento del hardware y el proceso para la identificación y lectura de las placas de un vehículo para el control de combustible en gasolineras usando redes neuronales convolucionales y visión artificial.

3.1. Desarrollo de la propuesta

Se presenta el siguiente diagrama de flujo en la Figura 10que detalla en forma general las etapas que se llevó a cabo para la implementación del sistema de control de combustible utilizando redes neuronales convolucionales y visión artificial.

Figura 10: Diagrama de flujo del sistema.



Elaborado por Anasi. R & Martínez. G

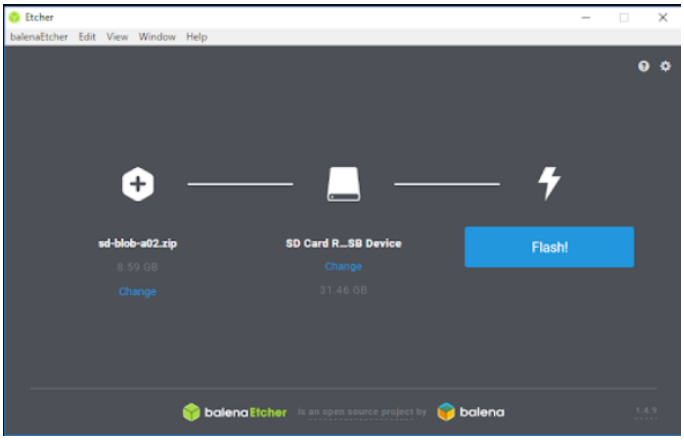
3.2. Configuración de la Jetson Nano

Para la instalación del sistema operativo con la ayuda de un computador descargar la imagen JetPack SDK 4.6_L4T 32.6.1. se procede a guardar la imagen en la tarjeta microSD con la ayuda del software ETCHER.

Insertar la tarjeta microSD en la ranura del módulo de Jetson Nano conecte teclado, mouse, pantalla HDMI y encienda, en el primer arranque le pedirá que acepte

términos y condiciones del software siga las indicaciones y al finalizar se debería mostrar esta pantalla

Figura 11: Interfaz gráfica del software ETCHER.



Elaborado por Anasi. R & Martínez. G

3.2.1. Biblioteca Jetson Nano (libjetson-inference)

Es una biblioteca de redes de aprendizaje profundo para el reconocimiento de imágenes, detección de objetos, segmentación, diseñada para ejecutarse en los sistemas de NVIDIA como Jetson con soporte en varios lenguajes de programación como C++, Python.

Se detalla la instalación de la biblioteca en la tarjeta jetson, se abre un terminal y se transcribe lo siguientes comandos.

- $ git clone --recursive https://github.com/dusty-nv/jetson-inference
- $ cd jetson-inference
- $ mkdir build
- $ cd build
- $ cmake. /

3.2.2. Montaje de Memoria SWAP

La memoria instalada 4gb resulta insuficiente para realizar operaciones de inferencia en redes neuronales complejas en tiempo real, sin embargo, con un artificio se

puede aumentar esta memoria de intercambio. Se abre un terminal en consola y se ejecuta los siguientes comandos como se indica en la Figura 12.

Figura 12: Comandos para aumentar la memoria de intercambio.

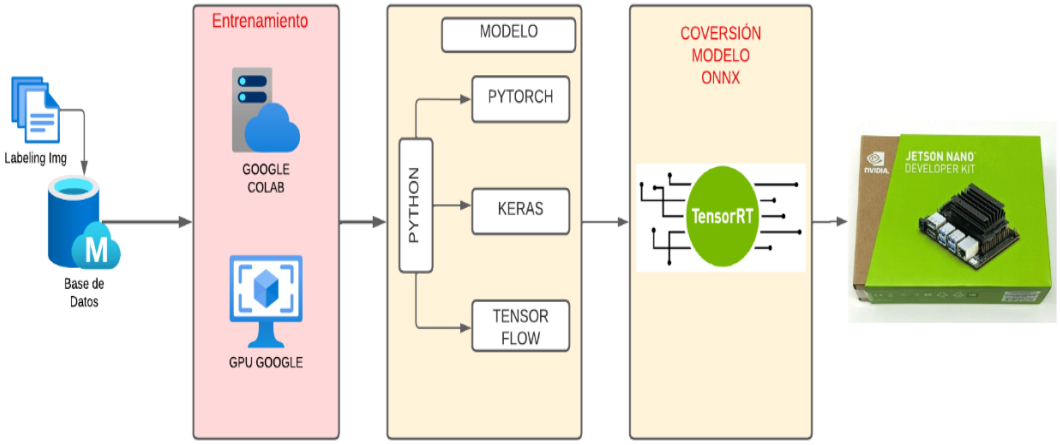
```
sudo systemctl disable nvzramconfig
sudo fallocate -l 4G /mnt/4GB.swap
sudo mkswap /mnt/4GB.swap
sudo swapon /mnt/4GB.swap
```

Elaborado por Anasi. R & Martínez. G

3.3. Reentrenamiento de la Red Neuronal Mobilnet V2

El proceso de aprendizaje de la red neuronal se indica en el siguiente diagrama de flujo la Figura 13 que detalla en forma general las etapas que se llevó a cabo para entrenar la red y utilizarla en la Jetson NANO.

Figura 13: Arquitectura del sistema.



Elaborado por Anasi. R & Martínez. G

3.3.1. Establecer y normalizar DATASET

Se creó un dataset formado por 2 tipos de muestra cómo se indica en la Figura 14, la primera recopila 6 fotografías de un mismo vehículo mostrando la placa en distintos ángulos con un subtotal de 900 imágenes, la segunda es una recopilación en

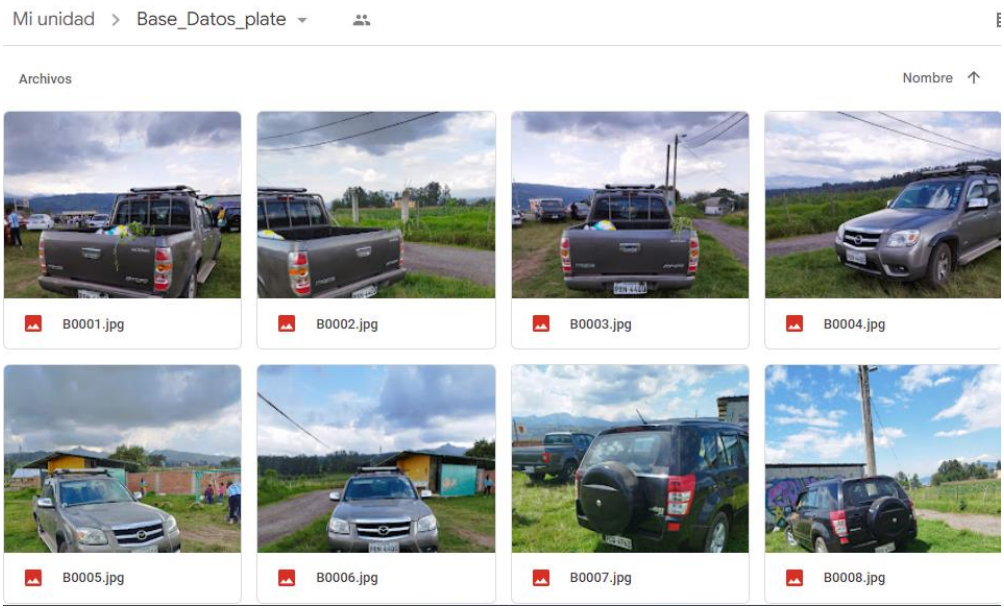
estacionamientos, centros comerciales, vías urbanas y perimetrales con un total de 1600 imágenes con un dataset de 2500 muestras

Figura 14: A) Dataset del mismo auto B) Dataset de varios autos

Elaborado por Anasi. R & Martínez. G

La totalidad de las muestras se someten a una conversión al formato .jpg con la finalidad de reducir su tamaño de almacenamiento sin perder calidad de imagen en el proceso para terminar, se cambia el nombre de cada muestra a un formato sencillo en este caso se usó B<número>.jpg como se muestra en la Figura 15.

Figura 15: Dataset de las placas de vehículos.

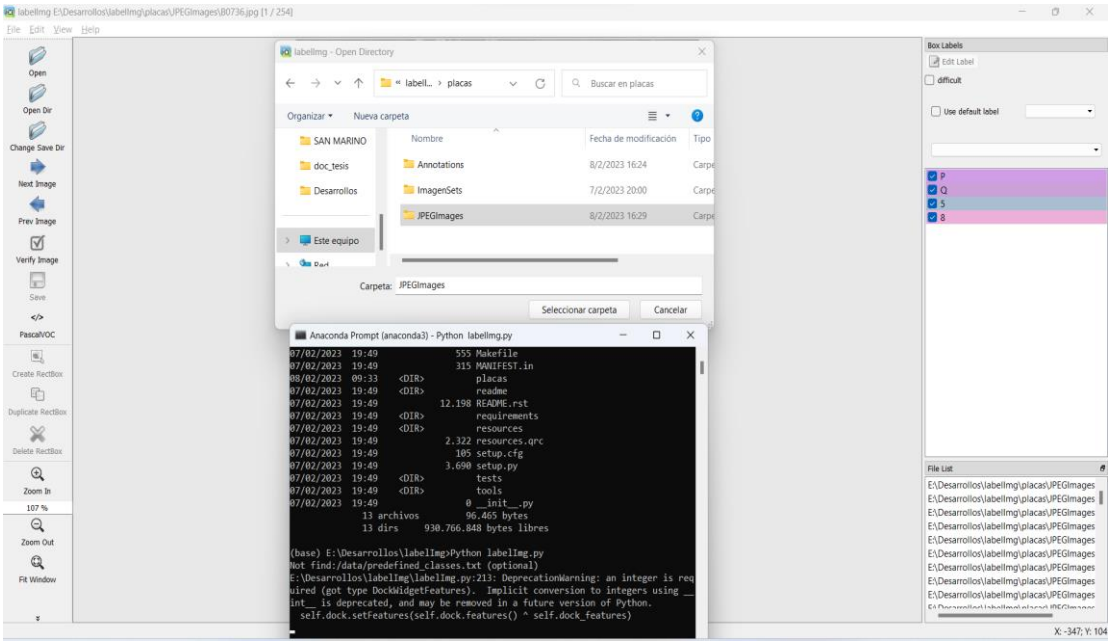


Elaborado por Anasi. R & Martínez. G

3.3.2. Etiquetado del Dataset con “labelImg”

Para el proceso del etiquetado de cada imagen se lo va a realizar desde la interfaz de la consola Anaconda en la cual se instalará el software “labelImg” que permite que su información de etiquetado se pueda convertir directamente en un archivo XML el cual será utilizado por ImageNet.

Figura 16: Interfaz gráfica del labelImg.



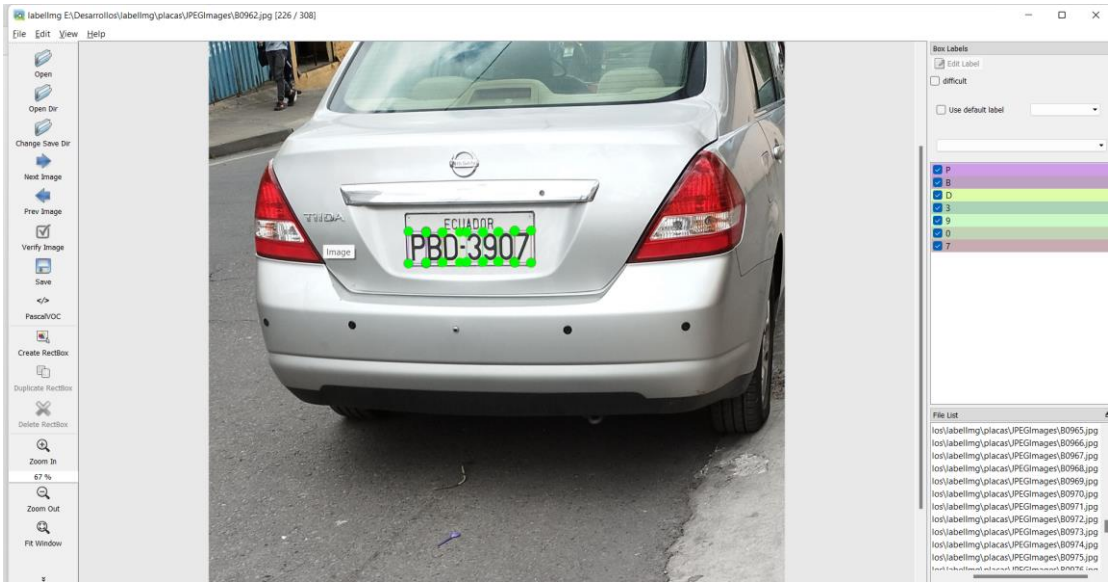
Elaborado por Anasi. R & Martínez. G

Como se observa en la Figura 16, la pantalla negra es la consola en la cual se instaló el software de **labelImg,** una vez dentro de la interfaz se sube el Dataset (imágenes de las placas de vehículos), posteriormente se observará en la parte inferior que está cargado el Dataset para proceder a etiquetarlos.

En la Figura 17 se observa que se selecciona el conjunto de datos e identificar cada característica para clasificar. Las anotaciones de las imágenes, se proporcionan cuadros delimitadores de las imágenes para después de etiquetar y extraer los datos del ordenador, se convierten y se guardan como archivos .XML como muestra la Figura 18, a continuación, se los convierte en formato PASCAL VOC, que es el formato que utiliza

ImageNet de Jetson NANO. Todo este proceso se lo realiza para las 2500 muestras para tener un dataset robusto.

Figura 17:: Etiquetado del Dataset.



Elaborado por Anasi. R & Martínez. G

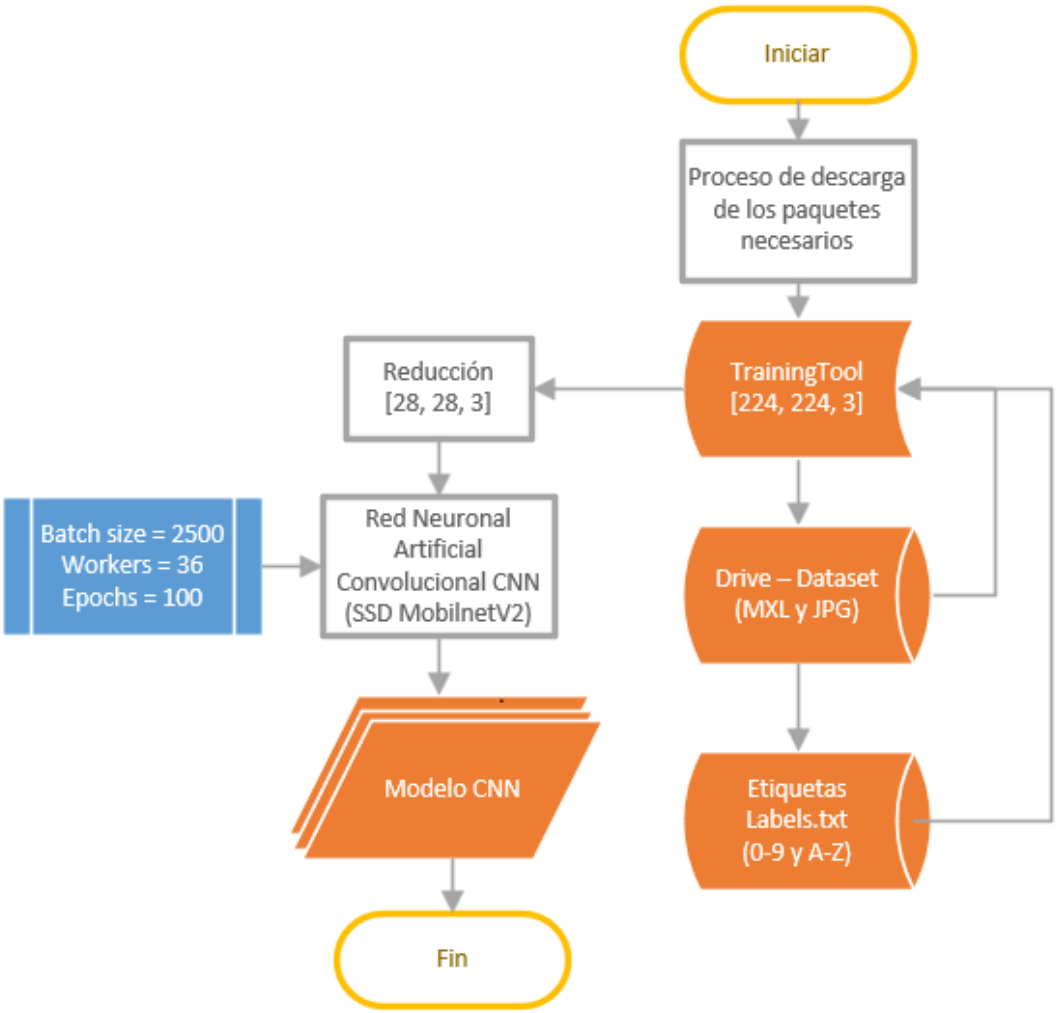
Figura 18: Archivos XML de salida del Dataset.

B0749.xml	7/2/2023 20:53	Archivo XML
B0750.xml	7/2/2023 20:54	Archivo XML
B0751.xml	7/2/2023 20:55	Archivo XML
B0752.xml	7/2/2023 20:56	Archivo XML
B0753.xml	7/2/2023 20:57	Archivo XML
B0754.xml	7/2/2023 20:57	Archivo XML
B0755.xml	7/2/2023 20:58	Archivo XML
B0756.xml	7/2/2023 20:59	Archivo XML
B0757.xml	7/2/2023 20:59	Archivo XML
B0758.xml	7/2/2023 21:00	Archivo XML
B0759.xml	7/2/2023 21:01	Archivo XML
B0760.xml	7/2/2023 21:01	Archivo XML
B0761.xml	7/2/2023 21:02	Archivo XML
B0762.xml	7/2/2023 21:03	Archivo XML

Elaborado por Anasi. R & Martínez. G

3.4. Reentrenamiento de la CNN en Google Colab

Figura 19: Entrenamiento de la red neuronal convolucional CNN.

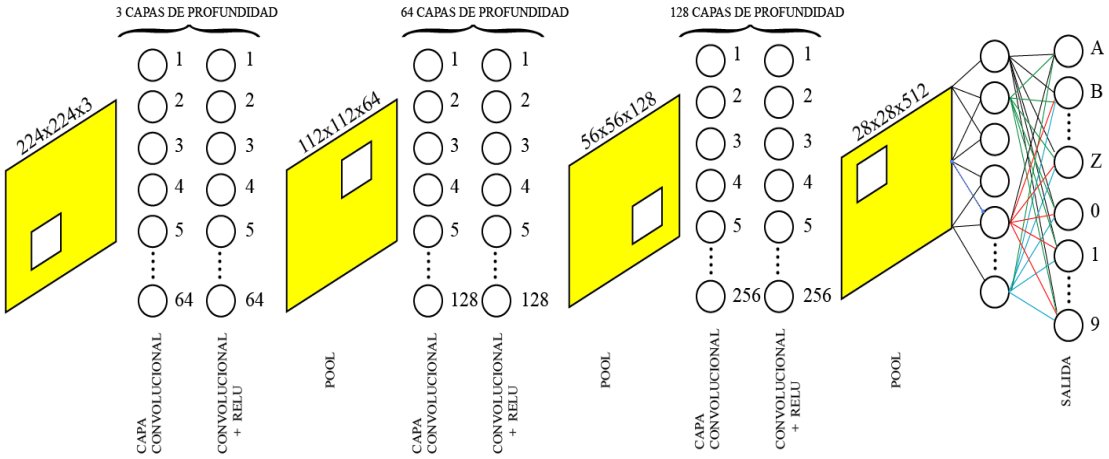


Elaborado por Anasi. R & Martínez. G

Se puede observar en la Figura 19 el entrenamiento de la red neuronal se lo va a realizar en Google Colab en la cual se compila las librerías con todo lo que respecta con la jetson inference, Se transfiere el drive donde se encuentran los Dataset (.MXL - .JPG) de igual manera el archivo de etiquetas labels.txt y lo traslada a la carpeta TrainingTool ya que ahí se realizará el entrenamiento de la red neuronal.

Se incorpora los parámetros de la red neuronal que ente caso tiene un tamaño de lote de 2500 muestras y con un número de épocas de 100

Figura 20: Arquitectura de la red neuronal CNN del sistema.



Elaborado por Anasi R & Martínez G

La red neuronal cuenta con 3 etapas

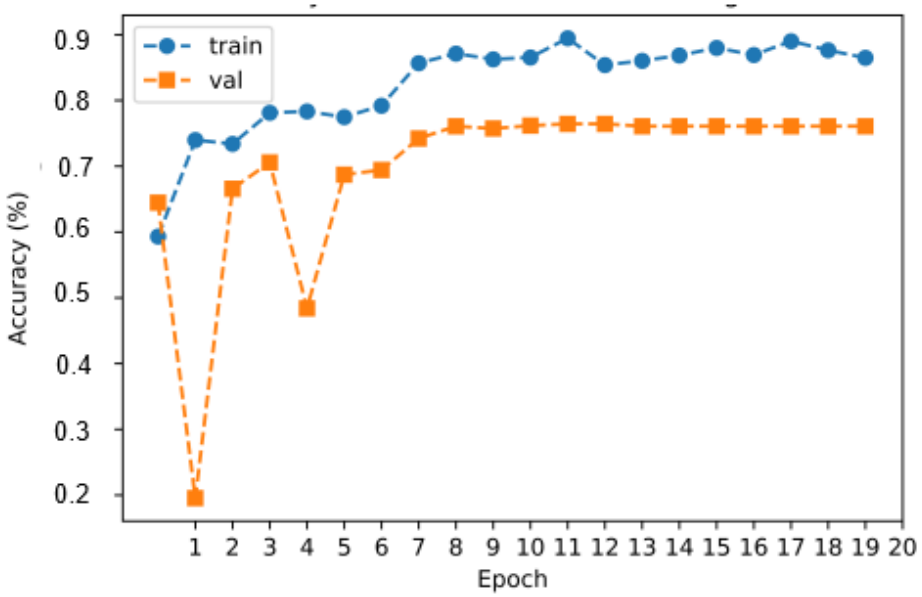
- 1^{era} etapa: 2 capas convolucionales las cuales ayudan analizar las proporciones del tamaño en la entrada, en esta etapa corresponde a un tamaño de 224x224 con 3 capas de profundidad, a una de las convoluciones se le añade una capa de activación RELU esto ayudará a sustituir todos los valores negativos recibidos en la entrada por ceros y cada una de estas capas constan de 64 neuronas.
- 2da etapa: La cual activa la capa pooling esto ayudará a reducir el tamaño de la imagen, pero preservando las características más indispensables, en esta etapa el tamaño es de 112x112x64 la cual representa la capa de profundidad y continua con 2 capas convolucionales, a una se le añade una capa de activación RELU ambas compuestas por 128 neuronas.
- 3^{era} etapa: de igual forma se reduce el tamaño de la imagen con la capa pooling que en esta etapa el tamaño es de 56x56x128 con 128 capas de profundidad y lo realiza con 2 capas convolucionales que de igual forma a una se le añade una capa de activación RELU para sustituir los valores negativos por ceros, ambas capas contienen 256 neuronas.

- Salida: es la etapa de clasificación donde debe existir la mejor probabilidad a que número o letra se asemeja más.

En la Figura 21 a medida que el modelo entrena la precisión de la red neuronal se va estabilizando con el número de épocas que van aumentado. Este modelo comienza con una precisión de un 60% en el conjunto de entrenamiento, y con el transcurso de las épocas se estabiliza con algo más de un 80% de precisión.

Se observa la precisión en el conjunto de validación, comienza en el 63% y hasta la décima época varía bastante y de ahí para adelante se estabiliza con un 70% y tienden a mejorar.

Figura 21: Gráfica de Precisión de la red neuronal.

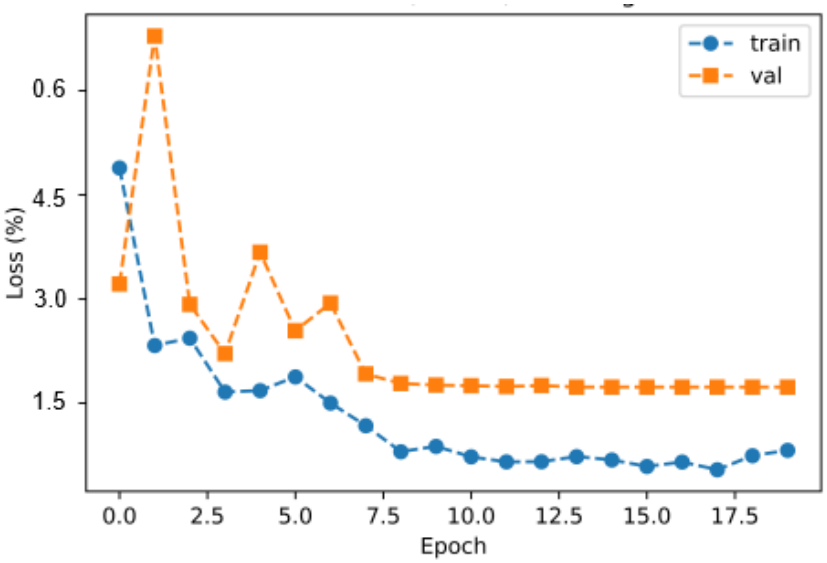


Elaborado por Anasi R & Martínez G

En la Figura 22 de igual manera para las pérdidas del entrenamiento de la red neuronal se observa que disminuye proporcionalmente mientras aumentan las épocas, tanto que el conjunto de validación comienza en un 30%, varía drásticamente aumentando hasta un 65% hasta la época 10 que comienza y se estabiliza con un 16% y en el conjunto de

entrenamiento comienza con un 49% hasta disminuir un 14%, todo este proceso es proporcional hasta que se reflejen estables.

Figura 22: Gráfica de la Perdida de la red neuronal.



Elaborado por Anasi R & Martínez G

3.5. Software

Una vez terminado el reentrenamiento como se indicó en el subcapítulo anterior, se adopta Python 3.8.10 como interprete, ya que, al tratarse de una versión antigua los bugs e incompatibilidad con las librerías Python ya están solucionados, como entorno de desarrollo integrado (IDE) se eligió Visual Studio Code a continuación las librerías utilizadas con su respectiva versión, ver Tabla 2

Tabla 2: Librerías de python

Librería	Versión	Descripción
Mediapipe	0.9.0.1	Detección de Objetos
opencv-python	4.7.0.68	Visión artificial
Openpyxl	3.1.0	Abrir archivos
Torch	1.13.1	Cálculos numéricos hacienda uso de tensores
Pandas	1.5.3	Leer y trabajar archivos cvs, xlsl
Numpy	1.24.1	Cálculos matemáticos.
Time	-	Tiempo actual
Mobilnet V2	-	Red neuronal Convolucional
Jetson Utilities	-	Pack de librerías de NVIDIA

Elaborado por Anasi. R & Martínez. G

3.5.1. Descripción del Software

El objetivo es crear un sistema con redes neuronales convolucionales CNN que tenga la capacidad de ubicar las placas de un vehículo para posteriormente leer los caracteres de esta, la información se recoge a través de una cámara streaming para enviar los datos a un servidor Web y realizar el control.

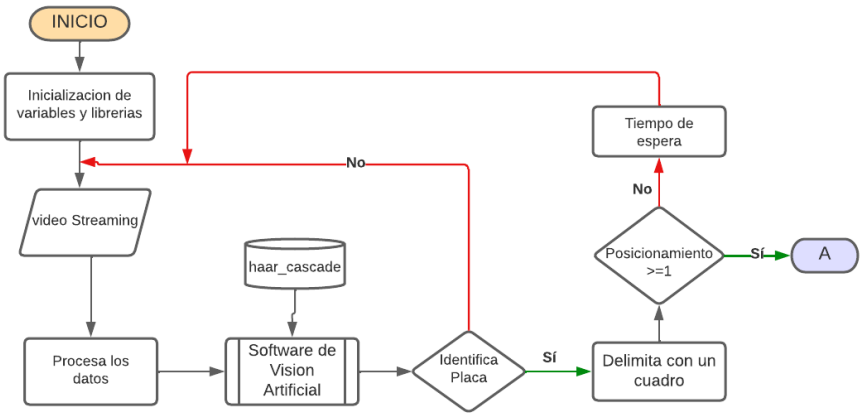
Para entender el funcionamiento se dividió en 3 etapas principales:

- Posicionamiento: Esta etapa busca eliminar lecturas falsas, si bien el rango de visión de la cámara es de algunos metros adelante su confiabilidad disminuye a mayor distancia.
- Tratamiento de los datos: Esta etapa busca optimizar el uso de los recursos computacionales.
- Repitencia: esta etapa busca optimizar de información a la CNN.

Posicionamiento

Para evitar dar lecturas erróneas se busca que el vehículo se encuentre a una distancia óptima de la cámara (alrededor de 5-6 metros) para que inicie el proceso de lectura. Usando el algoritmo Haar Cascade integrado en OpenCV si encuentra una placa entra a un bucle while donde espera 10 segundos, posicionando de esa manera al vehículo en el rango de distancia aceptable, ver en la Figura 23.

Figura 23: Diagrama de Flujo de Posicionamiento para la detección de la placa.



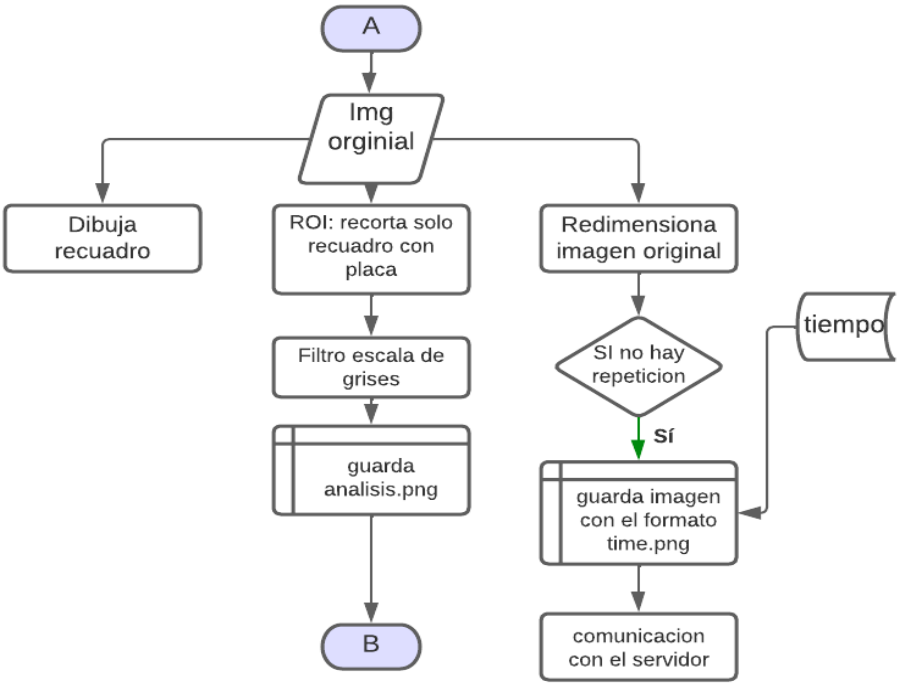
Elaborado por Anasi. R & Martínez. G

Tratamiento de datos

Enviar la información indicada para su tratamiento y procesamiento es fundamental para que el algoritmo muestre agilidad en el proceso de inferencias de información a la nube. Se generan 3 imágenes de manera simultánea cada una con características especiales, ver en la Figura 24.

- La primera ubica y dibuja sobre la imagen un recuadro en la placa
- La segunda imagen recorta únicamente la zona de interés y la trasforma en escala de grises facilitando el procesamiento al momento de extraer los caracteres, esta imagen se guarda localmente para usarla en la extracción de caracteres y es reemplazada constantemente en cada ciclo.
- La tercera imagen es la que se usa para subir a la nube, es única y se almacena localmente en la carpeta raíz, para evitar superponer las imágenes se emplea la librería "time" formando la siguiente estructura para salvar. 'hora_minutos_segundos.png'

Figura 24: Diagrama de Flujo de Tratamiento de Datos.

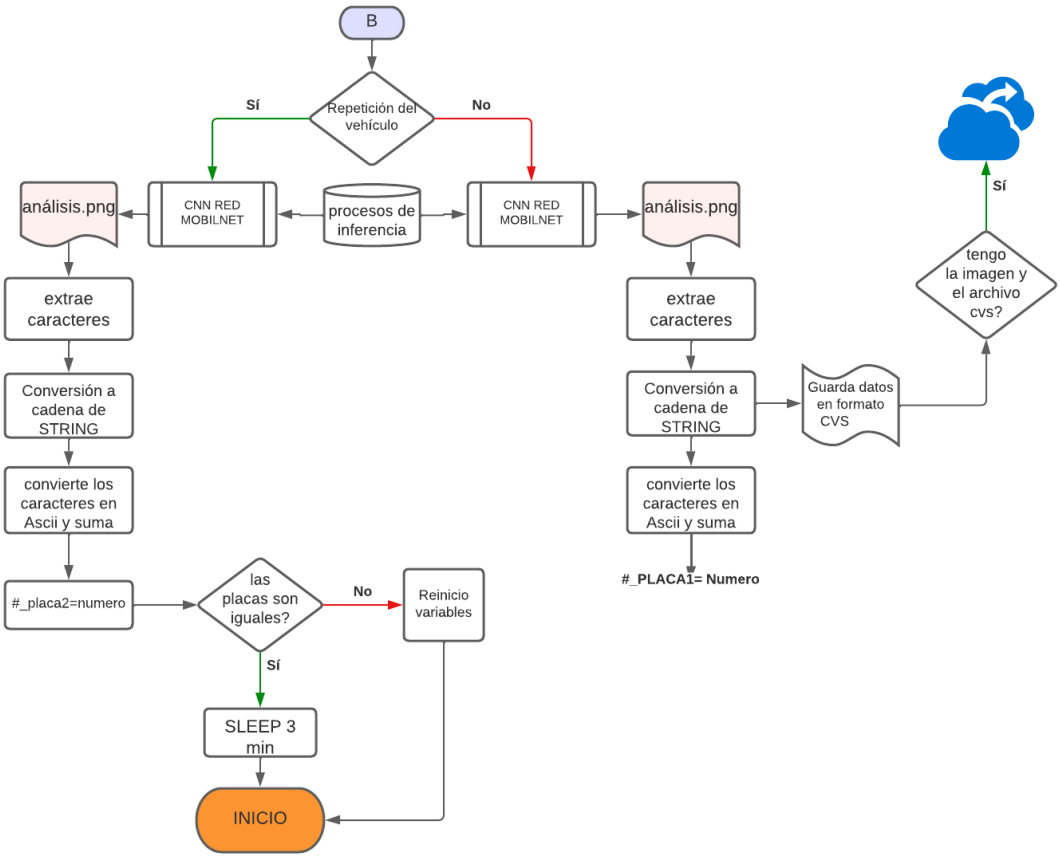


Elaborado por Anasi. R & Martínez. G

Repitencia:

En esta etapa se extrae los caracteres de la placa y los guarda en un vector para su análisis, se inicia un proceso secundario que recorre los pasos anteriores, pero en esta ocasión guarda la placa en un vector 2 con el objetivo de hacer una igualación y saber si se trata del mismo automotor o es uno nuevo, en el caso de ser el mismo vehículo el código entra en SLEPP por 3 minutos si es el caso opuesto se determina que es una placa nueva resetea las variables e inicia un nuevo análisis, ver en la Figura 25

Figura 25: Diagrama de Flujo de Repitencia.



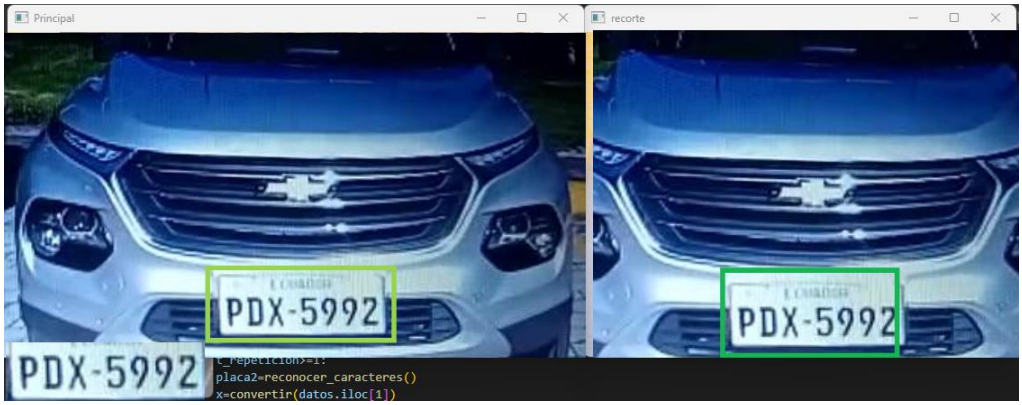
Elaborado por Anasi. R & Martínez. G

3.6. Funcionamiento del sistema

Como se puede observar en la Figura 26 la CNN ubica la placa y delimita con un cuadro verde, en la segunda imagen realiza un corte de la original para ser enviada al servidor

web ParkPow, y por último un recorte de únicamente la placa para la extracción de los caracteres.

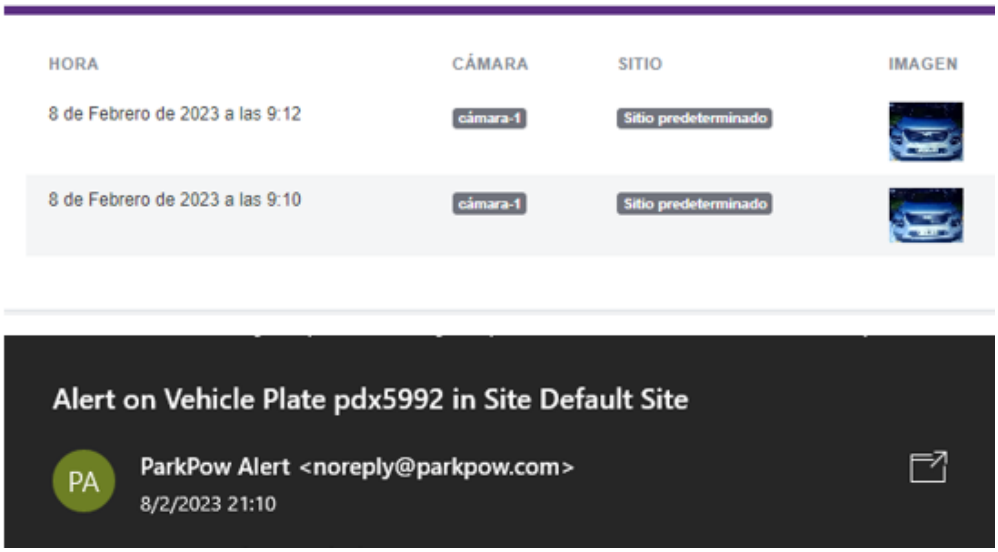
Figura 26: Interfaz de software ParkPow.



Elaborado por Anasi. R & Martínez. G

La siguiente figura muestra los paneles de ParkPow donde se observa que recibió de forma correcta la imagen del vehículo y el archivo CVS que contine la placa, de igual manera se observa que tiene una repitencia de 2 activaciones, una notificación se envía al correo electrónico configurado.

Figura 27: Registro y notificaciones de las placas de los vehículos.



Elaborado por Anasi. R & Martínez. G

3.7. Resumen de costo

El costo total del proyecto se encuentra especificado en la Tabla 3 que contiene una descripción de sus elementos y el precio en el mercado.

Tabla 3: Presupuesto final del proyecto

TIPO	DESCRPICIÓN	CANTIDAD	P. UNITARIO	P. FINAL
Materiales	Jetson Nano Developer Kit	1	$ 300,00	$ 300,00
	Case Silicon Jetson Nano	1	$ 35,00	$ 35,00
	Cooler Jerson Nano	1	$ 9,50	$ 9,50
	Tarjeta MicroSD 128GB	1	$ 40,00	$ 40,00
	Teclado USB	1	$ 9,00	$ 9,00
	Raton USB	1	$ 5,00	$ 5,00
	Cable HDMI	1	$ 5,00	$ 5,00
	Monitor	1	$ 80,00	$ 80,00
	Servicio en la nube Parkpow	1	$ 20,00	$ 20,00
	Cámara	1	$ 50,00	$ 50,00
Gastos varios	Horas de ingeniería	350	$ 2,50	$ 875,00
Valor Total Hardware				$ 473,50
Valor Total Software				$ 895,00
TOTAL				$ 1.368,50

Elaborado por Anasi. R & Martínez. G

CAPÍTULO 4

PRUEBAS Y RESULTADOS

En este capítulo se aborda los resultados que se obtuvieron en la identificación de placas por medio de visión artificial utilizando redes neuronales convolucionales, el rendimiento del sistema cuando se somete a variables externas.

4.1 Ensayo de la Red Neuronal Convolucional CNN

La red neuronal se desempeñó de forma correcta con un total de 2500 muestras, sin embargo, para llegar a ese comportamiento la CNN pasó por varios entrenamientos con datasets de menor tamaño con resultados como se indica en la Tabla 4

Para evaluar su desempeño se probó el código ingresando como input la proyección de un video que a lo largo de su duración se capturó 30 vehículos, se toma en cuenta 2 criterios para evaluar su desempeño: encuentra la placa, extraer lo caracteres de esta.

$$Encuentra\ placa = \frac{aciertos}{universo} * 100$$

$$Extrae\ caracteres = \frac{\Sigma\ \text{aciertos caracteres}}{universo * 6} * 100$$

Tabla 4: Ensayos

Tamaño Dataset	Encuentra la placa	Extrae lo caracteres de la placa
150	0%	0%
500	16.66%	0%
800	50%	16.66%
1200	80.66%	16.66%
1600	90%	66.66%
2000	92%	87.66%
2500	92%	86.33%

Elaborado por Anasi. R & Martínez. G

La Tabla 4 refleja el resultado no se aleja de lo esperado, al aumentar el tamaño del Dataset la red neuronal convolucional mejora de manera exponencial tanto en ubicar la placa como en el proceso de extraer caracteres, sin embargo, su progreso se detiene

abruptamente con un dataset de 2000 a 2500 muestras incluso disminuyendo su confianza esto se debe a la arquitectura que se propuso para entrenar la CNN.

4.2 Pruebas de funcionamiento en un ambiente controlado

Para evaluar el desempeño de la CNN se sometió a ensayos de distancia, ángulo de captura y tiempo a lo largo del día. Estas pruebas se realizaron en el Conjunto Loyola norte de Quito, con el permiso de la administración se procedió a instalar el sistema y a tomar mediciones cada hora a partir de las 5:30 am para observar su comportamiento especificado en la Tabla 5

Tabla 5: Pruebas del sistema en un ambiente controlado

N	Tiempo	DISTANCIA							
		5 metros				8 metros			
		0°		45°		0°		45°	
		Encuentra placa	Extrae placa	Encuentra placa	Extrae placa	Encuentra placa	Extrae placa	Encuentra placa	Extrae placa
1	5:30	NO	NO	NO	NO	NO	NO	NO	NO
2	6:00	SI	SI	NO	NO	NO	NO	NO	NO
3	7:00	SI	SI	SI	SI	NO	NO	NO	NO
4	8:00	SI	SI	SI	SI	SI	Parcial	Parcial	Parcial
5	9:00	SI	SI	SI	SI	SI	Parcial	SI	Parcial
6	10:00	SI	SI	SI	SI	SI	Parcial	SI	Parcial
7	11:00	SI	SI	SI	SI	SI	Parcial	SI	Parcial
8	12:00	SI	SI	SI	SI	SI	Parcial	SI	Parcial
9	13:00	SI	SI	SI	SI	SI	Parcial	SI	Parcial
10	14:00	SI	SI	SI	SI	SI	Parcial	SI	Parcial
11	15:00	SI	SI	SI	SI	SI	Parcial	SI	Parcial
12	16:00	SI	SI	SI	SI	SI	Parcial	SI	Parcial
13	17:00	SI	SI	SI	SI	SI	Parcial	SI	Parcial
14	18:00	SI	SI	SI	SI	NO	NO	NO	NO
15	19:00	NO	NO	NO	NO	NO	NO	NO	NO
Efectividad		86,66%	93%	80%	93%	66,60%	66,66%	60%	60%

Elaborado por Anasi. R & Martínez. G

En la Tabla 5 se puede observar que la distancia óptima para realizar mediciones evitando los falsos positivos es de [4-6] metros con un ángulo que pueden ser de entre 0° y 45° respecto a la cámara conforme la distancia aumenta se tienen a disminuir la

confianza en la predicción. La iluminación es un factor importante, ya que, en horas de la mañana y noche la cámara queda cegada por las propias luminarias de los vehículos haciendo imposible que realice cualquiera de las dos acciones predeterminadas.

4.3. Pruebas de funcionamiento en un ambiente no controlado

Tomando como premisa que el sistema debe funcionar en gasolineras y debe notificar la repitencia de un mismo vehículo surgen variables no controlables que a su vez son críticas para su desempeño, siendo el principal inconveniente el tiempo que tarda cada vehículo en cargar combustible.

Tabla 6: Pruebas del sistema en un ambiente no controlado

Tiempo Trascurrido	# de Repeticiones	Alertas
1:00	0	NO
1:15	0	NO
1:30	0	NO
2:00	0	NO
2:15	0	NO
2:30	0	NO
2:45	0	NO
3:00	1	SI
3:15	1	SI
3:30	1	SI
3:45	1	SI
4:00	2	SI
4:15	2	SI
4:30	2	SI
4:45	2	SI
5:00	2	SI
6:00	3	SI

Elaborado por Anasi. R & Martínez. G

Como se puede observar en la Tabla 6 el tiempo que le toma a cada automotor cargar combustible pude dar como resultados alertas falsas, como promedio el algoritmo toma muestras cada 2:50 min, si se sobrepasa ese tiempo inicia un ciclo nuevo enviando los datos nuevamente al servidor activando las notificaciones.

CONCLUSIONES

Se observó que, los resultados obtenidos en un ambiente controlado validan el funcionamiento del sistema logrando una certeza superior al 90% tanto encontrando la placa como extrayendo los caracteres de esta. Con los resultados de la Tabla 5 se puede inferir cual es la posición adecuada para colocar la cámara siendo a una distancia de entre 5 a 6 metros con un ángulo no mayor a 45° respecto a la cámara para evitar lecturas erradas.

El tiempo que tarda un vehículo abasteciéndose de combustible en una gasolinera es aleatorio, podría tardar desde 1 minuto hasta superar los 8 sin problema, debido a esta incertidumbre se eligió una media de 3 minutos por vehículo reduciendo drásticamente la confiabilidad, ya que, un nuevo automotor podría cargar en el tiempo que el software se encuentra inactivo saltándose el control. Lograr una comunicación entre la máquina surtidora de combustible y el software eleva la confianza en el control del sistema, sin embargo, este proceso conlleva una serie de tramites legales con las empresas que brindan el servicio de combustible.

La elección de la red MobilenetV2 como CNN ahorro recursos informáticos en comparación con otros modelos con altos costos de memoria. Al combinar el modelo CNN con el conjunto de datos se pudo lograr un sistema liviano con una buena tasa de predicción y demuestra que los esquemas de aprendizaje de transferencia son adecuados para aprender características relevantes en este dominio.

La red neuronal entrenada tardo aproximadamente 6 horas esto se debe a la cantidad de muestras del dataset y el número de épocas a entrenar. Como se muestra la Figura 21 y Figura 22 tanto para las perdidas como para la precisión son proporcionales y además que mitras más muestras y épocas el entrenamiento de la red neuronal será más preciso.

RECOMENDACIONES

El entrenamiento de la red neuronal convolucional se llevó a cabo en el entorno de Google Colab al constituirse de un DATASET considerablemente grande 2500 muestras y múltiples salidas 36, el servicio gratuito tiende a fallar deteniendo en cualquier momento el entrenamiento obligando a reiniciar el proceso esto implica una perdida sustancial de tiempo y recursos computacionales. Se recomienda contratar una suscripción de tier más alta que ofrece prestaciones más altas tanto en GPU, unidades de procesamiento, memoria evitando de esta manera los conflictos mencionados.

Se debe tener en cuenta que algunos modelos CNN se aplica la transferencia de aprendizaje, antes de elegir un modelo práctico se deben analizar las métricas de tamaño y precisión se relaciona con la memoria para el rendimiento del entrenamiento de la red neuronal realizado, con esto se evita fallos o demoras al momento del entrenamiento de la red nueronal

BIBLIOGRAFÍA

Álvarez Durán, M. A. (2014). Analisis, diseño e implementación de un sistema de control de ingreso de vehiculos basado en visión artificial y reconocimiento de placas en el parqueadero de la Universidad Politécnica Salesiana. *Universidad Politécnica Salesiana - Sede Cuenca*.

Anagnostopoulos, C. N., Anagnostopoulos, I., Loumos, V., & Kayafas, E. (2006). A license plate-recognition algorithm for intelligent transportation system applications. *IEEE Transactions on Intelligent transportation systems*, 377-392. Obtenido de IEEE Transactions on Intelligent transportation systems.

Andrade, P. (21 de Agosto de 2021). *La policia intenta combatir el contrabnado de combustible en Carchi*. Ecuavisa: https://www.ecuavisa.com/noticias/ecuador/la-policia-intenta-combatir-el-contrabando-de-combustible-en-carchi-NI648179

ArcGIS API. (5 de Mayo de 2021). *https://developers.arcgis.com/python/guide/how-ssd-works/*. ArcGIS API: https://developers.arcgis.com/python/guide/how-ssd-works/

ArcGIS API for Python. (5 de mayo de 2021). *How single-shot detector (SSD) works*. ArcGIS API for Python: https://developers.arcgis.com/python/guide/how-ssd-works/

ArcGIS API for Python. (s.f.). *Single-Shot Detector (SSD)*. ArcGIS API for Python: https://developers.arcgis.com/python/guide/how-ssd-works/

Benalcazar, W. (13 de Febrero de 2020). *Bajo la nueva normativa, en Carchi bajo el consumo de combustible*. EL COMOERCIO: https://www.elcomercio.com/actualidad/ecuador/consumo-combustible-carchi-vehiculos-normativa.html

Bonilla Carrión, C. (2020). Redes Convolucionales. *Universidad de Sevilla*.

Domínguez Pavón, S. (2019). Identificación del modelo de cámara mediante Redes Neuronales Convolucionales. *Escuela Técnica Superior de Ingeniería - Universidad de Sevilla*.

Ecomex360. (22 de Octubre de 2019). *Conbustibles se fuga hacia Colombia por 34 días*. Obtenido de Ecomex360: https://www.e-comex.com/combustible-se-fuga-hacia-colombia-por-34-rutas/

Endara, A., & Cabezas, R. (13 de Mayo de 2021). *En Carchi se limita la venta de combustible por vehículo, para evitar contrabando a localidades colombianas que sufren desabastecimiento por protestas*. EL UNIVERSO: https://www.eluniverso.com/noticias/ecuador/en-carchi-se-limita-la-venta-de-combustible-por-vehiculo-para-evitar-contrabando-a-localidades-colombianas-

que-sufren-desabastecimiento-por-protestas-nota/#:~:text=Ecuador-,En%20Carchi%20se%20limita%20la%20venta

Garcia Santillan, E. (2008). Detección y clasificación de objetos dentro de un salón de clases empleando técnicas de procesamiento digital de imágenes. *Universidad Autonoma Metropolitana*.

GHOURY, S., SUNGUR, C., & DURDU, A. (2019). Real-Time Diseases Detection of Grape and Grape Leaves using Faster R-CNN and SSD MobileNet Architectures. *International Conference on Advanced Technologies, Computer Engineering and Science*, 39-44.

Jímenez Varela, N. (2021). Evaluación de la plataforma Nvidia Jetson Nano para aplicaciones de visión artifcial. *UNIVERSIDADAUTÓNOMADEMADRID*.

Moya García, L. M. (2018). Módulo de accseso y visualización de contenido muktimedia proveniente de cámaras IP, para el Sistema Domótico del CEDIN. *Universidad de las Ciencias Informaticas*.

NVIDIA Corporation. (26 de Mayo de 2020). *Introducción a Jetson Nano Developer Kit*. Obtenido de NVIDIA Corporation: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#intro

NVIDIA Corporation. (26 de Mayo de 2020). *Kit y módulo para desarrolladores de Jetson Nano*. Obtenido de NVIDIA Corporation: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/#shop-all

NVIDIA Corporation. (2022). CUDA C++ Programming Guide. *Design Guide*.

NVIDIA DEVELOPER. (16 de Mayo de 2020). *JetPack SDK 4.6 Release Page*. Obtenido de NVIDIA Corporation: https://developer.nvidia.com/embedded/jetpack-sdk-46#collapseJetsonNano

NVIDIA DEVELOPER. (2020). TENSORRT.

Pérez, D. (2009). Sistemas Embebidos y Sistemas. *Universidad Central de Venezuela. Facultad de Ciencias. Escuela de Computación* .

Preeti, N., Rachna, J., Agam, M., Rohan, A., Piyush, K., & Jude, H. (2021). SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2. *ELSEVIER*.

Rebato, C. (14 de Abril de 2022). *Qué es el Edge Computing, explicado de manera sencilla*. Obtenido de Telefonia Tech: https://empresas.blogthinkbig.com/edge-computing-que-es/

Rentana Ribeiro, D. (11 de Mayo de 2020). *Edge computing con Nvidia Jetson Nano*. Obtenido de Paradigma Digital: https://www.paradigmadigital.com/dev/edge-computing-con-nvidia-jetson-nano/

Romero Gómez, Y. R. (2021). Plataforma de servicio para la identificación de placas vehiculares que permita la automatización del control de parqueaderos. *Universidad de las Fuerzas Armadas ESPE*.

Rosado, P., Figueras, E., Planas, M., & Reverter, F. (2015). La visión artificial, un nuevo aliado para las imagenes artisticas. 325-333.

Satyanarayanan, M. (2017). Edge Computing. *Computer*, 36-38.

Shashirangana, J., Padmasiri, H., Meedeniya, D., & Perera, C. (2021). Automated License Plate Recognition: A Survey on Methods and Techniques. *IEEE Access*.

Tolosa Cuadrado, C. L., & González Sanabria, J. S. (2014). Amazon Web Services: alternativa para el almacenamiento de información.

Úbeda Miñarro, B. (2009). Introducción y generalidades acerca del diseño de sistema embebidos.

Valvano, J. (2003). *Introducción a Los Sistemas de Microcomputadoras Embebidos: SimulaciÓn de Motorola G811 Y G812.* International Thomson Editores.

Vargas Maisa, W. G. (2019). ANÁLISIS DE OBJETOS TRANSLÚCIDOS USANDO TÉCNICAS DE VISIÓN POR COMPUTADOR. *UNIVERSIDAD TÉCNICA DE AMBATO*.

Varghese, B. W. (2016). Challenges and opportunities in edge computing. *IEEE Access*, 20-26.

Vega, H. C. (2019). Análisis de objetos tránslucidos usando técnicas de visión por computador. *Universidad Nacional Mayor de San Marcos*.

ANEXOS

Anexo 1: Entrenamiento de la red neuronal en Google Colab

Se inicia con un documento nuevo en Google Colab donde se transfiere las carpetas de unidad drive en las que se encuentra el Dataset de aproximadamente 2500 muestras en formato JPEG como MXL y se lo mueve a la carpeta TrainingTools para realizar entrenamiento de la red neuronal

```
from google.colab import drive
drive.mount('/content/drive')
```

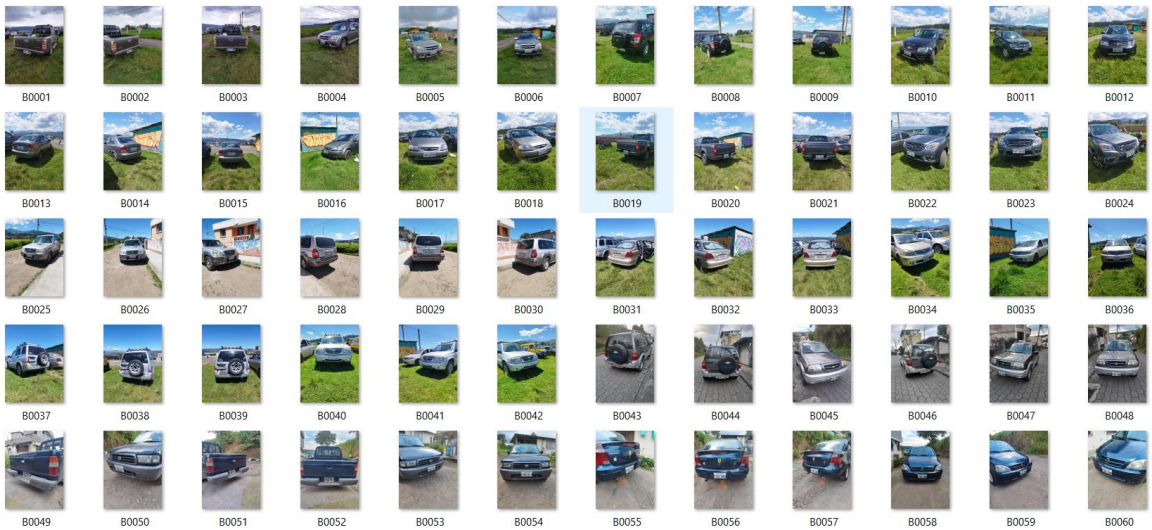
```
Mounted at /content/drive
```

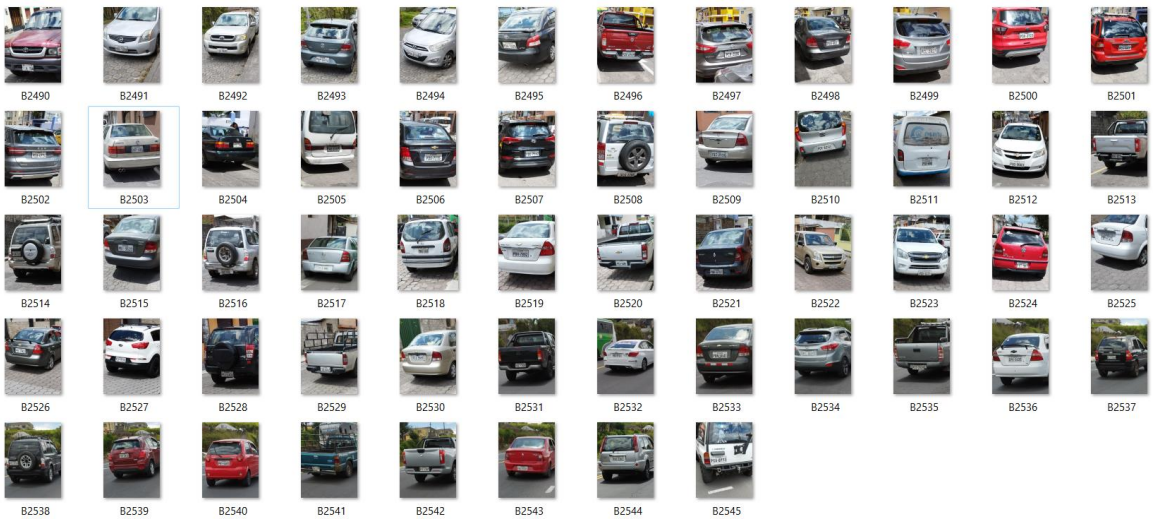
+ Código + Texto

Move Annotations and JPEGImages to TrainingTools/ssd/

Move the folders to the SSD Directory

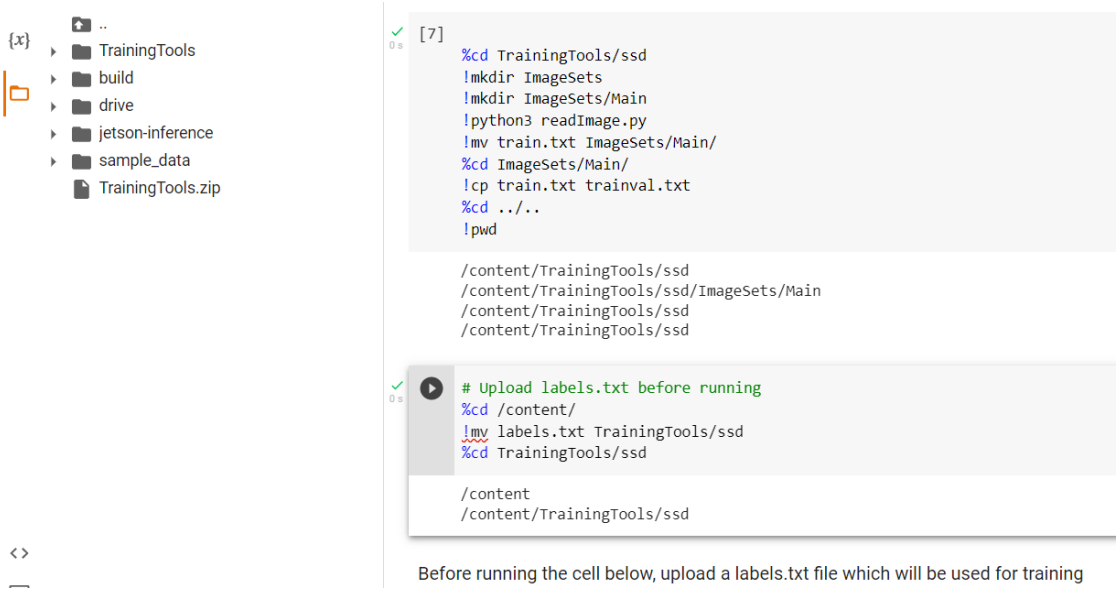
```
[ ]  !cp -r drive/MyDrive/Desarrollos/Placas/Annotations/ TrainingTools/ssd/
     !cp -r drive/MyDrive/Desarrollos/Placas/JPEGImages/ TrainingTools/ssd/
```





La siguiente celda creara la carpeta ImageSet y dentro de esta otra carpeta main en donde se generarán los archivos de entrenamiento y validación ambas en txt. A continuación, se debe cargar el archivo labels.txt en donde se encuentra cada una de las etiquetas (A – Z y 0 - 9) y ubicarlos en la carpeta TrainingTools en la parte ssd

Proceso de entrenamiento



Proceso de entrenamiento:

Con un tamaño de muestras de 2545 se ubica esa cantidad en “batch size”, en el parámetro de “works” = 36 ya que son la cantidad total de etiquetas de letras y números, por últimos una cantidad de “epocas” = 50

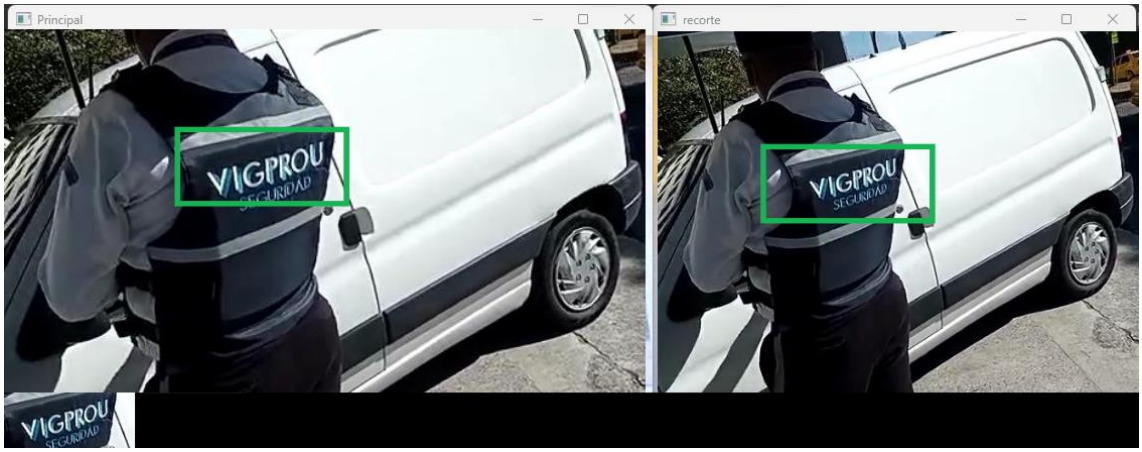
Como se observa se demora aproximadamente 5 horas esto es debido a la cantidad de muestras se tiene y es computacionalmente costoso

```
!python3 train_ssd.py --dataset-type=voc --data=./ --model-dir=models/Chess --batch-size=2500 --workers=36 --epochs=50
2023-02-09 04:42:21 - Epoch: 42, Step: 10/58, Avg Loss: 7.6706, Avg Regression Loss 1.7625, Avg Classification Loss: 5.9082
2023-02-09 04:42:52 - Epoch: 42, Step: 20/58, Avg Loss: 7.1311, Avg Regression Loss 1.6114, Avg Classification Loss: 5.5198
2023-02-09 04:43:17 - Epoch: 42, Step: 30/58, Avg Loss: 6.5759, Avg Regression Loss 1.4265, Avg Classification Loss: 5.1494
2023-02-09 04:43:43 - Epoch: 42, Step: 40/58, Avg Loss: 6.7266, Avg Regression Loss 1.2673, Avg Classification Loss: 5.4593
2023-02-09 04:44:11 - Epoch: 42, Step: 50/58, Avg Loss: 6.8244, Avg Regression Loss 1.4589, Avg Classification Loss: 5.3655
2023-02-09 04:44:29 - Epoch: 42, Validation Loss: 6.5416, Validation Regression Loss 1.0184, Validation Classification Loss: 5.5232
2023-02-09 04:44:29 - Saved model models/Chess/mb1-ssd-Epoch-42-Loss-6.541616519292195.pth
2023-02-09 04:45:09 - Epoch: 43, Step: 10/58, Avg Loss: 7.6063, Avg Regression Loss 1.5222, Avg Classification Loss: 6.0841
2023-02-09 04:45:32 - Epoch: 43, Step: 20/58, Avg Loss: 6.7821, Avg Regression Loss 1.3814, Avg Classification Loss: 5.4008
2023-02-09 04:45:56 - Epoch: 43, Step: 30/58, Avg Loss: 6.9110, Avg Regression Loss 1.3538, Avg Classification Loss: 5.5572
2023-02-09 04:46:25 - Epoch: 43, Step: 40/58, Avg Loss: 6.6428, Avg Regression Loss 1.3813, Avg Classification Loss: 5.2616
2023-02-09 04:47:02 - Epoch: 43, Step: 50/58, Avg Loss: 6.8114, Avg Regression Loss 1.4327, Avg Classification Loss: 5.3787
2023-02-09 04:47:25 - Epoch: 43, Validation Loss: 6.4897, Validation Regression Loss 0.9450, Validation Classification Loss: 5.5448
2023-02-09 04:47:26 - Saved model models/Chess/mb1-ssd-Epoch-43-Loss-6.489728609720866.pth
2023-02-09 04:48:04 - Epoch: 44, Step: 10/58, Avg Loss: 8.1164, Avg Regression Loss 2.2519, Avg Classification Loss: 5.8646
2023-02-09 04:48:33 - Epoch: 44, Step: 20/58, Avg Loss: 6.9889, Avg Regression Loss 1.6818, Avg Classification Loss: 5.3071
2023-02-09 04:48:56 - Epoch: 44, Step: 30/58, Avg Loss: 6.7331, Avg Regression Loss 1.4815, Avg Classification Loss: 5.2516
2023-02-09 04:49:18 - Epoch: 44, Step: 40/58, Avg Loss: 6.8024, Avg Regression Loss 1.4681, Avg Classification Loss: 5.3343
2023-02-09 04:49:45 - Epoch: 44, Step: 50/58, Avg Loss: 6.8691, Avg Regression Loss 1.4310, Avg Classification Loss: 5.4382
2023-02-09 04:50:01 - Epoch: 44, Validation Loss: 6.5667, Validation Regression Loss 1.0594, Validation Classification Loss: 5.5073
2023-02-09 04:50:01 - Saved model models/Chess/mb1-ssd-Epoch-44-Loss-6.566710074742635.pth
2023-02-09 04:50:36 - Epoch: 45, Step: 10/58, Avg Loss: 7.5661, Avg Regression Loss 1.6276, Avg Classification Loss: 5.9385
2023-02-09 04:51:03 - Epoch: 45, Step: 20/58, Avg Loss: 6.7129, Avg Regression Loss 1.4146, Avg Classification Loss: 5.2983
2023-02-09 04:51:26 - Epoch: 45, Step: 30/58, Avg Loss: 6.6498, Avg Regression Loss 1.2568, Avg Classification Loss: 5.3930
2023-02-09 04:51:59 - Epoch: 45, Step: 40/58, Avg Loss: 6.8562, Avg Regression Loss 1.5054, Avg Classification Loss: 5.3508
2023-02-09 04:52:25 - Epoch: 45, Step: 50/58, Avg Loss: 6.7973, Avg Regression Loss 1.4861, Avg Classification Loss: 5.3111
2023-02-09 04:52:42 - Epoch: 45, Validation Loss: 6.4469, Validation Regression Loss 0.9585, Validation Classification Loss: 5.4885
2023-02-09 04:52:42 - Saved model models/Chess/mb1-ssd-Epoch-45-Loss-6.446928977966309.pth
```

ANEXO 2

Funcionamiento del sistema en ambientes no controlados.

Como se puede observan en la imagen siguiente el sistema reconoce el chaleco del guardia como una placa, entregando incluso la lectura de esta y notificando su repitencia vía correo electrónico.



HORA	CÁMARA	SITIO	IMAGEN
9 de Febrero de 2023 a las 9:09	cámara-1	Sitio predeterminado	
9 de Febrero de 2023 a las 9:02	cámara-1	Sitio predeterminado	
9 de Febrero de 2023 a las 7:59	cámara-1	Sitio predeterminado	

ANEXO 3

Código de programación del sistema realizado en Python.

Código inicialización de variables, definición de funciones y video streaming

```
import cv2
import time
import random
import pandas as pd
import datetime   # libreria de fecha y hora
from pathlib import Path #libreria para crear directorios

dataPath='C:\Users\andyl\Documents\Python\plcas.cvs'

x=['', '', '', '', '', ''] # matriz donde guarda la placa
datos=pd.read_excel('plcas.xlsx',header=None)

def ParkPow_weebooks():···

def reconocer_caracteres(): # llama a la red mobilnet para que busque caracteres···

def verificador_placas(veri): # lee los caracteres en ascci y suma···

def convertir(v): #convierte los patrones en un vector de String···


width=640 #dimensiones ventana
height=360 #dimensiones ventana
park=""
t_espera=0 #contador 2 vez toma el dato
t_repeticion=0 #contador si el vehiculo es el mismo
placa1=['','','','','','']
placa2=['','','','','','']
num_p1=0 #suma assci
num_p2=0
#incializar camara
cam=cv2.VideoCapture(0,cv2.CAP_DSHOW)
```

Código para que detectar la placa

```
#incializar camara
cam=cv2.VideoCapture(0,cv2.CAP_DSHOW)
cam.set(cv2.CAP_PROP_FRAME_WIDTH, width)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT,height)
cam.set(cv2.CAP_PROP_FPS, 30)
cam.set(cv2.CAP_PROP_FOURCC,cv2.VideoWriter_fourcc(*'MJPG'))
 #nombre variable= cv2_nombre de la libreria(ruta)
placa_r=cv2.CascadeClassifier('haar\data\haarcascade_licence_plate_rus_16stages.xml')
fecha=datetime.datetime.now() #inicializo Fecha actual
fecha1=datetime.datetime.strftime(fecha,'%d_%m_%Y') #extraigo dia mes y año
#crear una carpeta y si ya existe no hace nada
Path(fecha1).mkdir(exist_ok=True)

while True:
    ignore,  frame = cam.read() #leo la camara
    frameGray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY) #escala de grises
    fecha=datetime.datetime.now() #iniciar fecha
    fecha1=datetime.datetime.strftime(fecha,'%H_%M_%S') #extraigo hora-min-seg
    park=fecha1
    placa = placa_r.detectMultiScale(frameGray, 1.1 , 3)

    for plr in  placa:···




    cv2.imshow('Principal', frame)
    cv2.moveWindow('Principal',0,0)
    if cv2.waitKey(1) & 0xff ==ord('q'):
        break
cam.release()
```

Código para ubicar la placa y guardar las 3 imágenes para su Procesamiento

```
if x>0 or y>0 :

    if t_espera>=1:

     # altura y ancho
     frameROI = frame[y:240,x:480] #recorta mi region de interes placa
     frameROIgray = cv2.cvtColor(frameROI,cv2.COLOR_BGR2GRAY)
     frameROI1=frame[0:380,0:480] #recorto una imagen más pequeña
     cv2.imshow('placaGray', frameROIgray)
     cv2.moveWindow('placaGray',0,360)
     cv2.imshow('recorte',frameROI1)

     cv2.moveWindow('recorte',640,0)
     cv2.imwrite("analisis.png",frameROIgray) #recorta la imagen de la placa para analisis
     if t_repeticion<1:
        cv2.imwrite("F_ROI.png",frameROI1) #recorte de placa para subir en la nube
        ParkPow_weebooks()

     if t_repeticion>=1:···


     else:···
    else:
        time.sleep(5)
        t_espera=t_espera+1
```

Código para la repetición del vehículo

```
    if t_repeticion>=1:
       placa2=reconocer_caracteres()
       x=convertir(datos.iloc[1])
       print(x)
       num_p2=verificador_placas(x)
       print(num_p2)
       ParkPow_weebooks()

       if num_p1==num_p2:
           print("mismo vehiculo")
           time.sleep(180)

       else:
           print("diferente vehiculo")
           t_espera=0

           t_repeticion=0
           placa1=[]
           placa2=[]


    else:

       placa1=reconocer_caracteres()
       x=convertir(datos.iloc[1])
       print(x)
       num_p1=verificador_placas(x)
       print(num_p1)
       print('')
       t_repeticion=t_repeticion+1
       time.sleep(30)
       #codigo para subir a la nube la placa
```

Código Función Park_Pow

```
def ParkPow_weebooks():
    CONTENT_TYPE_MULTIPART_FORM_DATA = 'multipart/form-data';
    uploadsDir = './uploads';
    var_User = 'german_martinez@outlook.com'
    const_http = 'https://app.parkpow.com/api/v1/create-camera/'
    var_contra = 'Ermelinda123'
    #intenta conectarse con el servidor
    server = http.createServer(('req', 'res') {
    if (req.method === 'POST') {
        collectRequestData(req, res);
         #se comunica con el servidor
         server.listen(8001, function(){
          console.log("Server listening on port 8001");
          # envia informacion
           if(!contentType){
                   response.end('OK!')
          }else if(contentType.indexOf(CONTENT_TYPE_MULTIPART_FORM_DATA)>-1){
                   filesParser(request, response, function (err)


    })
    } else {
        res.end(`Send a POST request negative.`);
    }
    })
```

Creación de la Red Neuronal Convolucional

```
from keras import Sequential
from keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatten, Dropout
from keras.layers import Input
from keras.optimizers import SGD

model = Sequential()

# Bloque 1
model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu', padding = 'same', name = 'block1_conv1', input_shape = (224, 224, 3)))
model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu', padding = 'same', name = 'block1_conv2'))
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), name = 'block1_pool'))
# Bloque 2
model.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu', padding = 'same', name = 'block2_conv1'))
model.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu', padding = 'same', name = 'block2_conv2'))
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), name = 'block2_pool'))
# Bloque 3
model.add(Conv2D(filters = 256, kernel_size = (3, 3), activation = 'relu', padding = 'same', name = 'block3_conv1'))
model.add(Conv2D(filters = 256, kernel_size = (3, 3), activation = 'relu', padding = 'same', name = 'block3_conv2'))
model.add(Conv2D(filters = 256, kernel_size = (3, 3), activation = 'relu', padding = 'same', name = 'block3_conv3'))
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), name = 'block3_pool'))

model.add(Flatten())
model.add(Dense(4096, activation = 'relu', name = 'fc1'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation = 'relu', name = 'fc2'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation = 'softmax', name = 'prediction'))
```