Plataforma de servicio para la identificación de placas vehiculares que permita la automatización del control de parqueaderos

Romero Gómez, Yury Ricardo

Departamento de Ciencias de la Computación

Carrera de Ingeniería de Sistemas e Informática

Trabajo de titulación, previo a la obtención del título de Ingeniero en Sistemas e Informática

Ph.D. Yoo, Sang Guun

27 de agosto de 2021

Curiginal

Document Information

Analyzed document	Yury Romero Tesis YOLO CLOUD VISION_Rev.docx (D111760560)
Submitted	8/27/2021 7:20:00 PM
Submitted by	
Submitter email	biblioteca@espe.edu.ec
Similarity	1%
Analysis address	ilbbioteca.GDC@analysis.urkund.com

Sources included in the report

W	URL: http://repositorio.espe.edu.ec/bitstream/21000/24469/1/T-ESPE-044514.pdf Fetched: 7/31/2021 8:42:30 PM	3
SA	**wilber mayta.pdf** Document wilber mayta.pdf (D98572225)	5
SA	**FARIAS-FATIMA_GUERRERO-JESSICA.docx** Document FARIAS-FATIMA_GUERRERO-JESSICA.docx (D47350266)	1
W	URL: https://repositorio.espe.edu.ec/bitstream/21000/22390/1/T-ESPE-043743.pdf Fetched: 5/29/2021 7:12:59 PM	2
W	URL: https://rua.ua.es/dspace/bitstream/10045/107576/1/Aplicacion_de_percepcion_mejorada_para_personas_con_Gama_Garcia_Angel_Manuel.pdf Fetched: 1/21/2021 12:43:56 PM	1

Firma:

SANG GUUN YOO .

PhD. Yoo, Sang Guun
C.I: 1306853720

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

CERTIFICACIÓN

Certifico que el trabajo de titulación, "**Plataforma de servicio para la identificación de placas vehiculares que permita la automatización del control de parqueaderos**" fue realizado por el señor **Yury Ricardo Romero Gómez** el cual ha sido revisado y analizado en su totalidad por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Sangolquí, 27 de agosto del 2021

Firma:

SANG GUUN YOO .

PhD. Yoo, Sang Guun
C.I: 1306853720

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

RESPONSABILIDAD DE AUTORÍA

Yo, **Yury Ricardo Romero Gómez,** con cédula de ciudadanía N°1724196322, declaro que el contenido, ideas y criterios del trabajo de titulación "**Plataforma de servicio para la identificación de placas vehiculares que permita la automatización del control de parqueaderos**" es de mi autoría y responsabilidad, cumpliendo con los requisitos legales, técnicos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Sangolquí, 22 de agosto del 2021

.....

Yury Ricardo Romero Gómez

C.I: 1724196322

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

AUTORIZACIÓN DE PUBLICACIÓN

Yo, **Yury Ricardo Romero Gómez,** con cédula de ciudadanía N°1724196322, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: "**Plataforma de servicio para la identificación de placas vehiculares que permita la automatización del control de parqueaderos**" en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Sangolquí, 22 de agosto del 2021

.....

Yury Ricardo Romero Gómez
C.I: 1724196322

DEDICATORIA

El presente trabajo de titulación representa la culminación de una etapa de mi vida, el esfuerzo, empeño y dedicación requerido para cumplir un objetivo; aun así, los logros no son nada si no se pueden alcanzar y celebrar junto a las personas que uno ama.

Dedico este trabajo en primera instancia a mis padres Yadira y Jaime, que desde pequeño me inculcaron la diciplina de estudiar y luchar por los sueños, por enseñarme a crecer y a que si caigo debo levantarme y continuar, por ser las bases que me ayudaron a llegar hasta aquí.

A Emilio y Camilo quienes supieron soportarme y brindarme su ayuda en incontables ocasiones, a mis tíos Pilar y Vladimir, siempre presentes a lo largo de esta travesía con consejos y lecciones de vida que sin duda las llevo conmigo día a día.

A mi novia Nathy Cristina por estar a mi lado en esta travesía, apoyarme en las circunstancias difíciles y compartir conmigo su inteligencia, tiempo, amor y cariño.

A mi tutor de tesis, opositora y a todos mis profesores a lo largo de estos años, por su arduo esfuerzo en trascender mediante sus enseñanzas y conocimientos.

Para finalizar, quiero compartir una frase que puede ayudar a muchos a seguir adelante con sus metas y sueños: “Ser más, para servir mejor”.

Yury Ricardo Romero Gómez.

AGRADECIMIENTO

Agradezco al PhD. Sang Guun Yoo, por el apoyo, guía y enseñanzas brindadas durante el desarrollo del trabajo de titulación, sin usted este logro no sería posible y espero algún día llegar a ser un increíble profesional como usted.

A mi familia por ser el pilar fundamental de mis convicciones, gracias por nunca perder la esperanza en mí. Al Area51 y Pordes puesto que la confianza, apoyo y vivencias compartidas me permitieron seguir adelante en los momentos más complicados.

A mis grandes amigos Kenneth, Fabricio, Adid, Angel, Steven, David y Tivi quienes siempre estuvieron para brindarme una mano y por aportar con su granito de arena a la solución trabajada.

A mi hermosa novia Nathy Heredia por corregir en múltiples ocasiones mi mala redacción, formato e incluso diseño en el presente trabajo. A mi mejor amiga Cristina Detken, por su apoyo, carisma, paciencia y sobre todo proveer los grupos de imágenes de entrenamiento utilizados.

Yury Ricardo Romero Gómez

Índice De Figuras

RESUMEN

Hoy en día, la falta de aplicación de nuevas tecnologías en distintos aspectos del control vehicular, junto al incremento de compra y uso de vehículos, acarrea una tarea enorme sobre el control de información de identificación vehicular. Se describe en este documento el proceso de desarrollo de un prototipo de software bajo la metodología incremental, el cual ofrece como solución la detección y reconocimiento de placas vehiculares ecuatorianas en tiempo real; permitiendo analizar el contenido del video y llevarlo al caso de aplicación de la normativa “Hoy no Circula”.

Con el uso de visión artificial y algoritmos de validación de placas vehiculares, se verifica si un vehículo puede o no circular dentro del área urbana de quito, antes de salir de las instalaciones de la Universidad de las Fuerzas Armadas ESPE.

Palabras clave:

- **VISIÓN POR COMPUTADORAS**
- **REDES NEURONALES CONVOLUCIONALES**
- **YOLO**
- **OCR**
- **RECONOCIMIENTO DE PLACAS**

ABSTRACT

Nowadays, the lack of application of new technologies in different aspects of vehicle control, added to the increase of in purchase and use of vehicles, generate an enormous task on the control and identification of vehicle transit information. This document describes the process of development of a software prototype under the incremental model methodology, which offers as a solution the detection and recognition of Ecuadorian licence plates in real time; allowing to analyze the content of a video and take it to the case of application of the regulation “Hoy no Circula”.

With the use of artificial vision and vehicle license plate validation algorithms, it is verified whether or not a vehicle can circulate within the urban area of Quito, before leaving the facilities of the Universidad de las Fuerzas Armadas ESPE

Keywords:

- **COMPUTER VISION**
- **CONVOLUTIONAL NEURAL NETWORK**
- **YOLO**
- **OCR**
- **VEHICLE LICENCE PLATE IDENTIFICATION**

Capítulo I

Introducción

Antecedentes

Para finales del año 2017, se presentó un incremento de 32 431 vehículos en la urbe de Quito, lo que generó un exceso de autos en circulación y distintos problemas de tráfico dentro de la ciudad. Un estudio realizado por Global Traffic Scorecard indicó que Quito se encuentra entre los 10 países con mayor congestión vehicular de América Latina (Ekos., 2019); esto ocasionó que varios usuarios presenten preocupaciones debido a problemas relacionados con la búsqueda de parqueaderos, como lo indica el comercio en una noticia publicada en el año 2019 (ElComercio., 2019).

Un censo realizado por el anuario de estadística de transporte (A.N.E.T.) reveló que desde el año 2008 hasta el 2018 se presentó un primer incremento del 7.5% anual en el parque automotor, es decir un total de 35000 vehículos más cada año (Lucero, 2020). Se produjo un crecimiento alarmante del 31% en el año 2018 con respecto al año anterior, cifra similar a la registrada en el año 2011 (ElUniversoEC., 2019). Con este afluente incremento vehicular se incurrió en la generación de distintos sistemas que permiten optimizar procesos de control vehicular como controladores de velocidad, controles matutinos y una forma de regulación vehicular para minorizar el impacto en el tráfico conocida como “Pico y Placa”, que debido a reformas de la alcaldía del Distrito Metropolitano de Quito a la presente fecha se conoce como “Hoy no circula”.

El crecimiento tecnológico ha generado un avance significativo en la automatización de procesos relacionados con el aparcamiento vehicular, desde detectores de zonas de parqueos, sensores de peso a las entradas de parqueaderos, hasta sistemas inteligentes conectados a aplicaciones que permiten tener comunicación directa con el usuario. Para inicios del año 2019 se anunciaba la llegada de un

aplicativo capaz de indicar a los usuarios las plazas disponibles en los distintos parqueaderos municipales de la ciudad, enfocándose en la automatización de reserva de espacios (ElComercio., 2019), sin priorizar la recolección de información de identificación vehicular.

En la ciudad capital, varios parqueaderos se encuentran en vías de automatización ante la ineficacia de los procesos de control actuales (ElComercio., 2019). Entre varios ejemplos, se puede mencionar el caso de los distintos centros comerciales, en los que se opta por el uso de expendedores de tickets, los cuales acarrean multas en caso de pérdida; por otro lado, también hay parqueaderos que realizan este proceso de forma manual (ElComercio., 2019).

Dentro del articulo realizado por Wichai Puarungroj y Narong Boonsirisumpun (2018), “Thai License Plate Recognition Based on Deep Learning” se define que a medida que el número de vehículos aumenta, la habilidad de llevar el control sobre el reconocimiento de placas vehiculares sobrepasa las capacidades humanas, por lo que un sistema de reconocimiento de placas vehiculares se vuelve imprescindible (Puarungroj & Boonsirisumpun, 2018). Es importante indicar que el sistema de reconocimiento de placas vehiculares consta de dos subsistemas:

- La detección de placa: busca la ubicación de la placa vehicular dentro de la imagen.
- El reconocimiento de la placa: reconoce los caracteres de la placa vehicular encontrada.

Durante los últimos años el uso de sistemas de reconocimiento de placas vehiculares o LPR (Licence Plate Recognition) en sus siglas en inglés, resultó ser de gran ayuda para la generación de una gran variedad de aplicaciones en áreas del día a

día como: seguridad vial, control de parqueo automático, cobro automático de peajes y radares de velocidad.

Planteamiento Del Problema

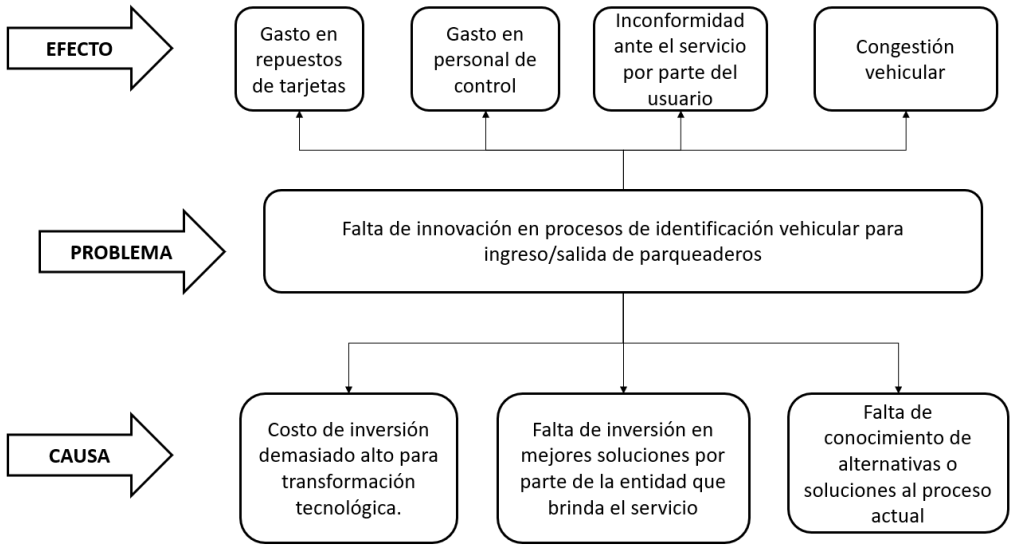
En la actualidad, el uso de vehículos ha aumentado considerablemente, lo que produce diversos problemas de movilidad dentro de la ciudad de Quito (ElUniversoEC., 2019). Por esta razón se requiere encontrar un proceso de identificación vehicular óptimo y fiable, que permita automatizar el control de entrada y salida de vehículos en parqueaderos. En Quito, el único sistema automatizado implementado es aquel que controla la velocidad vehicular mediante distintos sensores. Dentro del parqueadero de la Universidad de las Fuerzas Armadas ESPE, el procedimiento para identificación de clientes se realiza mediante el uso de SmartCard's (Tarjeta inteligente), las cuales utilizan la tecnología Near Field Communication (NFC), permiten el ingreso o la salida de vehículos.

El uso de SmartCard's solventa la necesidad de una identificación para el acceso al servicio, pero deslinda la verificación de la correspondencia entre cliente y vehículo; en los casos en que el usuario olvide su tarjeta o la extravíe, debe proceder a llamar al personal del parqueadero y solicitar el ingreso, lo que podría generar cierta congestión vehicular en horas pico. Adicionalmente, el costo de los equipos y tarjetas puede llegar a ser elevado.

A continuación, en la **Figura 1** se representa el árbol de problemas sobre el tema planteado con sus respectivas causas y efectos:

Figura 1.

Árbol de problemas



Como se aprecia en la **Figura 1**, las causas de la falta de innovación en procesos de identificación vehicular para el ingreso y salida de vehículos de parqueaderos son: el desconocimiento de soluciones o alternativas al proceso actual, debido a la poca atención que se presta a la falta de control de identificación vehicular dentro de la urbe de Quito; otra causa es el temor a los altos costos de inversión para transformación tecnológica. Finalmente, tenemos la falta de inversión en nuevas soluciones por parte de la empresa que brinda el servicio, ya que no mantienen interés en la recolección de información fiable. Sumado al problema central, esto ocasiona la pérdida de información vital de identificación vehicular, gastos en personal humano y problemas en el manejo adecuado de los procesos inmersos en el flujo de control vehicular.

Justificación

Debido al crecimiento de la urbe de Quito, poseer y manejar un automóvil se ha convertido en algo común en el día a día de muchos de los residentes. Un estudio realizado por Global Traffic Scorecard indicó que Quito se encuentra entre las 10 ciudades latinoamericanas con mayor congestión vehicular de Latinoamérica (Ekos., 2019) y una de las causas de la congestión vehicular es el tiempo que acarrea los conductores en encontrar e ingresar a los parqueaderos. Ante esta situación, mediante el uso de distintos avances tecnológicos, se ha originado diversas formas para la identificación de vehículos, con la finalidad de agilitar el proceso de ingreso a los parqueaderos.

En muchos parqueaderos de la capital, se hace uso de un lector de tarjetas NFC para permitir el acceso a vehículos de usuarios que poseen este medio, lo que supone algunos problemas en caso de olvido o pérdida; de la misma forma, no puede cederse a invitados, puesto que acarrearía costos adicionales para el usuario y la empresa. En vista de ello, resulta fundamental que el usuario se coloque a una distancia apropiada para poder identificarse, lo que podría ocasionar accidentes con los equipos inmersos.

Este trabajo de titulación pretende impulsar el uso de tecnologías modernas como la visión por computadoras e inteligencia artificial, con el objetivo de evitar los inconvenientes que se presentan al utilizar tarjetas NFC, garantizando la fiabilidad y consistencia de la información obtenida.

El presente documento se centra en el uso de redes neuronales junto con la diciplina de visión por computadores, para solventar el problema expuesto en la **Figura 1**, acarreando beneficios como:

- Rápidos tiempo de respuesta del análisis de la placa vehicular.

- Consistencia e integridad de la información obtenida
- Disminución de gastos de personal operativo.
- Disminución de gastos en mantenimiento de equipos NFC.

Objetivos

Objetivo General

Desarrollar un prototipo de software aplicando visión artificial para brindar el servicio de identificación de placas vehiculares y validar la normativa “Hoy no Circula” que rige en la zona urbana de Quito, en la Universidad de las Fuerzas Armadas ESPE.

Objetivos Específicos

i. Realizar un estudio del estado del arte, para entender la situación actual de la visión por computadoras y su campo de aplicación en reconocimiento de texto.
ii. Realizar un estudio de los métodos de identificación de placas vehiculares.
iii. Desarrollar un prototipo de software para identificar placas vehiculares aplicando diferentes algoritmos.
iv. Evaluar la funcionalidad y rendimiento del prototipo de software a partir de distintas imágenes de placas vehiculares capturadas en el campus matriz de la Universidad de las Fuerzas Armadas ESPE.
v. Analizar los resultados obtenidos en las pruebas de validación.

Alcance

Desarrollar un prototipo de software que permita identificar las placas con el formato de tres letras seguidas de cuatro o tres números (XXX-### o XXX-####) y fondo blanco, al solicitar la respectiva imagen del vehículo y devolver su número de placa. No se consideran dentro de este análisis placas de motocicletas.

Para la validación del prototipo de software se utilizará imágenes de placas vehiculares del campus matriz de la Universidad de las Fuerzas Armadas ESPE con un formato de 720p. Este aplicativo estará construido mediante el uso de visión por computadoras y redes neuronales convolucionales.

Capítulo 2

Marco Teórico

inteligencia Artificial

Se define a la Inteligencia Artificial (IA) como la capacidad de una máquina para usar algoritmos que simulen funciones cognitivas como el aprendizaje y la toma de decisiones mediante el análisis y procesamiento de la información proporcionada (Rouhiainen, 2018). En este sentido, las metodologías deben ser procesos autónomos capaces de minimizar aquellos errores que un humano podría cometer.

De acuerdo con Delgado (1997), para alcanzar los objetivos y el carácter deseado de una IA, se debe contar con un ordenador capaz de simular conductas y tareas humanas para lo cual es necesario un estudio de la naturaleza tanto de la mente humana como del ordenador.

Algunos de los campos de aplicación de la IA son (Dennet, 1984):

- El reconocimiento de imágenes, clasificación y etiquetado.
- El mejoramiento del desempeño de la estrategia algorítmica comercial.
- Procesamiento eficiente y estable de datos de pacientes.
- El mantenimiento predictivo.
- La distribución de contenido en redes sociales.
- La protección de amenazas de seguridad cibernética.

Esta extensa aplicación hace necesario que una IA sea capaz de resolver el problema de formalizar una solución del conocimiento que maneja, lo que se define como “The Frame Problem” (Dennet, 1984).

Aziz F. Zambak (2013) refuta: “Sí, pero nunca conseguirás que una maquina haga X” en su estudio *“The Frame Problem – Autonomy Approach VS Designer Approach”*. En el que “X” es el problema principal por resolver, como detectar objetos, reconocer caracteres, autonomía, etc. Esto se debe a la búsqueda de una forma de enfrentarse a un problema con condiciones altamente cambiantes. Por ejemplo, en el caso de variables en un entorno real, el uso de datos semánticos permite al analista definir un trasfondo de la información con la que la IA se alimentará, entenderá y definirá sus propias reglas para realizar la toma de una decisión basada en un modelo lógico que entregue sentido a su respuesta (Dennet, 1984).

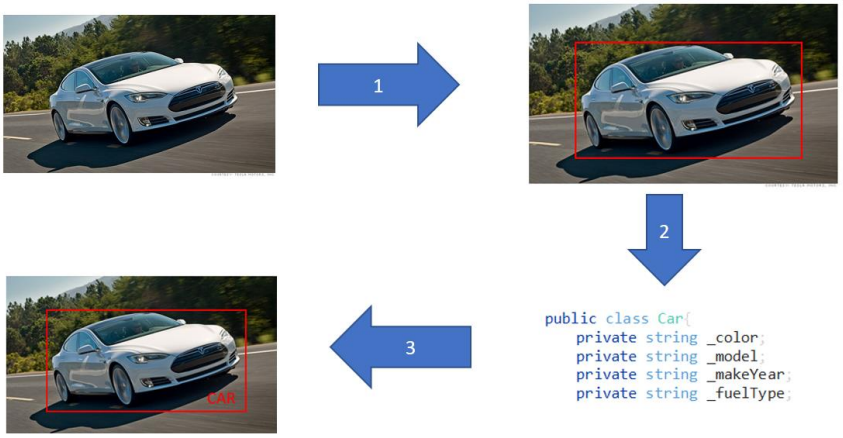
Visión Por Computadora

La visión por computadora es un campo de estudio de la inteligencia artificial, que permite a los ordenadores y sistemas obtener información significativa de imágenes digitales, videos u otras fuentes visuales, y tomar acciones o hacer recomendaciones sobre la información recibida (IBM, 2016). En la actualidad, los avances conseguidos en el desarrollo de la visión por computadora han permitido crear nuevas diciplinas como el procesamiento de documentos o texto, sensores remotos, radiología, microscopía, inspecciones industriales y guía de robots (Rosenfeld, 1988).

La estructura de un sistema de reconocimiento de objetos mediante imágenes estáticas se describe en la **Figura 2**, donde se observa la entrada de una imagen de un vehículo, el cual pasará por un proceso de (1) segmentación, (2) etiquetado, (3) emparejamiento de modelo (P.O.O.), cuyo resultado será el reconocimiento y la descripción del objeto.

Figura 2.

Paradigma de análisis de imágenes 2D



Machine Y Deep Learning

Machine Learning es una rama de la IA que utiliza algoritmos que permiten a los ordenadores realizar distintas tareas como identificación de objetos, transformación de imágenes a texto o unión de intereses de los usuarios en redes sociales; aun así, esta tecnología se volvió limitada al momento de capturar información en formato “crudo”, puesto que estos sistemas requieren cierto nivel de experticia en el diseño de funciones, para la interpretación o clasificación (Bismart, 2020).

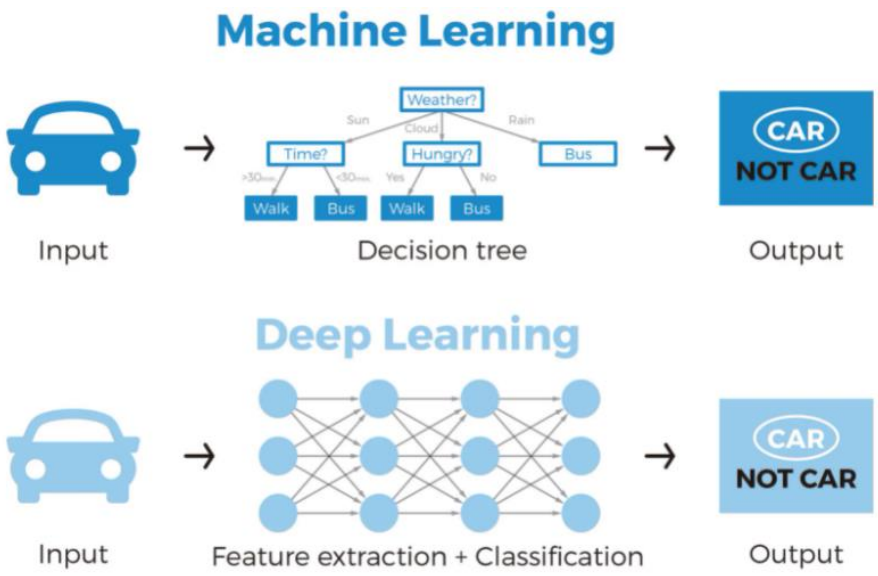
En este punto se desarrollan técnicas de aprendizaje profundo conocido como “*Deep Learning*”, el cual hace uso de redes neuronales, que permiten al ordenador tomar información y automáticamente reconocer la representación necesaria para su detección (LeCun, Bengio, & Hinton, 2015).

Ambos casos permiten el aprendizaje supervisado y no supervisado, con la particularidad que en un conjunto de datos no supervisado se conglomera como una agrupación de información no etiquetada y un set supervisado le otorga la facilidad de

datos de entrada al contener elementos etiquetados (Bismart, 2020); esto permite la extracción y clasificación de información como se ejemplifica en la **Figura 3**:

Figura 3.

Machine Learning vs Deep Learning



Nota: Recuperado de ¿Cuál es la diferencia entre el machine learning y el deep learning? Copyrigth 2021 por Bismart Blog.

Li Deng y Dong Yu (2014) en su monografía “Deep Learning: Methods and Applications” exponen que Deep Learning es una nueva área del Machine Learning, el cual lo impulsa a alcanzar su objetivo general; es decir, la Inteligencia Artificial, con la ayuda de múltiples niveles de representación y abstracción, permite darle sentido a la información como imágenes, sonidos o textos.

Hoy en día, distintas técnicas de Deep Learning han mejorado la capacidad de clasificar, reconocer, detectar palabras, detectar imágenes o describir objetos; varias de estas novedades están integrando avances como los expuestos a continuación (Deng & Yu, 2014):

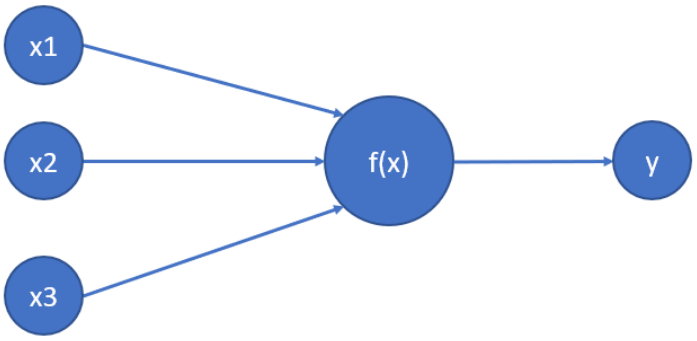
- Mejoras algorítmicas han elevado el desempeño de los métodos de aprendizaje profundo.
- Nuevos métodos de aprendizaje basado en máquina han mejorado la precisión de los modelos.
- Se han desarrollado nuevas clases de redes neurales, que encajan bien en aplicaciones como la traducción de texto y la clasificación de imágenes.
- Tenemos muchos más datos disponibles para construir redes neurales con muchas capas profundas, incluyendo datos de "*streaming*" de la Internet de las Cosas, datos textuales de medios sociales, notas de médicos y transcripciones de investigaciones.
- Los adelantos computacionales en la nube distribuida y unidades de procesamiento gráfico han puesto a nuestra disposición una gran capacidad de cómputo.

Redes Neuronales Convolucionales

La CNN (por sus siglas en inglés, "*Convolutional Neural Network*") es un tipo de red neuronal que hace uso de aprendizaje supervisado y una variación de un perceptrón multicapa, para tomar imágenes de entrada y devolver una detección dentro de la imagen; esta imita el córtex visual del ojo humano. Se puede describir de forma simple como una red neuronal artificial, la cual realiza varias copias de una misma neurona artificial, permitiéndole estar compuesta de varias neuronas y expresar computacionalmente grandes modelos, con un número relativamente bajo de parámetros (Olah, 2014). Las neuronas artificiales se modelan mediante unidades de proceso; cada unidad de proceso se compone de entradas, un cuerpo de procesamiento y una salida, como se muestra en la **Figura 4.**

Figura 4.

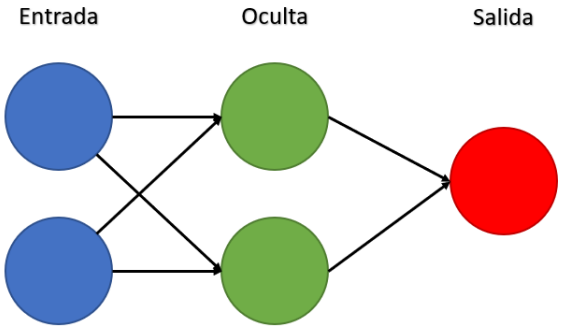
Bloque de construcción básico de la red neuronal artificial.



En el caso de las redes neuronales convolucionales, dicho cuerpo de procesamiento está definido por una función de activación conocida como ReLU (siglas del inglés, "*Rectified Linear Unit*"). Estos nodos de tipo neuronal están conectados de manera que reciben la entrada de uno de los nodos y la envían a otros nodos formando una malla como se muestra en la **Figura 5** (Cohen, 2018)**.**

Figura 5.

Red Neuronal Básica

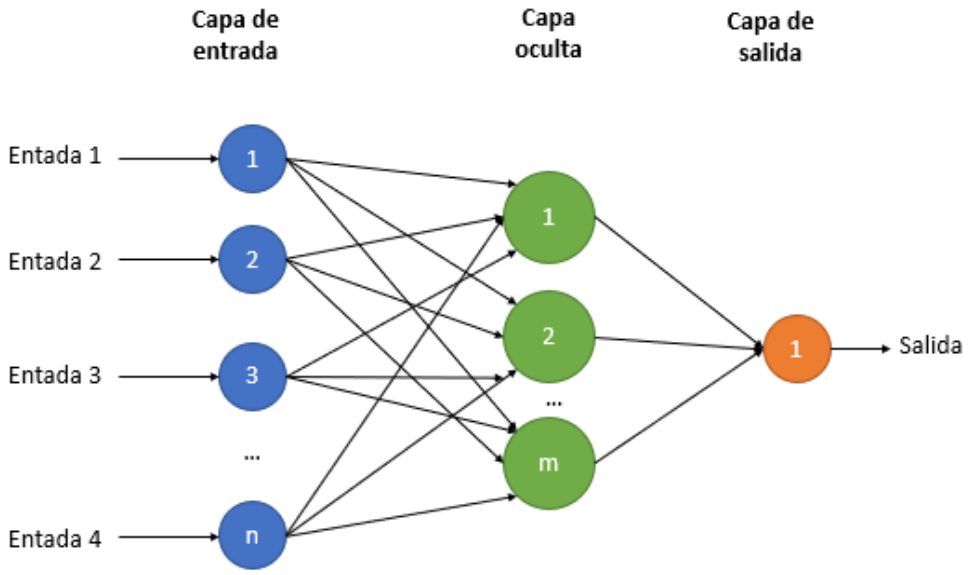


Cuando varios nodos se agrupan se forma una Red Neuronal Artificial. Estas redes neuronales pueden llegar a ser extremadamente grandes y poseer miles de millones de parámetros, con millones de nodos y conexiones intermedias entre ellas.

Las redes neuronales se clasifican de acuerdo con el tipo de conectividad entre sus elementos (neuronas individuales). En las redes neuronales "*feedforward*" (en cascada), las conexiones se dirigen siempre en dirección de las neuronas de la capa de salida; mientras que en las redes neuronales "*feedback*", se puede detectar conexiones de ida y retorno entre capas de neuronas. Un modelo de redes neuronales "*feedforward*" se presenta en la **Figura 6** (Mishra, 2019):

Figura 6.

Red Neuronal en cascada de tres capas

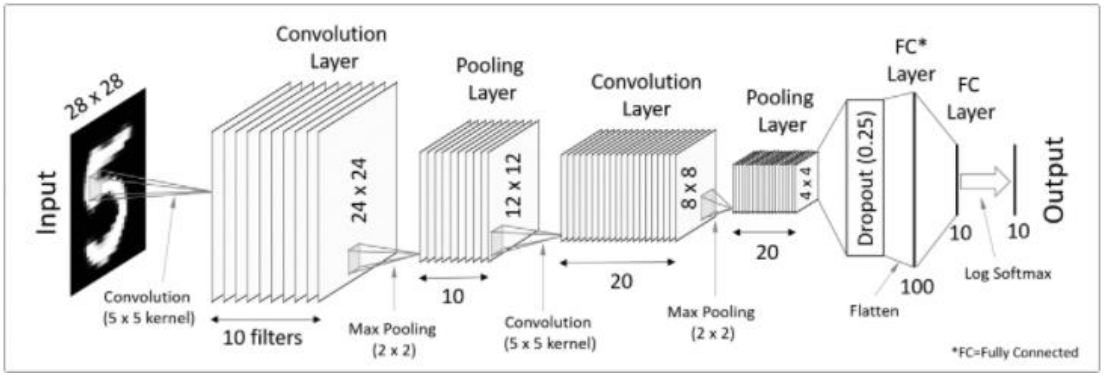


Convolucional (Convolutional Newral Network). En una red neuronal de convolución, cada neurona procesa los datos solo de su área de análisis y los organiza de tal manera que representan la imagen completa. Además, tanto el sistema visual biológico como la red neuronal de convolución tienen una jerarquía de capas presentadas en la **Figura 7,** las cuales extraen progresivamente más y más características. Es decir que en la primera capa de una CNN se podrán identificar bordes, líneas o curvas, y a medida que se profundiza en niveles las capas tienen la

capacidad de detectar personas, objetos, rostros o cosas más complejas (Mishra, 2019).

Figura 7.

Arquitectura de una red neuronal de convolución



Nota: Recuperado de Getting Started with Pytorchfor Deep Learning Copyrigth 2017 por la compañia Code to light.

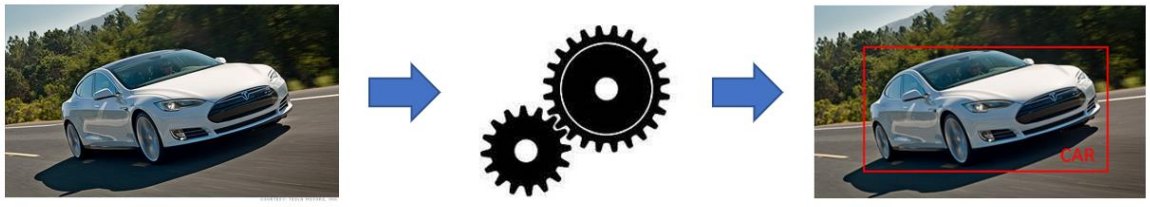
Una red neuronal de convolución maneja varias capas, entre ellas (Saha, 2018):

- **Capa convolucional (Conv.):** Se encarga de extraer las características espaciales y temporales a partir del uso de filtros; en el proceso reduce el tamaño de la imagen a través de la operación de convolución de matrices sobre la entrada, lo que da como resultado una matriz de menor tamaño a la siguiente capa.
- **Capa de reducción (Pooling):** Reduce el tamaño espacial de la salida producida por la capa convolucional, para disminuir el tiempo de cómputo de procesamiento y detección de información alto nivel.
- **Capa totalmente conectada (FC):** Conecta cada nodo de las capas convolucionales y de reducción, con el propósito de extraer información y clasificarla acorde a un método de retro propagación.

En la visión por computadoras, se ingresa la imagen con la que se alimenta a la red neuronal como entrada, con la intención de que el ordenador analice e infiera sobre ella, como la secuencia mostrada en la **Figura 8***:*

Figura 8.

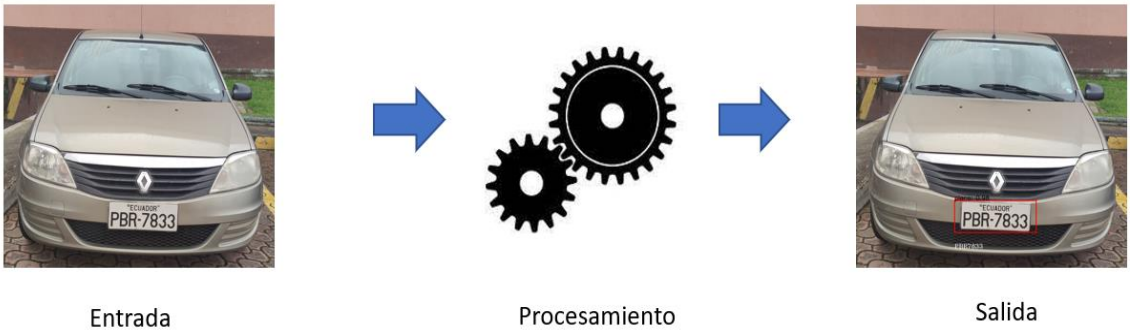
Flujo de proceso de visión por computadoras.



Para este proyecto, se plantea que el prototipo utilice la visión por computadoras y una red neuronal convolucional de manera que, a partir de una entrada, que para este caso es una imagen en tiempo real, aprenda a identificar las placas vehiculares, para obtener información precisa de la identificación del vehículo. El flujo de proceso se describe en la **Figura 9***.*

Figura 9.

Flujo de proceso del prototipo de Software



Arquitecturas de Redes Neuronales Convolucionales

Algunas de las arquitecturas de redes neuronales convolucionales más utilizadas son las siguientes (Moreno & Lara, 2020):

- LeNet: Presentada por LeCun y colaboradores (1998) para el reconocimiento de caracteres de texto en documentos; en su primera versión usaba la función de activación TANH.
- AlexNet: Presentada por Krizhevsky y colaboradores (2012) para el reconocimiento de objetos en fotografías, introdujo el uso de una función de activación ReLU y Softmax, al igual que la aplicación de máxima reducción, en lugar de reducción media y la aplicación de capas convolucionales continuas.
- VGGNet: Presentada por Simonyan y Zisserman (2015) para reconocimiento de objetos sobre imágenes de gran escala, introdujo el uso de varios filtros pequeños a fin de conseguir resultados precisos.
- RestNet: Presentada por He y colaboradores (2015), hace uso de conexiones residuales para conectar capas no consecutivas, saltando capas intermedias.

En el presente proyecto se hace uso de la arquitectura ResNet implementada en Y.O.L.O.

Y.O.L.O (You Only Look Once)

Y.O.L.O. es una red entrenada para la predicción de los límites de un objeto dentro de un cuadro y su respectiva “Clase”; es una red rápida, capaz de procesar imágenes en tiempo real a 30 fotogramas por segundo. Existe a su vez una versión simplificada conocida como Fast-Y.O.L.O., la cual puede procesar la información a una velocidad de 120 fotogramas por segundo (Redmon, Divvala, Girshick, & Farhadi, 2016).

La clave de la optimización, fiabilidad y velocidad de Y.O.L.O. se encuentra en la unificación de los componentes separados de detección de objetos en una única red neuronal, que evoca un proceso simple de uso descrito en la **Figura 10**. Y.O.L.O. se basa principalmente en la predicción de límites de cajas y clases a los que los objetos pertenecen (Redmon, Divvala, Girshick, & Farhadi, 2016).

Figura 10.

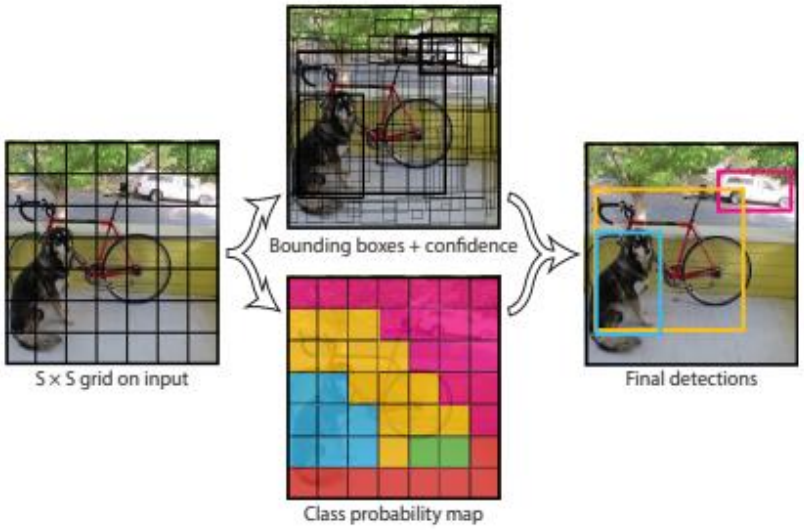
Sistema de detección YOLO

Nota: Recuperado de You Only Look Once: Unified, Real-Time object Detection. Copyrigth 2016 por Redmon, Divvala, Girshick & Farhadi.

Para realizar esta predicción YOLO cambia el tamaño de la imagen a uno prestablecido por el archivo de configuración. YOLO toma como parámetros el alto, ancho y profundidad de la imagen (Cantidad de canales. Ejemplo: RGB = 3 canales), cambia de redimensiona las entradas por un factor de 32, 16 y 8. Este proceso infiere en cada una de ellas y pasa por un filtro final conocido como supresión de no-máximos, que evita múltiples detecciones sobre un mismo objeto y presenta el resultado con un porcentaje de confidencia, valor que representa la intersección sobre unión entre las cajas predichas y cualquier caja real que haya sido representada con anterioridad. En la **Figura 11** se visualiza el flujo de detección de Y.O.L.O. después de la aplicación del método de supresión de no-máximos (Redmon, Divvala, Girshick, & Farhadi, 2016).

Figura 11.

Modelo de predicción de cajas



Nota: Recuperado de You Only Look Once: Unified, Real-Time object Detection. Copyrigth 2016 por Redmon, Divvala, Girshick & Farhadi.

Data Augmentation

Con el fin de obtener mejores resultados sobre el entrenamiento de las redes neuronales, se aplica un conjunto de transformaciones aleatorias sobre las imágenes o entradas originales del conjunto de datos de entrenamiento; ya que la red neuronal puede acostumbrase a la imagen original con facilidad después de un cierto número de iteraciones, por lo que, al obtener variantes aleatorias de la misma imagen, aumenta la cantidad de información y permite a la red generalizar de una mejor manera las detecciones. Dentro del mercado existen varias soluciones, así como librerías de uso gratuito que permiten optimizar el proceso de aumento de la información, como lo son RoboFlow u OpenCV (Gandhi, 2020).

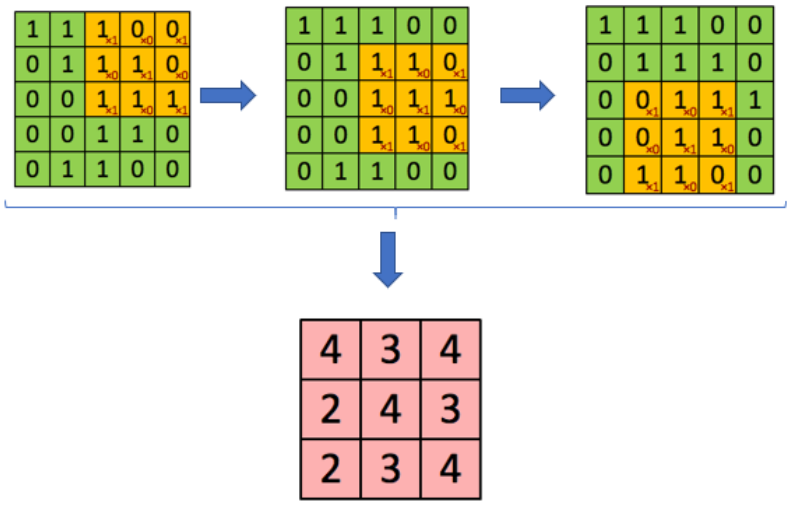
OpenCV

OpenCV comenzó como un proyecto de investigación de Intel en 1999. Hoy en día se ha convertido en la librería más grande de visión por computadora "*open source*" disponible en el mercado. Nació de la necesidad de disponer de un paquete de herramientas para visión y procesamiento de imágenes flexible para alto nivel; está escrito en C++ y maneja interfaces en lenguajes como Python, C++ y Java, lo que permite su sencilla integración con varios sistemas, a la vez que despliega el uso de sus más de 2500 distintos algoritmos, para tratamiento de imágenes y video, entre ellos algunos con manejo de Machine Learning y visión artificial (Garcia, 2013).

Dentro de visión artificial se define el concepto de un núcleo (Kernel), el cual es una matriz utilizada para realizar operaciones matemáticas sobre cada píxel de la imagen, que toma en cuenta el ancho y la altura relativa de la imagen original, así como un punto de anclaje para moverse alrededor de ella, y así generar una convolución. Esto se aprecia en la **Figura 12** (OpenCV, 2021).

Figura 12.

Convolución sobre una imagen



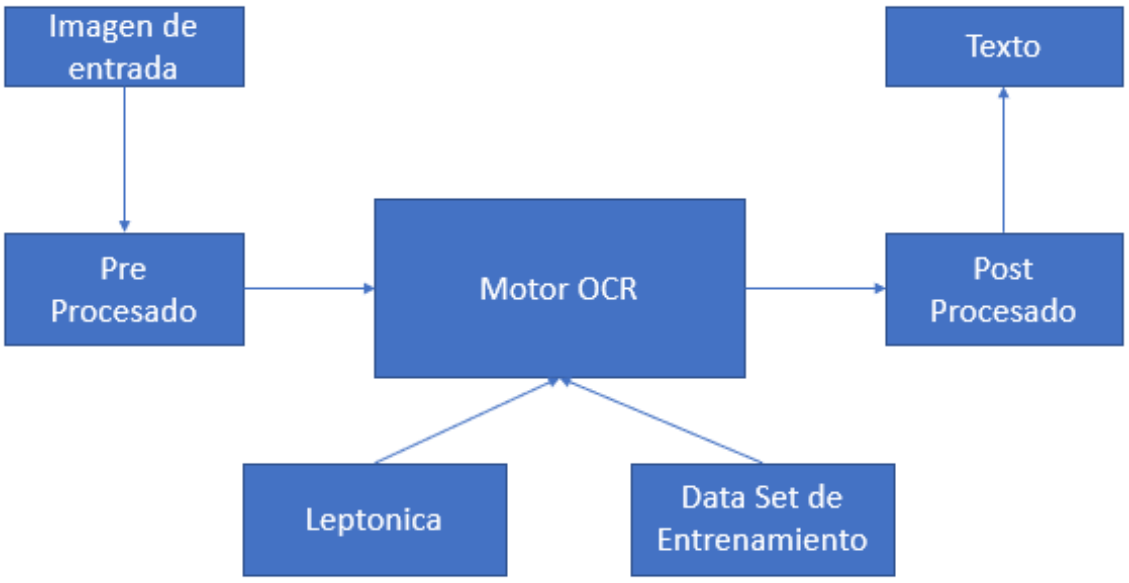
Es mediante este proceso que OpenCV permite aplicar distintos filtros a la imagen a fin de detectar contornos, bordes u objetos y diferenciarlos de su fondo. Dentro de este proceso existen algoritmos que permiten cambiar el espacio de color y remover el fondo de una imagen, para que los procesos de "*Machine Learning*" reconozcan con mayor facilidad patrones o rasgos que mejoren sus resultados en procesos de detección de objetos (Akshayan, Vishnu Prashad, Soundarya, Malarvezhi, & Dayana, 2018).

Reconocimiento Óptico De Caracteres (OCR)

Un OCR es un sistema que permite realizar reconocimiento de texto en imágenes, documentos escaneados o texto escrito a mano. A lo largo del tiempo se han presentado distintas soluciones y motores OCR que no requieren de entrenamiento y trabajan con varios tipos de fuentes (Tech target Contributor, 2019). La **Figura 13** presentada a continuación, describe el flujo de proceso de un OCR de forma general.

Figura 13.

Flujo de proceso de un OCR



Para el pre procesado de una imagen, un OCR utiliza algunas técnicas como las descritas a continuación (Reddy, 2019):

- Ordenar las líneas del documento de forma horizontal.
- Limpiar las impurezas en la imagen.
- Cambiar la imagen de una escala de colores o gris a blanco y negro.
- Limpiar símbolos no reconocidos.
- Analizar el lienzo para la identificación de párrafos o columnas.
- Detectar las palabras y líneas / Reconocer el texto / Segmentar los caracteres
- Normalizar las escalas

Al momento existen varias soluciones OCR como Tessetract, EasyOCR o Google Cloud Vision API.

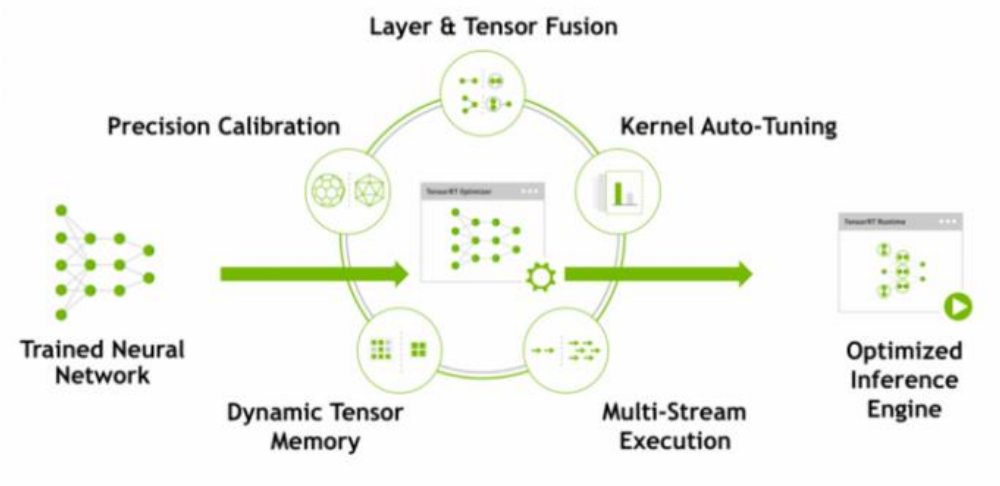
Tensorflow

Tensorflow es un sistema de código abierto desarrollado por el equipo de Google Brain, con la finalidad de ayudar a mejorar procesos de aprendizaje automático, construir redes neuronales y facilitar su implementación a partir de interfaces en dispositivos que no cuentan con propiedades para procesamiento de datos. Dicho sistema deriva las tareas de sus redes neuronales sobre matrices de datos o arreglos multidimensionales, los cuales se conocen como Tensores (Buhigas, 2018).

Esta solución posee librerías en Python, las cuales son capaces de realizar transformaciones de modelos entrenados en otras redes; con el uso de TensorRT, el cual, gracias a sus múltiples optimizaciones presentadas en la **Figura 14**, efectiviza procesos de detección y cómputo de modelos entrenados en redes neuronales, que mejora el tiempo de respuesta hasta en un 50% (Chaturvedi A. , 2020).

Figura 14.

Tipos de optimizaciones de TensorRT



Nota: Recuperado de Understanding Nvidia TensoRT for deep learning model optimization. Copyrigth 2020 por la compañia Medium.

Microservicios

La arquitectura de microservicios permite diseñar aplicaciones, de forma que cada componente funcione autónomamente sin afectar otro. Cada elemento se enfoca en la resolución de un problema específico y, de requerir comunicación entre los mismos, esta se realiza mediante el consumo de API's, lo que permite crear aplicaciones escalables en tiempos relativamente cortos si son administrados, coordinados y gestionados de manera adecuada.

Capítulo 3

Estado del Arte

Para realizar el análisis del estado del arte acerca del uso de visión artificial para detección de placas se realizó un proceso de revisión de literatura preliminar como se indica a continuación:

Planteamiento del estudio sistemático de literatura

En la fase inicial del desarrollo, se realizó una descripción del problema central de investigación, a fin de facilitar la búsqueda de estudios científicos, siguiendo con la definición de objetivos y la especificación de los criterios de inclusión y exclusión.

Definición del grupo de control y extracción de términos

El grupo de control se conformó por una persona, luego de realizar el análisis de varios estudios científicos; se logró la selección de los artículos descritos a continuación:

- System for Monitoring and Guarding Vehicles on Parking Areas. (Nemec, Janota, & Rastislav, 2017)
    - Palabras Clave: Monitoring, Guarding, Vehicle ICT device, Parking area.
- Image Analysis and Computer Vision with OpenCV. (Neli, 2018)
    - Palabras Clave: OpenCV, Phyton.
- LPRNet: License Plate Recognition via Deep Neural Networks. (Zherzdev & Gruzdex, 2018)
    - Palabras Clave: License plate recognition, Vehicle identification, Deep Neural Networks.

- Quality Assessment for a Licence Plate Recognition Task Based on a Video Streamed in Limited Networking Conditions. (Leszczuk, Janowski, Romaniak, Glowacz, & Mirek, 2011)
    - Palabras Clave: Quality of experience, content distribution, real time(live) content distribution.

Una vez culminado el análisis de estudio, se seleccionaron las palabras claves más importantes con respecto al objetivo de la búsqueda: computer visión, image detection, licence plate identification, monitoring, open cv, vehicle identification.

Construcción de la cadena de búsqueda

Identificados los términos, en esta fase se procede a crear y probar las posibles cadenas de búsqueda en la base digital escogida, para este caso SpringerLink. Se construyó una cadena inicial como se muestra a continuación:

ALL (“Computer Vision” OR “Image Detection”) AND ALL (“licence plate identification” OR “vehicle identification”) AND ALL (“Monitoring”)

Selección de los estudios primarios

Al aplicar la cadena de búsqueda en la base digital SpringerLink se obtuvo un total de 87 artículos relacionados al tema. Se aplicaron los siguientes filtros adicionales de manera que los estudios candidatos sean válidos al ser considerados como parte del estado de la temática:

- Año: Se tomaron artículos de las fechas entre 2013 - 2019.
- Tipo de documento: Se eligieron los estudios de tipo conference paper.

Con estos filtros se presentó un número menor de artículos científicos, de los cuales se seleccionaron cinco como estudios primarios:

- **Feature Based Multiple Vehicle License Plate Detection and Video Based Traffic Counting** (Chithra & Prashanthi, 2017):Los autores crean un sistema capaz de reconocer vehículos en videos mediante técnicas de movimiento para realizar un conteo vehicular a fin de monitorear el tráfico, se usaron clasificadores de cascada para realizar la detección de vehículos con un porcentaje de `precisión del 97.2% y a esto se suman técnicas de extracción de placas, con la aplicación de una estrategia de predicción, que busca la coincidencia más cercana para mejorar la eficiencia del OCR, obteniendo resultados de un 99% de exactitud en imágenes con fondos complejos.
- **Building a Character Recognition System for Vehicle Applications** (Bansal, Gupta, & Tyago, 2019): En este paper se intenta proponer una metodología para identificar placas tanto en vehículos como en motocicletas en la India utilizando un CCTV(Closed circuit Television Videocamera) para beneficiar el proceso de extracción de texto con el OCR, también se plantea el uso de cámaras en distintas ubicaciones, a fin de que todas puedan capturar la imagen de las placas de un vehículo para conocer la ruta en la que se ha encontrado dicho vehículo, al momento de tomar una fotografía, esta se compara con un compendio de imágenes de las mismas placas, se proponen distintas aplicaciones como control de parqueadero, tratamiento de ley, control de velocidad, peajes automáticos, etc.
- **Review Paper: Licence Plate and Car Model Recognition** (Akshayan, Vishnu Prashad, Soundarya, Malarvezhi, & Dayana, 2018): El enfoque de este paper es analizar los numerosos métodos que existen para el reconocimiento de placas de vehículos, el análisis se realiza mediante un estudio exhaustivo de estos métodos (lectura OCR, delimitación y predicción de placas) y se determinó que la manera más eficiente seria generar un API para proveer respuestas precisas, y que en su efecto este anexada a una base de datos para soportar varias soluciones o

aplicaciones, de esta forma se provee una solución fácil de programar y aplicable a diferentes casos.

- **Smart Toll Collection Using Automatic License Plate Recognition Techniques** (Mahalakshmi & Sendhil, 2018): Debido al crecimiento urbano de la India, el uso de vehículos se ha vuelto una necesidad creando un problema respecto al control de vehículos y monitoreo de tráfico, los parqueaderos se han sobrellenado y la gestión de toda esta información pasa desapercibida. Por ende, un ANPR (Automatic Number Plate Recognition System) juega un rol importante en solventar estos problemas por lo que se opta por realizar un análisis de las diferentes tecnologías e implementar un sistema basado en OCR.
- **License Plate Detection and Recognition in Unconstrained Scenarios** (Silca & Jung, 2018): Este paper trata de mejorar la detección de placas con el uso de una red neuronal convolucional, a fin de detectar cualquier placa a pesar de las imperfecciones o distorsiones de ángulo dentro de distintas fotografías, con el fin de acercarse más al nivel de software comercial, y minimizar la brecha entre las distintas técnicas de nivel académico y comercial.

Técnicas de extracción de caracteres para placas vehiculares

- **Reconocimiento óptico de caracteres**: Este proceso involucra la captura y extraccion del segmento de la placa vehicular, para proceder a hacer una segmentación de caracteres y realizar el reconocimiento individual de cada letra. Dentro de este método, se requiere hacer uso de machine learning para un mejor reconocimiento de la placa
- **Método delimitador**: Este proceso confia todo su analsis en el reconocimiento de bordes, dentro del metodo de reconocimiento, lo que reduce tiempos de procesamiento de imagen, sin embargo, no puede ser utilizado en procesos

complejos de reconocimiento de placas, ya que pueden presentar múltiples bordes lo que puede llevar a la confusión de deteccion de la placa.

- **Proceso de extracción para caracteres:** Este sistema basa su eficiencia en enmascarar los componentes no deseados de la imagen mediante la camara. La calidad de la imagen se mejora con procesos con OpenCV, mediante el uso del umbral adaptativo para segmentacion.

Características del estado del Arte

Los estudios antes mencionados demuestran que la factibilidad de la visión por computadoras, sumado a alguna técnica para el reconocimiento de placas en tiempo real, efectivizan su uso, entre las cuales destacan procesos anexados como la consideración de ángulos oblicuos con métodos para comprender e identificar placas en estos casos, aumentando la fiabilidad de la información en casos “extremos”, sin embargo este estudio tiene como finalidad hacer uso de redes neuronales para mejorar la eficiencia de estos algoritmos de visión por computadora, con la finalidad de obtener información fiable y precisa.

En Ecuador, debido a la carencia de la aplicación de redes neuronales en el tema de control de identificación vehicular, hace necesaria la búsqueda de ampliar dichos conocimientos, también se eliminan las limitaciones de los trabajos antes presentados; como la ejecución de los procesos de forma local, al proveer una solución web que permita evidenciar la eficiencia del algoritmo y la red neuronal.

Capítulo 4

Desarrollo de la Solución Propuesta

Metodología

Para el desarrollo del presente trabajo de titulación se ha utilizado la metodología de desarrollo incremental, la cual hace uso del modelo en cascada, que resulta en la mejora del software con cada iteración planteada. A causa de su enfoque en construcción de prototipos, se optan por los beneficios otorgados por esta metodología, tales como:

- Funcionalidad en las primeras iteraciones del proyecto.
- Es un modelo flexible, por lo que se reduce el coste con el cambio de requerimientos.
- Permite gestionar la complejidad del proyecto, al trabajar los requisitos que mayor valor aportan en una iteración.
- Minimiza el número de errores que se producen en el desarrollo, lo que aumenta la calidad del software.

Aspectos técnicos del desarrollo

El prototipo trabajado durante esta tesis se desarrolló con el uso de lenguaje de programación Python 3.7, con el uso de herramientas como YoloV3/V4/V5, Pytorch, Pytesseract, DarkNet, Tensorflow, Google Colab, OpenCv, Flask y RoboFlow. El prototipo tiene como objetivo recibir una imagen como entrada, encontrar la placa vehicular dentro con el uso de redes neuronales, mediante procesos de visión por computadora leer la placa y devolver el resultado descrito en la **Figura 9.**

Debido a que entrenar una red neuronal desde cero puede llegar a ser un proceso muy lento y costoso a nivel de cálculo computacional, se hará uso del fenómeno de transferencia de aprendizaje, el cual permite trabajar con una red entrenada y ajustar sus parámetros para el caso de detección deseado.

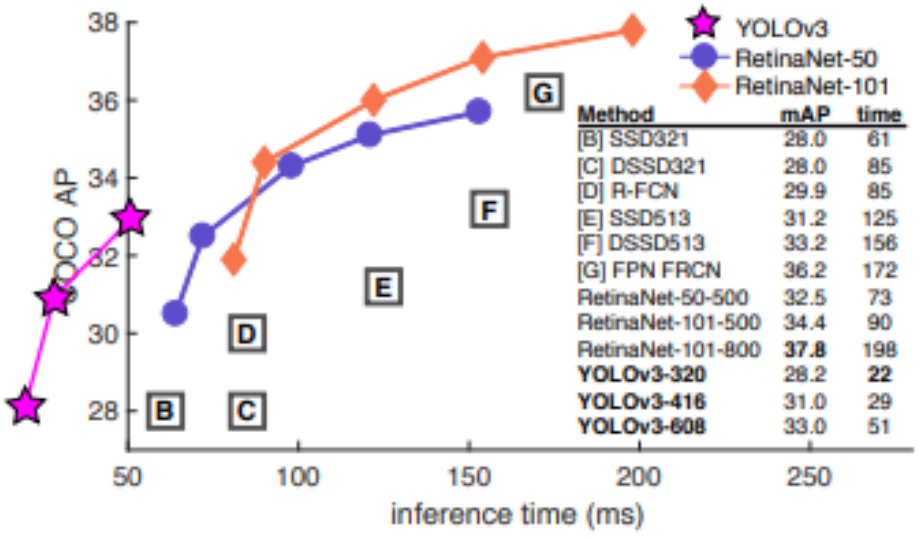
Implementación de metodología

Primera Iteración

En la primera iteración, se optó por YOLO V3, debido a su performance, tiempo de respuesta y resultados respecto a redes neuronales como RetinaNet o Fast R-CNN. En la **Figura 15** se puede apreciar que el tiempo de respuesta de esta red neuronal, lo cual es un determinante critico debido a que este prototipo debe manejar tiempos de respuesta rápidos al ofrecerse como un servicio.

Figura 15.

Grafica de análisis velocidad-precisión sobre el COCO Data set.



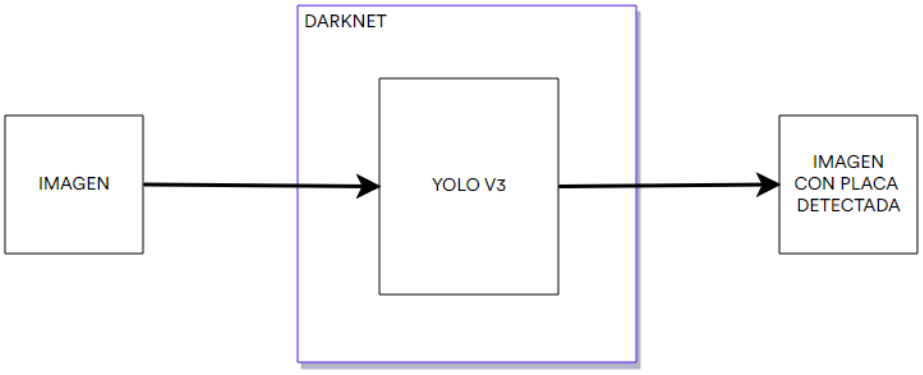
Nota: Recuperado de YOLOv3: An incremental improvement. Copyrigth 2018 por Redmon & Farhadi.

La forma en que se utilizó la red neuronal se describe en la **Figura 16**, esta se encarga de generar las cajas contenedoras de la detección de la placa vehicular, se realizó su implementación sobre Windows con ayuda del repositorio configurado por AlexeyAB.

Este despliegue requirió de la instalación de librerías Cuda y OpenCV, un gestor de paquetes como CONDA y Darknet como framework de entrenamiento para redes neuronales, el cual se compiló localmente con el uso de OpenCV, a fin de optimizar procesos de trabajo de imagen y detección de objetos.

Figura 16.

Arquitectura propuesta para la primera iteración



YoloV3 se ejecuta sobre DarkNet, el cual está escrito en C y utiliza CUDA permitiendo el uso de cómputo de GPU y CPU. Para el desarrollo de este prototipo se decidió desplegar el aplicativo sobre el entorno Windows, el detector de placas vehiculares requirió el entrenamiento previo de la red neuronal, para este fin se utilizó una base de 400 imágenes provistas por Robert Lucian en su repositorio.

Preparación de Entorno

Para la preparación del entorno, se requirió etiquetar las placas dentro de las imágenes. Se utilizó el formato solicitado para DarkNet (XML), el cual incluye puntos X1, Y1, X2, Y2; los cuales definieron el corte de la placa dentro de la imagen, el nombre utilizado en la etiqueta fue: “license-plate”; respecto al resto de atributos que se presentan en las figuras 17 y 18, definen la conexión con el archivo JPG o PNG al que se asocian.

Figura 17.

Nombramiento de archivos para etiquetado

Figura 18.

Estructura de etiquetado de imágenes

El proceso comenzó por clonar el repositorio del programador AlexeyAB, el cual provee una interfaz de YOLO lista para su despliegue en Windows, se requirió la instalación de programas como CMAKE, CUDA, cuDNN y OpenCV. Con el entorno listo se realizó la configuración del archivo MakeFile para habilitar el uso de GPU, OPENCV y CUDNN y proceder a compilar la DarkNet.

Se hizo uso de pesos pre-entrenados sobre el grupo de datos (Data-Set) COCO, el cual maneja 80 clases distintas de detección, debido a esto los resultados fueron refinados y disminuyendo el tiempo del nuevo entrenamiento a realizar. Como último paso de configuración se realizaron cambios sobre el archivo **cfg/yolov3.cfg.**

Por recomendación del dueño del repositorio se trabajó con un lote (Batch) de 64 y subdivisiones de 16, a fin de evitar problemas de memoria, debido a que cada entrenamiento depende de la cantidad de clases de detección con las que se vaya a trabajar, los filtros fueron configurados a partir de la siguiente fórmula:

$$filtros = (\text{No. clases} + 5) \times 3$$

Y la cantidad máxima de lotes se definió a partir de:

$$max - batches = \text{No. clases} \times 2000$$

Este valor no debe ser inferior a la cantidad total de imágenes para entrenamiento, por lo que se entiende que, por cada época, se tomara un lote de 64 imágenes del grupo de datos, estas entran a la DarkNet en grupos de 16, tomando un total de 4 sub-lotes, y se repite hasta completar el total máximo de lotes configurado. Esto sumado al grupo de entrenamiento previamente configurado, permitieron que la red neuronal maneje un óptimo entorno de aprendizaje.

Entrenamiento y Resultados

Para proceder a realizar en entrenamiento, se ejecutó el siguiente comando:

```
!./darknet detector train data/obj.data cfg/yolov3_custom.cfg darknet53.conv.74 -dont_show
```

El entrenamiento tomó alrededor de 8 horas, por lo que es necesario esperar que la red neuronal cargue la información y estudie las imágenes del grupo de entrenamiento, como se aprecia en la siguiente **figura 19.**

Figura 19.

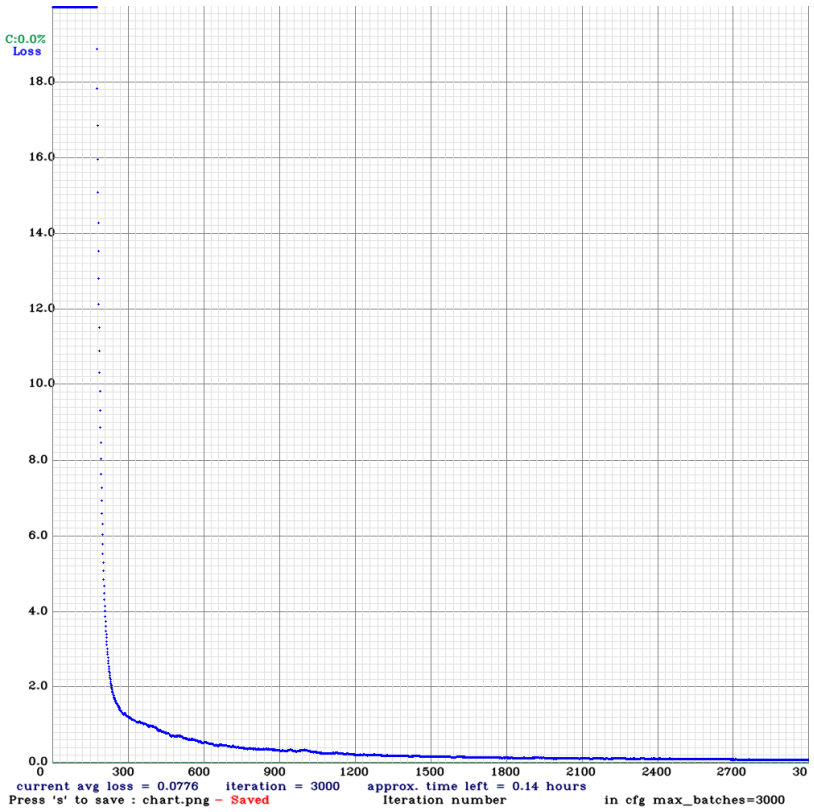
Resultados impresos en consola del entrenamiento de YOLOv3

```
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.538724), count: 1, class_loss = 0.080689, iou_loss = 0.120164, total_loss = 0.200853
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.857820), count: 3, class_loss = 0.008570, iou_loss = 0.047427, total_loss = 0.055997
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.000007, iou_loss = 0.000000, total_loss = 0.000007
 total_bbox = 183329, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.000000), count: 1, class_loss = 0.000010, iou_loss = 0.000000, total_loss = 0.000010
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.790012), count: 3, class_loss = 0.002842, iou_loss = 0.080817, total_loss = 0.083659
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.845991), count: 1, class_loss = 0.021556, iou_loss = 0.009038, total_loss = 0.030594
 total_bbox = 183333, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.000000), count: 1, class_loss = 0.000003, iou_loss = 0.000000, total_loss = 0.000003
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.846185), count: 3, class_loss = 0.000090, iou_loss = 0.036617, total_loss = 0.036706
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.875427), count: 1, class_loss = 0.000033, iou_loss = 0.008078, total_loss = 0.008111
 total_bbox = 183337, rewritten_bbox = 0.000000 %
```

Una vez culminado el proceso, se analizó la gráfica de aprendizaje entregada por los scripts de AlexeyAB, la cual indica la perdida promedio del modelo frente a las iteraciones, expuesta en la **figura 20**

Figura 20.

Grafica de comparación perdida promedio vs número de iteraciones DarkNet



Se pudo apreciar que, pasadas las 250 iteraciones, el modelo apunta a una perdida menor a 2.0, lo cual indica que su precisión mejoró con respecto a su entrada inicial. Sin embargo, al momento de realizar la prueba, por cada imagen de entrada es necesario inicializar la darknet para proceder al análisis, restándole agilidad al proceso en cuestión. Se obtuvo durante este trabajo un margen de 0.85 a 0.99 de confianza en cada detección, como se presenta en la **Tabla 1**.

A partir de las pruebas realizadas sobre este prototipo de detección, se definieron los siguientes resultados:

- Al tener que cargar la Darknet sobre cada solicitud, el tiempo de detección de la placa aumenta.
- YOLO demuestra ser una eficiente solución en la búsqueda de un sistema de detección de placas.
- El Grupo de entrenamiento utilizado, permite detectar placas con ciertos márgenes de error, aun así, se requiere aumentar el mismo, para mejorar estos niveles de confianza.

Conclusiones y Limitaciones

Después de un análisis sobre una serie de imágenes procesadas en el detector, se puede definir las siguientes conclusiones y limitaciones para futuras iteraciones:

- El tiempo de entrenamiento demanda varios recursos locales (GPU - CPU).
- La cantidad, calidad y etiquetado de imágenes, es fundamental para el óptimo entrenamiento de la red neuronal.
- YOLO requiere distintas configuraciones para la cantidad de objetos que se requieran detectar.

- Con el grupo de datos de entrenamiento utilizado, la red neuronal deja de aprender pasadas las 1800 iteraciones.
- El framework Darknet requiere inicializar el modelo por cada imagen que procesa, ralentizando el tiempo de respuesta.
- El despliegue en un entorno Windows conlleva configuraciones adicionales sobre el proyecto original del creador de YOLO.
- El grupo de entrenamiento debe aumentar con el fin de abarcar más escenarios.

Solución

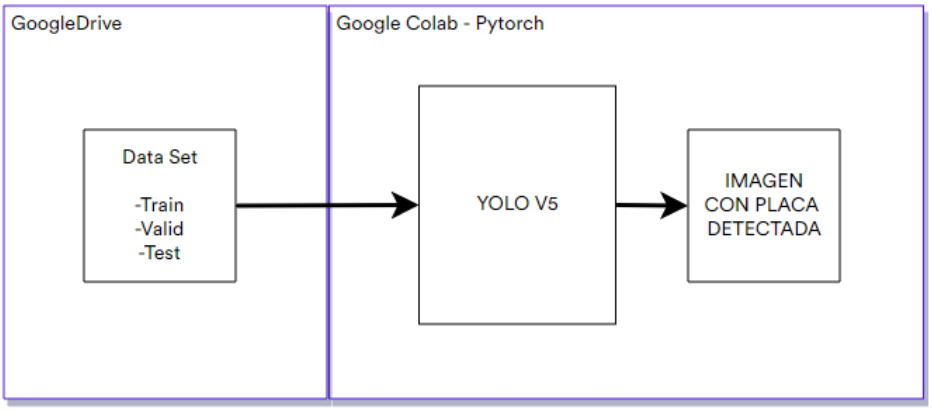
Para reducir el margen de error de la red neuronal en la detección de placas, se propone hacer uso de técnicas para “aumento” de datos pues mejoran las condiciones con las cuales la red neuronal es capaz de aprender a identificar placas, tomando en cuenta factores como distorsión, inclinación, luz y enfoque, a su vez se optará por el uso de YOLOV5 sobre Pytorch, con la finalidad de obtener mejores tiempos de respuesta en el procesamiento de imágenes.

Segunda Iteración

Para la segunda iteración se realizó una investigación sobre el uso de “supercomputadores” de Google Colab, ya que son una solución a las limitadas capacidades de cómputo local, al hacer uso de TPU (unidad de procesamiento de tensores), permite la ejecución de operaciones matemáticas matriciales complejas, permitiendo al GPU y CPU enfocarse en otros procesos y reducir el tiempo de aprendizaje o entrenamiento de una red neuronal. Con esta propuesta se generó la nueva arquitectura para esta iteración, que se presenta en la **Figura 21**.

Figura 21.

Arquitectura propuesta para la segunda iteración

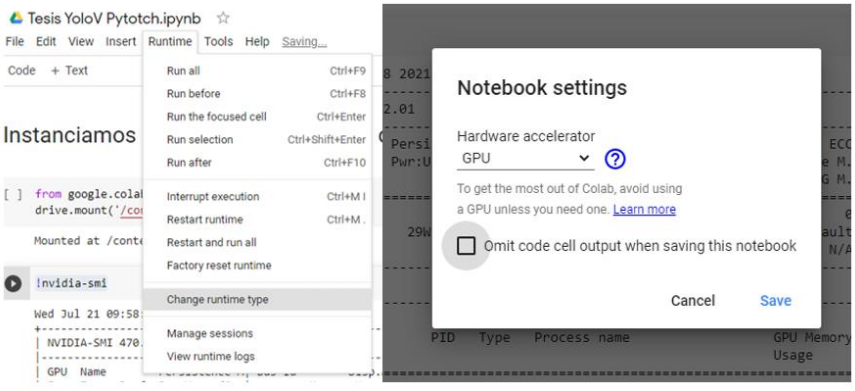


Preparación de Entorno

Dentro de esta iteración se realizó la preparación del entorno de la maquina Jupyter provista por Google Colab para el trabajo de entrenamiento de la red neuronal, es decir, la implementación de las librerías Pytorch. Para esto se realizó una modificación sobre el repositorio provisto por Ultralitycs con YOLOV5 configurado para su uso en los dispositivos de Google Colab. Se cambió el entorno de trabajo, a uno entorno con GPU habilitado como se presenta en la **figura 22**.

Figura 22.

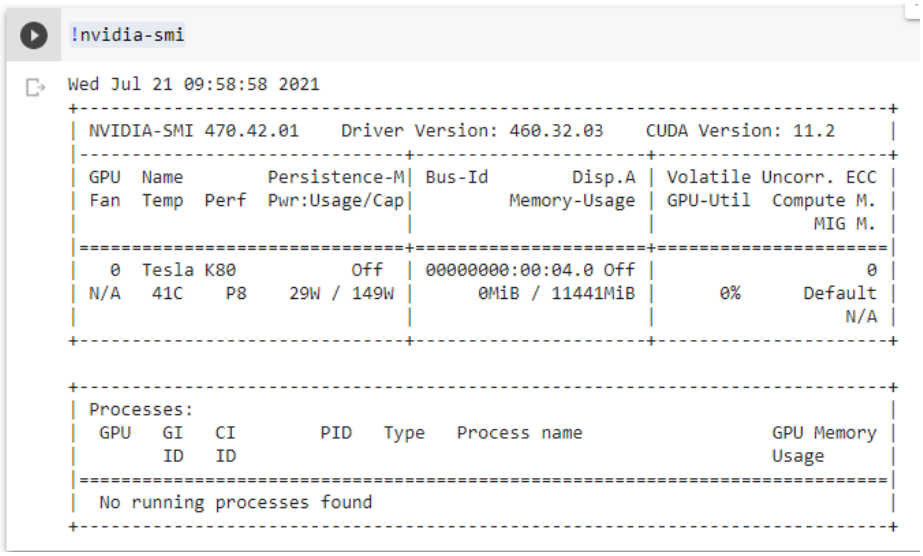
Configuración maquina Jupyter en Google Colab



A continuación, se verificó las características del dispositivo GPU mediante el comando expuesto en la **Figura 23.** Se pudo identificar las diferencias de entorno sobre el cual se entrenó cada red neuronal, cambiando de una GPU RTX 2080 (Entorno local) a una GPU Tesla K80.

Figura 23.

Comando y resultado del análisis de dispositivo GPU

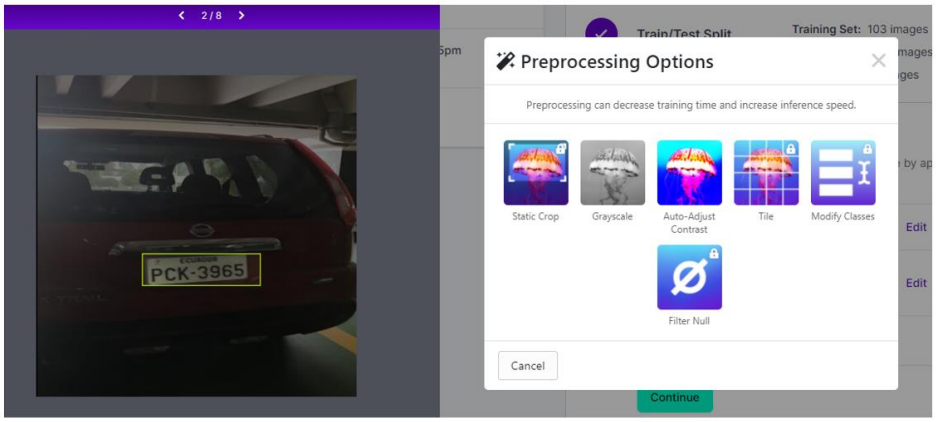


Nota: Recuperado de GoogleColab. Características de unidad de procesamiento Nvidia. Copyrigth 2021.

Para este entrenamiento se manejó el grupo de datos de la anterior iteración, para verificar el cambio de los resultados obtenidos entre un entorno y otro; se optó hacer uso de técnicas de “aumento”, con la herramienta Cloud RoboFlow. Dentro de esta herramienta se realizó el etiquetado de todo el set de datos, y cumplido este objetivo, se procedió a incrementar el grupo de entrenamiento mediante técnicas de procesado de imagen, como se indica en la **Figura 24.**

Figura 24.

Uso de herramienta RoboFlow para técnicas de etiquetado y "aumento"



Dentro de esta herramienta se colocaron las siguientes opciones de "aumento" y preprocesado:

- Preprocesado:
  - Orientación automática (Horizontal)
  - Reajuste de tamaño a 416x416
- Aumento:
  - Cortes en +/- 15 grados Horizontales y verticales
  - Saturación de +/- 25%
  - Brillo de +/- 25%

La herramienta permitió el aumento del grupo de información a un tamaño de 1.2k imágenes, debido a que, por cada proceso de aumento, se provee una salida de dos imágenes editadas con estas mejoras. En este grupo de entrenamiento se separó las imágenes en tres grupos, el grupo de entrenamiento con un 88% del total e imágenes, el grupo de imágenes validas con un 8% y el grupo de prueba con un 4% del total de las imágenes.

Entrenamiento y Resultados

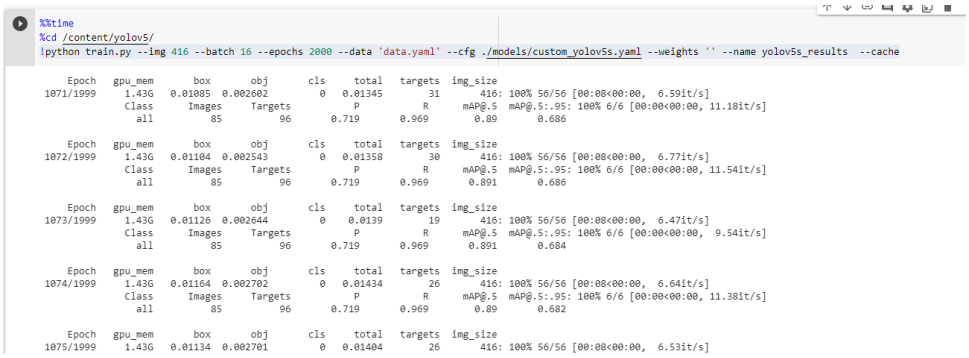
Para realizar el entrenamiento de esta red neuronal se ejecutó el siguiente comando:

```
!python train.py --img 416 --batch 16 --epochs 2000 --data 'data.yaml' --
cfg ./models/custom_yolov5s.yaml --weights '' --name yolov5s_results  --cache
```

En la **Figura 25** se presenta el proceso de aprendizaje, con un total de 2000 épocas, y lotes de 16.

Figura 25.

Resultados impresos en consola del proceso de aprendizaje utilizando Pytorch.



Terminado el proceso de entrenamiento, gracias al uso de TPU y de las herramientas que proveen GoogleColab y tensorboard se obtuvo el desempeño del modelo, que se exhibe en la **Figura 26**.

El análisis indico que al manejar una perdida promedio de menos 0.04 antes de las 100 primeras iteraciones, se espera obtener un mejor margen de detección. Sin embargo, dentro del set de pruebas, el margen de detección oscila entre 0.45 a 0.98, siendo este muy amplio y con escasos problemas para detectar ciertas placas vehiculares como se observa en la **Figura 27.**

Figura 26.

Grafica de comparación perdida promedio vs número de iteraciones

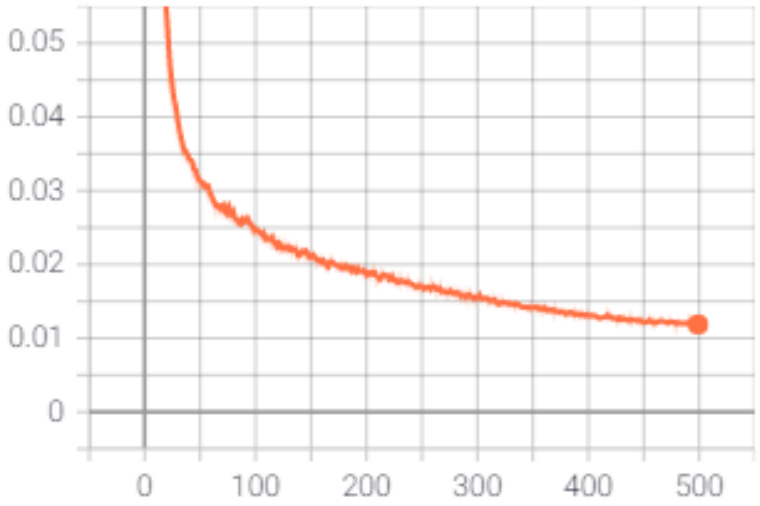


Figura 27.

Detecciones de placas vehiculares en set de pruebas de YoloV5

Con al uso de Pytorch se obtuvieron detecciones más rápidas que el entrenamiento en DarkNet, como se respalda en la **Tabla 2**. En la **Figura 28** se presenta el trabajo sobre un grupo de pruebas de 43 imágenes, donde se realizó detecciones sobre 41 de ellas en un total de 0.839 segundos. Se concluye que es una mejora sustancial en tiempos de respuesta, mas no en márgenes de detección.

Figura 28.

Resultados de análisis con un grupo de 43 imágenes.

```
Fusing layers...
Model Summary: 232 layers, 7246518 parameters, 0 gradients, 16.8 GFLOPS
image 1/43 /content/test/images/dayride_type1_001-mp4-t-1062_jpg.rf.e726b2d9de0e54c181d188d5522a1c1f.jpg: 416x416 1 placas, Done. (0.011s)
image 2/43 /content/test/images/dayride_type1_001-mp4-t-1063_jpg.rf.4e229cd1f7d4555000e66e9819282245.jpg: 416x416 1 placas, Done. (0.011s)
image 3/43 /content/test/images/dayride_type1_001-mp4-t-1075_jpg.rf.88f3eda9f7692f345d3381855492c9ce.jpg: 416x416 1 placas, Done. (0.012s)
image 4/43 /content/test/images/dayride_type1_001-mp4-t-1159_jpg.rf.4f2769daad11a07dd080bebc464f4ae1.jpg: 416x416 2 placas, Done. (0.010s)
image 5/43 /content/test/images/dayride_type1_001-mp4-t-119_jpg.rf.5a1a551661186a2dc0f9a99049b60021.jpg: 416x416 1 placas, Done. (0.010s)
image 6/43 /content/test/images/dayride_type1_001-mp4-t-1259_jpg.rf.369bcbad2c9c7e214bbb1814510ad87b.jpg: 416x416 1 placas, Done. (0.010s)
image 7/43 /content/test/images/dayride_type1_001-mp4-t-1297_jpg.rf.4556bce192ff1adb0b49bc3eeb34916d.jpg: 416x416 1 placas, Done. (0.010s)
image 8/43 /content/test/images/dayride_type1_001-mp4-t-1299_jpg.rf.83fae54e75f20f4767ca39c826d968e5.jpg: 416x416 1 placas, Done. (0.010s)
image 9/43 /content/test/images/dayride_type1_001-mp4-t-1356_jpg.rf.c024377f8e45519df9923a987c545443.jpg: 416x416 1 placas, Done. (0.010s)
image 10/43 /content/test/images/dayride_type1_001-mp4-t-1466_jpg.rf.d42be31c78b66dbb78b2302809e3fca8.jpg: 416x416 1 placas, Done. (0.010s)
image 11/43 /content/test/images/dayride_type1_001-mp4-t-148_jpg.rf.724c13505f9004ddc158a642281ae1bf.jpg: 416x416 1 placas, Done. (0.010s)
image 12/43 /content/test/images/dayride_type1_001-mp4-t-1552_jpg.rf.810d852acbfcbb9e8f2bb408c30a5d21.jpg: 416x416 1 placas, Done. (0.010s)
image 13/43 /content/test/images/dayride_type1_001-mp4-t-1553_jpg.rf.f9355fb948528038962ca3c840752403.jpg: 416x416 2 placas, Done. (0.010s)
image 14/43 /content/test/images/dayride_type1_001-mp4-t-245_jpg.rf.5e29b30aa6bdaca90827b3d17aeabfda.jpg: 416x416 1 placas, Done. (0.010s)
image 15/43 /content/test/images/dayride_type1_001-mp4-t-279_jpg.rf.bed85688c262f2b2f6dadceb48f97495.jpg: 416x416 2 placas, Done. (0.013s)
image 16/43 /content/test/images/dayride_type1_001-mp4-t-301_jpg.rf.df1c5ae09f089b95cd19670cd490d72c.jpg: 416x416 2 placas, Done. (0.010s)
image 17/43 /content/test/images/dayride_type1_001-mp4-t-31_jpg.rf.b69e4f505b6a9d61e31536f8a982edbd.jpg: 416x416 Done. (0.009s)
image 18/43 /content/test/images/dayride_type1_001-mp4-t-3_jpg.rf.38fb646bbcefe5b90b1abd6eca72b749.jpg: 416x416 1 placas, Done. (0.010s)
image 19/43 /content/test/images/dayride_type1_001-mp4-t-430_jpg.rf.0851e2af3fe75fe24b6032d3acbe2165.jpg: 416x416 1 placas, Done. (0.010s)
image 20/43 /content/test/images/dayride_type1_001-mp4-t-532_jpg.rf.64e534eac6f827481393731d25664fb7.jpg: 416x416 1 placas, Done. (0.012s)
image 21/43 /content/test/images/dayride_type1_001-mp4-t-557_jpg.rf.093b6165bad1c04e3ae8066017451026.jpg: 416x416 2 placas, Done. (0.010s)
image 22/43 /content/test/images/dayride_type1_001-mp4-t-564_jpg.rf.553ae9608e4d25306d360280cbeb8729.jpg: 416x416 2 placas, Done. (0.011s)
image 23/43 /content/test/images/dayride_type1_001-mp4-t-635_jpg.rf.3acb1e888311e90f59c9dd29f8e4aaf1.jpg: 416x416 1 placas, Done. (0.010s)
image 24/43 /content/test/images/dayride_type1_001-mp4-t-735_jpg.rf.91b073157606d825472f81d25c1128b1.jpg: 416x416 1 placas, Done. (0.010s)
image 25/43 /content/test/images/dayride_type1_001-mp4-t-758_jpg.rf.fc0d3fb4d68dc14990581fedd7f932a9.jpg: 416x416 1 placas, Done. (0.010s)
image 26/43 /content/test/images/dayride_type1_001-mp4-t-815_jpg.rf.1bd727d17d80eef80d03a9d8f8373a51.jpg: 416x416 1 placas, Done. (0.010s)
image 27/43 /content/test/images/dayride_type1_001-mp4-t-822_jpg.rf.fc9f1a30d91ecb6ec20fb04b2dbe879c.jpg: 416x416 1 placas, Done. (0.010s)
image 28/43 /content/test/images/dayride_type1_001-mp4-t-895_jpg.rf.aa204d15e527332a12796ed50350ea78.jpg: 416x416 1 placas, Done. (0.010s)
image 29/43 /content/test/images/dayride_type1_002-mp4-t-531_jpg.rf.9a5402fb7c37f038497a623cfe6f942e.jpg: 416x416 2 placas, Done. (0.010s)
image 30/43 /content/test/images/dayride_type1_002-mp4-t-647_jpg.rf.87be8a978b99807cfbbd55fb9e3b571c.jpg: 416x416 1 placas, Done. (0.016s)
image 31/43 /content/test/images/dayride_type1_002-mp4-t-654_jpg.rf.1009e9efcc7d42315fc473ccacecaac3.jpg: 416x416 2 placas, Done. (0.010s)
image 32/43 /content/test/images/dayride_type1_003-mp4-t-121_jpg.rf.7cbb8dea111bceabe7d5f39e12a52443.jpg: 416x416 2 placas, Done. (0.013s)
image 33/43 /content/test/images/dayride_type1_003-mp4-t-184_jpg.rf.fba0035e0b2268e33abe76056d8ac4fb.jpg: 416x416 1 placas, Done. (0.010s)
image 34/43 /content/test/images/dayride_type1_003-mp4-t-80_jpg.rf.47efa46cde041cb3f4274436d15a2c90.jpg: 416x416 1 placas, Done. (0.011s)
image 35/43 /content/test/images/nightride_type3_001-mp4-t-110_jpg.rf.8152c46a5a8c84baf271c2eb6592b18b.jpg: 416x416 2 placas, Done. (0.011s)
image 36/43 /content/test/images/nightride_type3_001-mp4-t-11_jpg.rf.f54c0593eacedce74d67a50ea4e2ea0f.jpg: 416x416 1 placas, Done. (0.010s)
image 37/43 /content/test/images/nightride_type3_001-mp4-t-161_jpg.rf.995cab11ebb856fa1972030599debf28.jpg: 416x416 2 placas, Done. (0.011s)
image 38/43 /content/test/images/nightride_type3_001-mp4-t-1_jpg.rf.ff470581110fa9da887373efd1f397ce.jpg: 416x416 1 placas, Done. (0.010s)
image 39/43 /content/test/images/nightride_type3_001-mp4-t-21_jpg.rf.21c113467bc262fda8bbd90dd41d0f7c.jpg: 416x416 1 placas, Done. (0.011s)
image 40/43 /content/test/images/nightride_type3_001-mp4-t-259_jpg.rf.5a7ae0468910ef8530133c53b04c839e.jpg: 416x416 1 placas, Done. (0.010s)
image 41/43 /content/test/images/nightride_type3_001-mp4-t-350_jpg.rf.9c578ef5e2b98962023acbe51fb9baf7.jpg: 416x416 2 placas, Done. (0.010s)
image 42/43 /content/test/images/nightride_type3_001-mp4-t-37_jpg.rf.3405795d37ff833e7cc87505bf638b3b.jpg: 416x416 1 placas, Done. (0.010s)
image 43/43 /content/test/images/nightride_type3_001-mp4-t-74_jpg.rf.033fd126ec04b5b1c70bee1a8d949c9c.jpg: 416x416 1 placas, Done. (0.010s)
Results saved to runs/detect/exp
Done. (0.775s)
```

Conclusiones y Limitaciones

Se puede definir las siguientes conclusiones y limitaciones para futuras iteraciones:

- Google Colab será la herramienta de entrenamiento para la próxima red neuronal, ya que otorga un entorno listo para el despliegue de varios frameworks y permite ahorrar recursos para el proceso de aprendizaje profundo.

- Se requiere de una forma de supervisar Google Colab a fin de no perder la sesión en curso.
- Pytorch ofrece una base sólida para detecciones más rápidas, pero un margen de error elevado.
- Se requiere un nuevo set de imágenes que asemejen un entorno real a fin de obtener mejores resultados de detección.
- Darknet permite un entrenamiento en el que se obtienen mejores resultados.
- La resolución de las imágenes debe ser de al menos 720p.
- En la siguiente iteración el grupo de entrenamiento deberá hacer uso de imágenes con placas ecuatorianas.

Solución

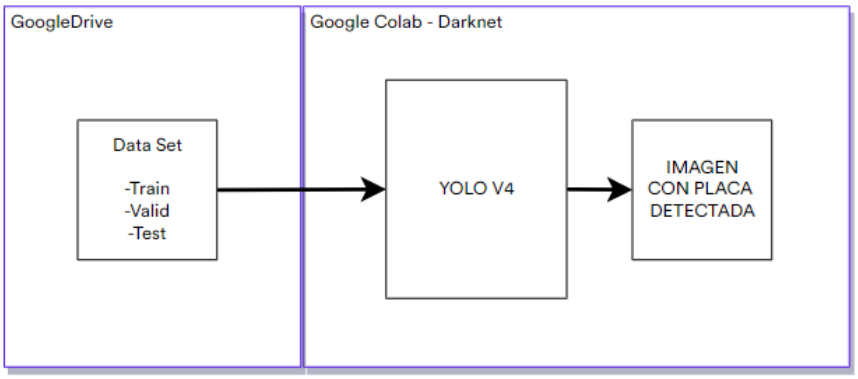
Para reducir el margen de error de la red neuronal en la detección de placas, se propone obtener un nuevo grupo de imágenes de entrenamiento, hacer uso de técnicas de “aumento” de datos, pues mejoran las condiciones con las cuales la red neuronal es capaz de aprender a identificar placas. Se optará por el uso de YOLOV4 sobre Darknet en Cloud (GoogleColab).

Tercera Iteración

En la tercera iteración se optó por realizar el entrenamiento de la red neuronal sobre una maquina Cloud de GoogleColab, esta vez con una base DarkNet, ya que la primera iteración presentó mejores resultados en un margen de detección de la red. Para esto se define la arquitectura de esta iteración en la **Figura 29.**

Figura 29.

Arquitectura propuesta para la tercera iteración



Preparación de Entorno

En esta iteración se realizó el entrenamiento de la red neuronal dentro de GooglColab. Se realizó el proceso de la primera iteración, pero esta vez se trabajó con YOLOV4. En la Figura **30** se observa las GPU provista por GoogleColab**.**

Figura 30.

Comando y resultado del análisis de dispositivo GPU en tercera iteración

```
!nvidia-smi

Thu Jul 22 21:21:29 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.42.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   44C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Nota: Recuperado de GoogleColab. Características de unidad de procesamiento Nvidia. Copyrigth 2021.

Para este proceso, GoogleColab asigno un procesador Tesla T4, distinto al de la iteración anterior. El set de imágenes para esta iteración manejó un total de 1.8k imágenes, que parten de una base de 600 fotografías obtenidas de parqueaderos del Centro comercial El Jardín, Urbanización el Condado y el parqueadero de la Universidad de las Fuerzas Armadas ESPE. Para el etiquetado de información se delimito el área contenida de la placa vehicular y una vez etiquetado el set de datos. Se utilizó una herramienta cloud RoboFlow y con ayuda de sus herramientas de “aumento”, se aplicaron los siguientes filtros:

- Preprocesado:
    - Orientación automática (Horizontal)
    - Reajuste de tamaño a 416x416
- Aumento:
    - Cortes en +/- 15 grados Horizontales y verticales
    - Saturación de +/- 25%
    - Brillo de +/- 25%

Para el proceso de entrenamiento se tomó en cuenta la siguiente configuración:

- Lotes 64 (BATCH)
- Sub-divisiones 16
- Máximo número de Lotes 6000 (MAX-BATCH)
- Pasos entre lotes 4800, 5400

Entrenamiento y resultados

Para realizar el entrenamiento de esta red neuronal se ejecutó el siguiente comando:

```
!./darknet detector train data/obj.data cfg/yolov4-obj.cfg yolov4.conv.137 -dont_show -map
```

El cual hizo uso de los pesos reentrenados en el set de datos de COCO y el framework DarkNet, pues presento mejores resultados ante Pytorch. Culminado el tiempo de entrenamiento que duró alrededor de 10 horas, la consola presentó los resultados mostrados en la **Figura 31**:

Figura 31.

Resultado de entrenamiento YOLOv4

```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.563
avg_outputs = 489778
 Allocate additional workspace_size = 52.43 MB
Loading weights from /content/gdrive/MyDrive/YoloV4ver2/BackUp/yolov4-obj_3000.weights...
 seen 64, trained: 192 K-images (3 Kilo-batches_64)
Done! Loaded 162 layers from weights-file

 calculation mAP (mean average precision)...
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
196
 detections_count = 991, unique_truth_count = 260
class_id = 0, name = placa, ap = 70.39%          (TP = 181, FP = 87)

 for conf_thresh = 0.25, precision = 0.68, recall = 0.70, F1-score = 0.69
 for conf_thresh = 0.25, TP = 181, FP = 87, FN = 79, average IoU = 47.70 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.703883, or 70.39 %
Total Detection Time: 4 Seconds
```

En la **Figura 31** se muestra que el mAP (De las siglas en Ingles, Mean Average Presicion) de la red neuronal es de un 70.39%, lo cual indica el promedio de promedios de precisión de detección. En la gráfica presentada en la **Figura 32,** se observa que las detecciones se estabilizaron pasadas las 3000 iteraciones; permitiendo obtener valores óptimos para detención. Esto se demostró durante las pruebas evaluadas en un total de 10 imágenes de vehículos de la urbe de Quito, en el que se presentaron porcentajes de confianza altos, con niveles de detección entre un 0.78 a un 0.99 de acierto, reduciendo el margen de error obtenido en la iteración anterior como se indica en la detección de la **Figura 33**.

Figura 32.

Comando y resultado del análisis de dispositivo GPU en tercera iteración

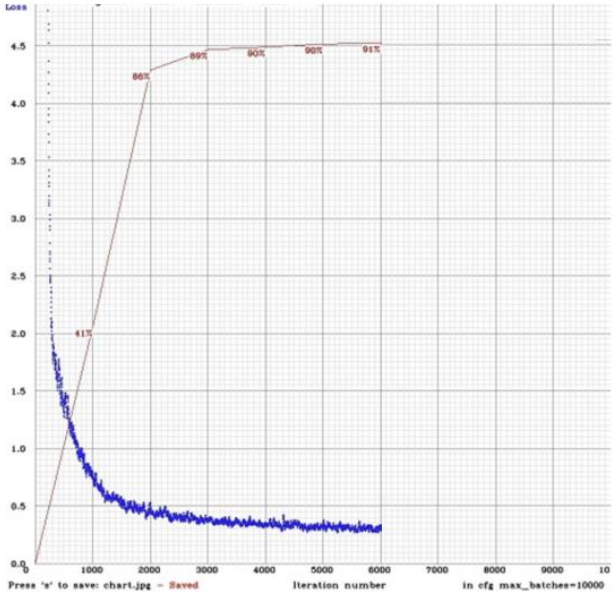
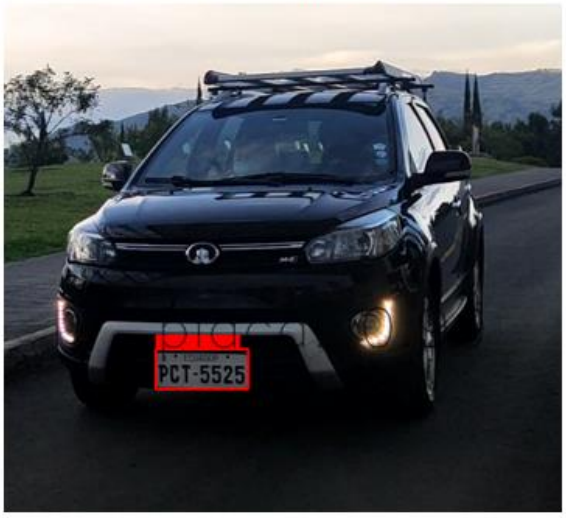


Figura 33.

Resultado de detección de placa vehicular con YOLOv4

```
Loading weights from /content/gdrive/MyDrive/YoloV4/BackUp/yolov4-obj_last.weights...
 seen 64, trained: 384 K-images (6 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
/content/gdrive/MyDrive/YoloV4/Test/foto2.jpg: Predicted in 32.888000 milli-seconds.
placa: 99%
Unable to init server: Could not connect: Connection refused
```



En base a los datos antes mencionados y respaldados por los datos indicados en la **Tabla 3.** Se puede definir que el entrenamiento realizado en Darknet fue acertado que el realizado en Pytorch.

Conclusiones y Limitaciones

A partir de los resultados se pudo definir las siguientes conclusiones y limitaciones para futuras iteraciones:

- Darknet será el framework para el despliegue de la red neuronal en las siguientes iteraciones, debido a sus mejores resultados en sus niveles de confianza para detección de objetos con YOLOV4.
- Se hará uso YOLOV4 en siguientes iteraciones gracias a sus óptimos tiempos de respuesta.
- Se requiere un subsistema capaz de identificar las placas vehiculares.
- Se requiere una solución para despliegue del software

Solución

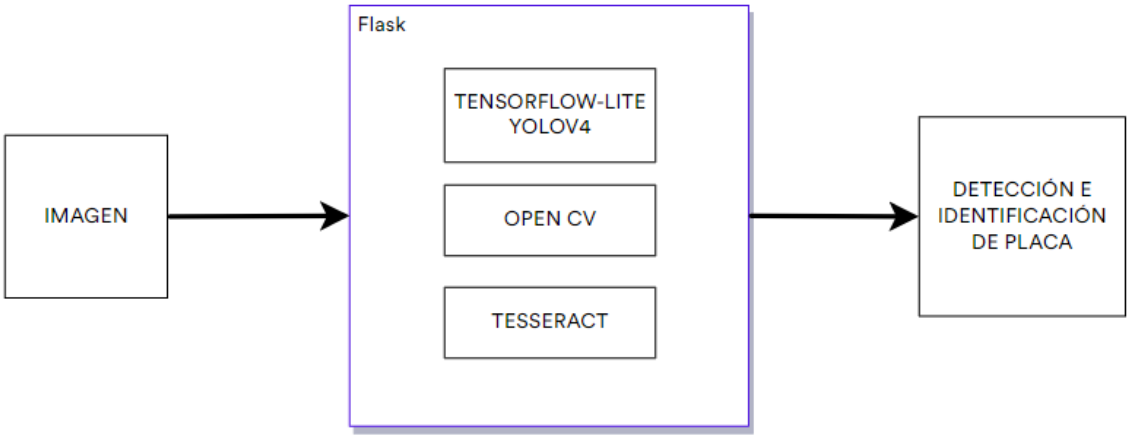
Una vez obtenida una red neuronal entrenada para la detección de placas vehiculares, con un nivel de confianza aceptable, se propone el uso de Flask, un framework para microservicios en Python, TensorFlow-lite y Open CV para desplegar la solución en un entorno Web como servicio.

Cuarta Iteración

En la cuarta iteración se realizó la implementación web del prototipo de software con el uso de Flask. Se planificó una interfaz web para la carga de información, para lo cual se propuso la arquitectura descrita en la **Figura 34:**

Figura 34.

Arquitectura propuesta para la cuarta iteración



Dentro de la gráfica se puede apreciar que Flask se encarga de tomar la imagen o frame de entrada, para procesarlo mediante el módulo de detección formado por TensorFlow Lite y YoloV4, una vez identificados los puntos de la caja contenedora de la placa vehicular, procede a enviar un corte de la imagen al módulo de identificación de la placa vehicular creado mediante OpenCV y Tesseract.

Desarrollo de Aplicativo Flask

Para el desarrollo de una primera versión del API, se optó por el uso de una herramienta para manejo de entornos y paquetes de dependencia conocida como CONDA, siendo las más importantes:

- OpenCV
- TensorFlow-GPU
- Flask
- Pytesseract

En esta iteración se reutilizó los pesos entrenados en la tercera iteración, pasando por un proceso de inferencia por medio de la librería Nvidia TensorRT, la cual permite obtener una mejora entre 4 y 5 veces mayor en tiempos de respuesta sobre modelos de aprendizaje profundo, como es el caso de YoloV4. En este proceso se utilizó una implementación de YoloV4 en Python junto con librerías de Tensorflow del desarrollador Duogviet Hung (Hùng, 2020).

Una vez con el repositorio clonado, se convierte el modelo previamente entrenado en la iteración anterior, a un formato valido para TensorFlowLite, con el uso del programa save_model.py mediante el comando:

```
python save_model.py --weights ./data/yolov4.weights –output /checkpoints/yolov4.tf --input_size 416 --model yolov4
```

Esto permitió a la librería Nvidia TensorRT optimizar nuestro modelo para ser utilizado por Tensorflow, lo que permitió realizar detecciones sobre cualquier input entregado en mejores márgenes de tiempo de detección de objetos. Para evitar cargas en los tiempos de respuesta del prototipo, fue necesario iniciar una única sesión de Tensorflow, para que este inicie el modelo de YoloV4, dicho proceso se aplicó en la sección de código de la **figura 35**.

Figura 35.

Proceso de carga de configuración de modelo YOLOv4

```
def load_config():
    config = ConfigProto()
    config.gpu_options.allow_growth = True
    session = InteractiveSession(config=config)
    STRIDES, ANCHORS, NUM_CLASS, XYSCALE = utils.load_api_config(FLAGS)
    print(" * Cargando Modelo tensorFlow")
```

El prototipo inicial implementó una interfaz simple en la que se solicita al usuario el ingreso de una imagen y se verifica en efecto si es un archivo valido para ejecución de la detección de placa como se indica en el código de la **Figura 36**:

Figura 36.

Verificación de archivo para carga en red neuronal

```
if 'file' not in request.files:
        return render_template('upload.html', msg='No has seleccionado un archivo')
file = request.files['file']
if file.filename == '':
        return render_template('upload.html', msg='No has seleccionado un archivo')
if file and allowed_file(file.filename):

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

La detección se llevó a cabo en una clase utilitaria, en la cual, previo al análisis, es necesario el cambio del orden de los colores de BGR (azul, verde, rojo) a RGB (rojo, verde, azul) y el cambio de tamaño a 416px x 416px, que fue el tamaño de input que se definió durante el entrenamiento de YoloV4 en DarkNet. Se asignó esta entrada a un tensor (matriz de información) y se ejecutó un método de supresión de no máximos, el cual eliminó las cajas superpuestas por unión sobre cajas anteriormente seleccionadas, dando como resultado una caja de predicción con su respectivo nivel de confianza. Si dicho nivel de confianza es mayor a 0.8 y el objeto detectado es una “placa”, se procede al reconocimiento de los caracteres que la componen con el uso de Tesseract, un OCR configurado localmente, este proceso se corre dentro de otra función, como se presenta en la **Figura 37.**

Figura 37.

Validación de confianza para identificación de placa vehicular

```
if class_name not in allowed_classes:
        continue
 else:
        if score > 0.80:
                plate_number = recognize_plate(image, coor)
```

Este subsistema se encarga de recortar la placa de la imagen, utilizando las coordenadas recibidas de YoloV4, y realizar una serie de procesos sobre a imagen a fin de que Tesseract sea capaz de identificar el texto en la misma. Los filtros aplicados sobre la imagen se presentan en la **Figura 38:**

Figura 38.

Filtros aplicados sobre la placa detectada

El proceso de aplicación de filtros se describe a continuación:

- Cambio a escala de grises y agrandado de imagen
- Blur Gaussiano para eliminar cualquier ruido de la imagen
- Binarización por el método de umbral de Otsu para distinguir el fondo de la placa
- Se aplica Dilatación para exaltar los contornos blancos
- Se plantea una solución para un entorno generalizado con varias resoluciones. presentado en la **Figura 39**, si la imagen no cumple el mínimo de 720p de resolución, se procesa el cuadro completo de la placa en Tesseract.

Figura 39.

Filtros aplicados para la segmentación de caracteres

- Si la imagen cumple con el tamaño mínimo de 720p, se realiza el siguiente proceso:
    - Se buscan todos los contornos dentro de la placa.
    - Se filtran los contornos respecto a su tamaño, esos análisis se pueden ver en el anexo de la **Tabla 2.**

- Individualmente se toma cada carácter y se invierten los colores, ya que Tesseract está entrenado sobre fuentes de color negro.
- Se aplica otro Blur para eliminar ruidos de la imagen.
- Se concatena cada carácter leído por el OCR.
- Se limpia el texto leído y se devuelve la placa extraída.

Resultados

La placa extraída se renderizó junto a la fotografía con la detección realizada por YoloV4 y se imprimió la placa, el nivel de confianza de detección y el tiempo que demoró en realizar el proceso de identificación de placa vehicular. Estos datos se encuentran en la **Tabla 4** y la visualización de resultados se presenta en la **Figura 40.**

Figura 40.

Render HTML del prototipo de software

Resultado:

La placa de la imagen cargada es: **Placa: PCV4306 Confianza de YOLOv4tfTyny: 0.97526765 Segundos 2.763056516647339**

El análisis nos demostró que la placa fue extraída con éxito, al igual que un intervalo de confianza de YoloV4 de 0.98 y un tiempo total de proceso (Desde la subida del archivo, hasta la repuesta de la imagen junto a sus detecciones) de 2.7 segundos. Para verificar resultados y veracidad del algoritmo se ejecutó un set de pruebas de 10 imágenes, cuyos resultados se pueden apreciar en la **Tabla 4**, determinando lo siguiente:

- La mayoría de las imágenes con una resolución superior a 720p tuvieron resultados óptimos, con identificación de placas vehiculares adecuadas.
- El método por segmentación de placa genera mejores resultados que el método de lectura por bloque.
- Algunos caracteres de las placas ecuatorianas son mal interpretados por el OCR
- El ruido de la imagen, así como variaciones en el color de la placa, generan resultados erróneos al distorsionar la figura de los caracteres de la placa vehicular.
- Flask es un entorno que permite el rápido despliegue de soluciones para microservicios.
- Se requiere de un end-point capaz de procesar video.

Conclusiones y Limitaciones

Se pudo definir las siguientes conclusiones y limitaciones para futuras iteraciones:

- Los bordes detectados en algunas placas vehiculares chocan con el marco de la placa, por lo que se requiere un mejor ángulo para el análisis de texto.
- El procesado de la imagen implica un tiempo extra sobre la respuesta debido al trabajo de matrices que realiza OpenCV, lo cual se mantendrá presente en las siguientes iteraciones debido al equipo sobre el que se ejecuta el aplicativo.

- Se requiere disminuir el error de reconocimiento presentado por la confusión de caracteres entre números y letras.
- Se requiere validaciones sobre el subsistema de identificación de placas vehiculares.
- Se requiere mejorar el grupo de imágenes de prueba, con mejor iluminación y menor distorsión para mejorar el reconocimiento de caracteres.

Solución

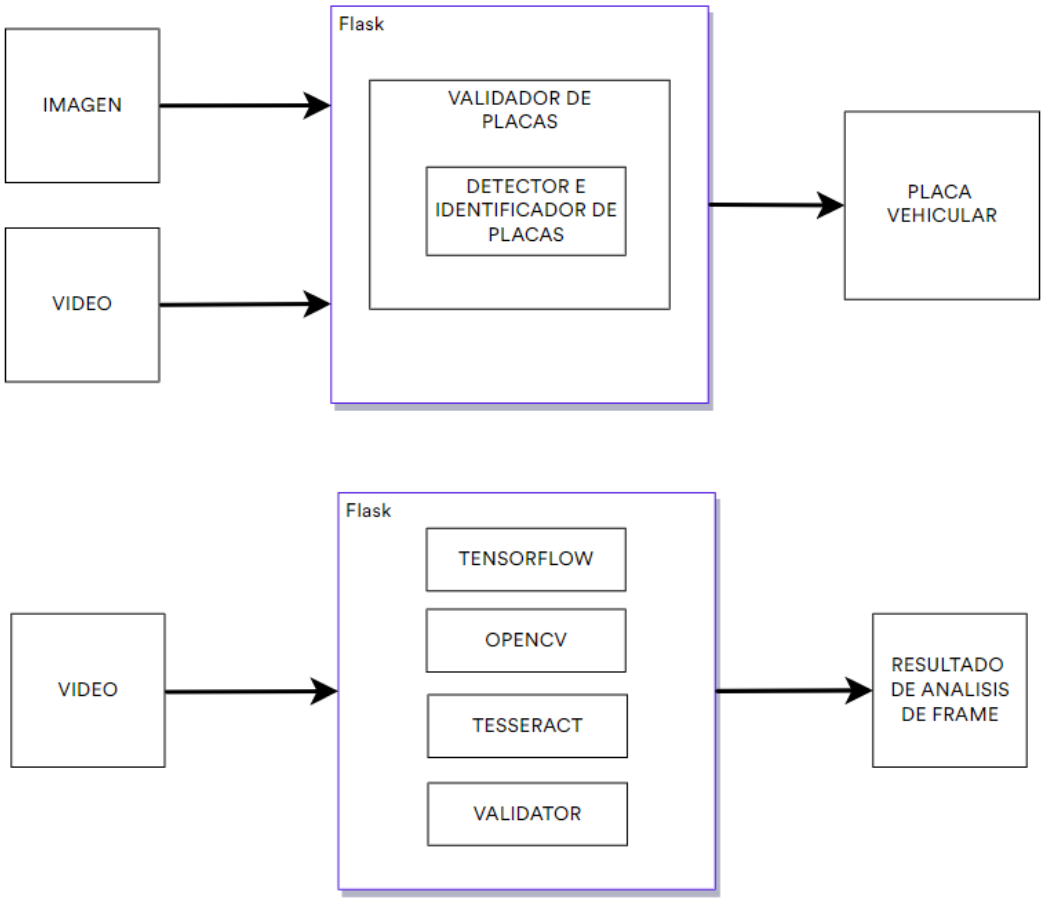
Se propone generar un módulo para la detección de placas con video en tiempo real, tomando los frames de dicha entrada para detectar y reconocer la placa vehicular en el video. Debido a la cantidad de frames y las distintas posiciones del objeto en los mismos, se requiere crear un algoritmo de validación de placas, el cual alojará las detecciones de placas en una lista y tomará la placa con tres coincidencias como la placa vehicular correcta, se validará la lectura de caracteres individuales acorde a su posición de lectura, para solventar los problemas presentes en Tesseract al momento de realizar la lectura de placa.

Quinta Iteración

En esta iteración se incorporó una fuente de video, para lo cual se actualizó el diagrama de arquitectura, con los módulos presentados en la **Figura 41**. En esta arquitectura se encapsula el módulo de detección e identificación de placas vehiculares, ya que el end-point de video procesará cada frame al igual que una imagen, pero realizará las validaciones especificas después de obtener un grupo de resultados con respecto al frame de corte con el que se defina por configuración. El módulo de detección y reconocimiento de imágenes no sufre cambios en esta iteración.

Figura 41.

Arquitectura propuesta para la quinta iteración

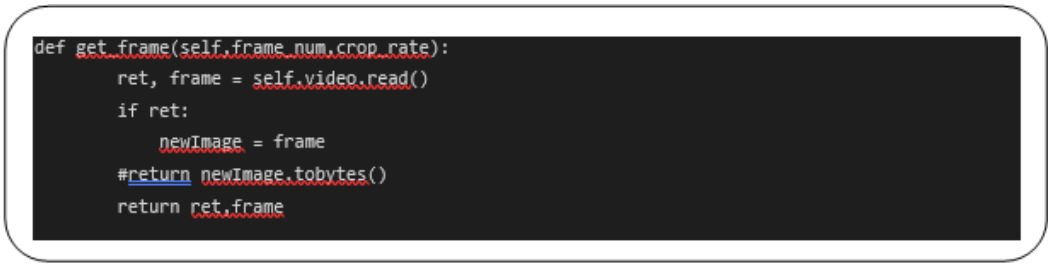


Inclusión De Modulo Para Entrada De Video

Se añadió un nuevo módulo al prototipo de software, el cual se encarga del procesamiento del video, tanto de la detección como reconocimiento de la placa inmersa. Para ello se incorporó una clase “camera.py” la cual solicita una fuente de video, y que mediante el método get_frame(), devuelve una imagen del video en ese instante, capturando frame por frame, como se expone en la **Figura 42**.

Figura 42.

Código para la obtención de frames de una fuente externa de video



```
def get_frame(self,frame_num,crop_rate):
        ret, frame = self.video.read()
        if ret:
            newImage = frame
        #return newImage.tobytes()
        return ret,frame
```

En la iteración anterior, se evidenció que el reconocimiento de caracteres implica un mayor tiempo que el proceso de detección, debido al trabajo de matrices que se aplica sobre la imagen para poder ser reconocida por el OCR; por esta razón fue necesario definir una tasa de frames sobre la cual trabajar, para realizar el análisis de dichos espacios de tiempo. En este caso se definió en una captura de imagen cada 15 frames (crop-rate de análisis), definido en la variable “crop_rate”, como lo indica la F**igura 43**.

Figura 43.

Código para tomar un frame cada 15 frames.

```
if ret:
      if frame_num % crop_rate == 0:
            contador +=1
            print("Numero total de intentos : "+str(contador))
            frame_analisys(frame)
            frame_num += 1
      ret, frame = cv2.imencode('.jpg', frame)
```

Cada frame es tratado como una imagen, y procesado siguiendo los pasos descritos en la cuarta iteración. El análisis de cada frame fue guardado en una lista de String, la cual debido al exceso de “datos basura” o placas no validas, requirió de un algoritmo de depuración de placas vehiculares.

Validador de placas vehiculares

Con la finalidad de no poblar la lista de aciertos con datos basura y evitar coincidencias erróneas, se implementó un validador de placas vehiculares ecuatorianas, con la ayuda de las expresiones regulares descritas en la **Figura 44**. Estas permiten generar patrones los cuales una cadena de caracteres debe seguir para ser tomados en cuenta como datos válidos.

Figura 44.

Expresiones regulares para validación de placas vehiculares

```
LICENCE_PLATE_REG_NEW = r"^(?:[A-C|E|G-N|O-Z][A-Z][A-Z][0-9]{4})$"
LICENCE_PLATE_REG_OLD = r"^(?:[A-C|E|G-N|O-Z][A-Z][A-Z][0-9]{3})$"
LICENCE_PLATE_REG_DIP = r"^(?:[C|O|A|I][C|D|I|T][0-9]{4})$"
```

Como se aprecia en la **Figura 44**, el validador cuenta con los 3 tipos de placa vigentes en Ecuador, placas nuevas (XXX####), placas antiguas (XXX###) y placas diplomáticas (XX####). Esto resta el margen de error de identificación de placas vehiculares al tomar solo las placas vehiculares en un formato valido instruido por la ANT. Cada resultado del proceso de detección e identificación entra en este algoritmo de validación de placa vehicular, para luego ser analizado por el contador máximo de repeticiones, como se indica en la **Figura 45**, una vez cumplido el máximo número de repeticiones, la lista se vacía para dar paso a la siguiente detección.

Figura 45.

Extracción de la placa con mayor número de repeticiones en la lista.

```
num_plate = utils.validate_ecuadorian_licence_plate(num_plate)
    if num_plate!="":
        LICENCE_PLATES.append(num_plate)
        print(LICENCE_PLATES)
        licence_plate = utils.most_frequent(LICENCE_PLATES)
        c = Counter(LICENCE_PLATES)
        collection = c.most_common(1)
        counter = collection[0]
        print("La placa : "+licence_plate+" se repite "+str(counter[1])+" veces")
        if counter[1]==MAX_COMMON_NUMBER:
            print("La placa del vehiculo es: "+licence_plate)
            LICENCE_PLATES.clear()
            LICENCE_PLATES = [""]
            print("Finalizando proceso de analisis")
```

Restricciones al OCR

En el desarrollo de las pruebas de la cuarta iteración, se pudieron evidenciar errores de lectura u omisión de caracteres durante el proceso de segmentación de placa vehicular, esto se debe a que en algunas placas caracteres como la letra Q o el número 1 se confunden fácilmente por el OCR que los traduce por valores numéricos o letras (9 / L). Por tal razón se implementó un valor de indexación que permite conocer en qué posición de la placa nos encontramos y por ende si se requiere el reconocimiento de únicamente letras o números. Dichas validaciones pueden corroborarse en la **Figura 46**, la sentencia tessedit_char_whitelist permite definir un conjunto de caracteres que deberán buscarse en el input de tesseract, permitiéndonos separar los grupos de letras y números en la placa vehicular.

Figura 46.

Iteración entre segmentaciones detectadas por OpenCV

```
if indexRoi < 3:
        text = pytesseract.image_to_string(roi, config='-c
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ\|/ --psm 13 --oem 3')
        if "|" not in text:
        clean_text = re.sub('[\W_]+', '', text)
        clean_text = re.sub('LL', 'L', text)
        else:
        clean_text = "I"
else:
        text = pytesseract.image_to_string(roi, config='-
c tessedit_char_whitelist=0123456789Q/ --psm 13 --oem 3')
        if "/" not in text:
        clean_text = re.sub('[\W_]+', '', text)
        clean_text = re.sub('Q', '9', text)
         else:
        clean_text = "7"
```

Es decir, durante las 3 primeras iteraciones válidas (iteraciones en las que los cortes cumplan con cierto tamaño proporcional a la imagen general), buscará únicamente caracteres del alfabeto entre la A y la Z, mientras que, para el resto de las iteraciones, buscara únicamente números. Dentro de este ajuste se utilizó el valor del segmentado de página de 13 y el motor número 3, pues permiten obtener la lectura precisa de los caracteres individuales.

Pruebas y Resultados

Al ejecutar el prototipo, Flask desplegó los puertos de comunicación necesarios, para este caso de prueba el enrutamiento por default fue seleccionado para realizar el procesamiento de video como se presenta en la **Figura 47**. Los resultados se presentan en la consola mientras el video corre en el template default de Flask.

Figura 47.

Resultados del análisis de video en tiempo real

Para este caso de pruebas se ejecutó el algoritmo con cortes a los 15 y 30 frames sobre 37 videos grabados en la entrada del parqueadero de la Universidad de las Fuerzas Armadas ESPE, los resultados se adjuntan en las **Tablas 5 y 6**. Una vez finalizado el proceso de análisis de video, se realizó cortes de frames al momento de llegada a la barra, para verificar si el módulo de identificación de placas para imágenes tiene mejores resultados. Se trabajó con dos grupos de imágenes, resolución original de video para imágenes, y resolución 720p, cuyos resultados se reflejan en las **Tablas 7 y 8**. De estos resultados podemos indicar que:

- El filtro de validación de placas vehiculares permitió evitar llenar la memoria de datos basura, provenientes de la lectura errónea de placas vehiculares realizada por Tesseract.
- El algoritmo de segmentación tuvo mejores resultados sobre imágenes a 720p.

- El tratamiento de la imagen en video en tiempo real generó un corto tiempo de congelamiento en el frame sobre el que se procesó la lectura de placa debido al uso de OPENCV.
- Al trabajar con varias imágenes de un mismo vehículo en distintos estados de tiempo, se obtuvo distintos resultados debido al ruido y desenfoque como se presenta en la **Figura 48**.
- La ubicación lateral de la cámara generó problemas como la confusión de caracteres debido al grado de inclinación, o mezcla de letras con el marco gracias a problemas de mantenimiento de placas, como se presenta en la **Figura 49**.
- Si el video cargado provee información desde un punto medio del carril, el algoritmo de validación de placas puede confundir resultados, al vaciar la lista antes de identificar las placas de ambos vehículos del carril.

Figura 48.

Contornos obtenidos de un frame distorsionado por movimiento.

Figura 49.

Resultado de error de lectura de placa vehicular inclinada

La placa de la imagen cargada es: **Placa: POH2623 Confianza de YOLOv4tfTyny: 0.936936 Segundos 2.8809061050415O4**

- En todos los casos se puede detectar la placa vehicular, sin embargo, al momento de identificar los caracteres que la componen, se presentan errores sobre la lectura de placa.
- En el acto de realizar la dilatación de las placas, sus bordes se pueden unir, formando un único contorno el cual no puede ser identificado por Tesseract, pues no pasa la validación previa de tamaño establecida para caracteres individuales en el proyecto como se presenta en la **Figura 50**.

Figura 50.

Contornos unidos entre letras o al margen, a causa de un de dilatación

Debido a las razones antes mencionadas, se optó por probar el OCR en imágenes con placas que sean visibles de forma frontal, a fin de determinar la eficiencia del OCR implementado. Estos estudios se realizaron sobre un set de 50 fotografías obtenidas en los parqueaderos de la Universidad de las Fuerza Armadas ESPE, los resultados pueden ser visualizados en la **Tabla 9**, los cuales permiten concluir que:

- Los niveles de confianza de la red neuronal son óptimos, pues se evidenció un promedio de 0.95% dentro de este grupo de pruebas, con un único valor mínimo de 0.81%
- El método de extracción de caracteres por segmentación consiguió mayores aciertos sobre el método de detección por bloque de texto.
- La detección de caracteres por segmentación presentó problemas relacionados a niveles de luz o impurezas provenientes de distintas placas como se aprecia en la **Figura 51**.
- Tesseract puede confundir ciertos números entre sí, como se suscitó con los valores 1, 4 y 7.

Figura 51.

Perdida de información a causa de niveles de luz.

Solución

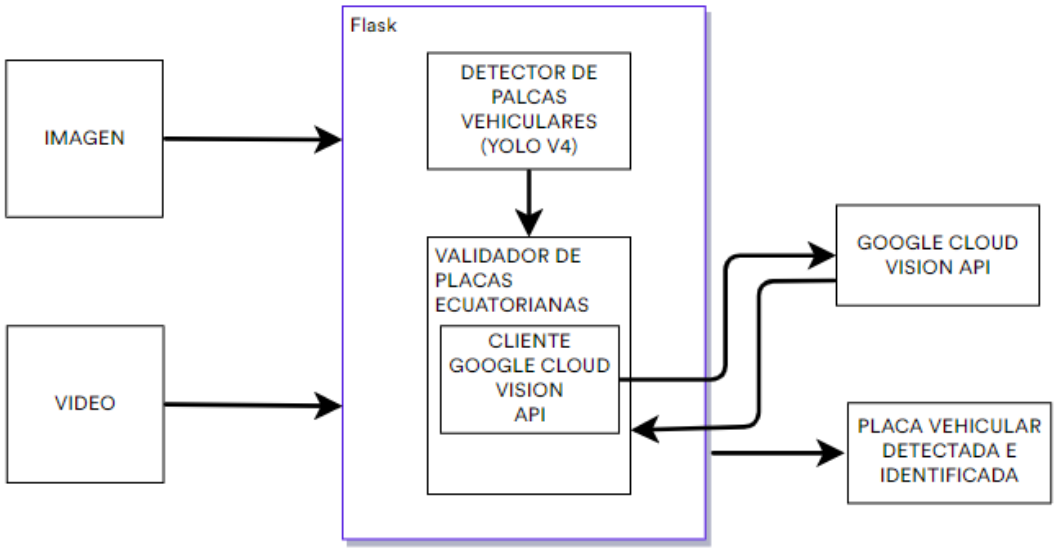
Se propone el cambio de O.C.R. de Tesseract por Google Cloud Vision API, volviendo al prototipo dependiente de una solución Third Party.

Sexta Iteración

En esta iteración se implementó un cliente de servicio para la herramienta Google Cloud Vision, para lo cual se definió la siguiente arquitectura en la **Figura 51.**

Figura 51.

Arquitectura propuesta para la sexta iteración

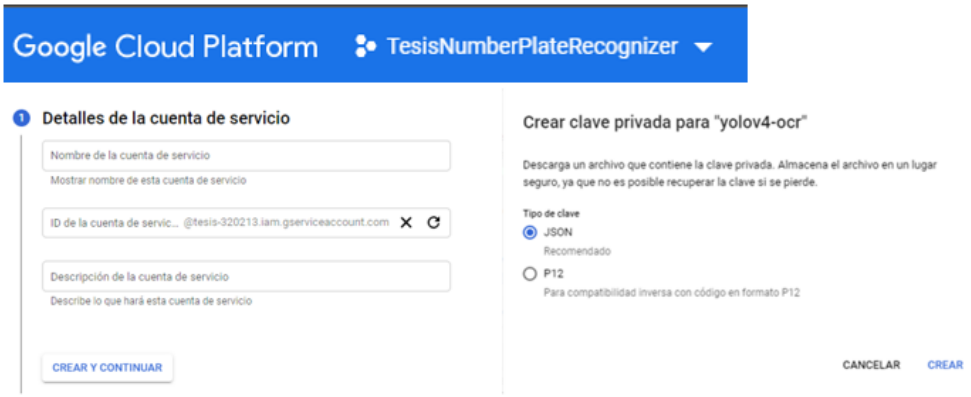


Cliente de servicio Google Cloud API

Para implementar el cliente de servicio de Google Cloud Vision, se requiere un proyecto dentro de Google cloud platform, en del cual, se habilita el API de servicio de Cloud Vision, y configuran las credenciales compatibles para consumo del API, agregando una cuenta de servicio y seleccionando un Key-Type de formato JSON, el cual se coloca dentro de la ruta interna del proyecto como se presenta en la **Figura 52**.

Figura 52.

Interfaces de configuración en Google Cloud Platform



Mediante el controlador de entornos y dependencias Conda, se agregaron las siguientes librerías al entorno de desarrollo:

- Google-cloud-vision
- Easydict

Dentro de la Clase utilitaria se generó un nuevo método que realice la llamada al servicio API de Google mediante las siguientes líneas de código presentes en la **Figura 53**.

Figura 53.

Interfaces de configuración en Google Cloud Platform

```
with io.open('dilation.png', 'rb') as image_file:
        content = image_file.read()
    image = vision.Image(content=content)
response = client.text_detection(image=image)
    texts = response.text_annotations
for text in texts:
        detection =text.description
return detection
```

En la figura anterior se aprecia que el servicio de Google respondió con una lista de detecciones, las cuales se trabajaron posteriormente para obtener la placa. Para aumentar el porcentaje de éxito de lectura, se requirió él envió de una imagen previamente procesada, para lo cual se aplicó un cambio a escala de grises, se incrementó el tamaño de la imagen, se aplicó un filtro Blur para eliminar cualquier impureza, el método de Otsu para diferenciar background de foreground y finalmente una dilatación con un kernel rectangular, descritos en la **Figura 54.**

Figura 54.

Filtros para el procesamiento de imagen previo al envió para API Google Cloud Vision Service

```
box = img[int(ymin)-5:int(ymax)+5, int(xmin)-5:int(xmax)+5]
gray = cv2.cvtColor(box, cv2.COLOR_RGB2GRAY)
gray = cv2.resize(gray, None, fx = 3, fy = 3, interpolation = cv2.INTER_CUBIC)
blur = cv2.GaussianBlur(gray, (5,5), 0)
ret, thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)
rect_kern = cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))
dilation = cv2.dilate(thresh, rect_kern, iterations = 1)
cv2.imwrite('dilation.png', dilation)
```

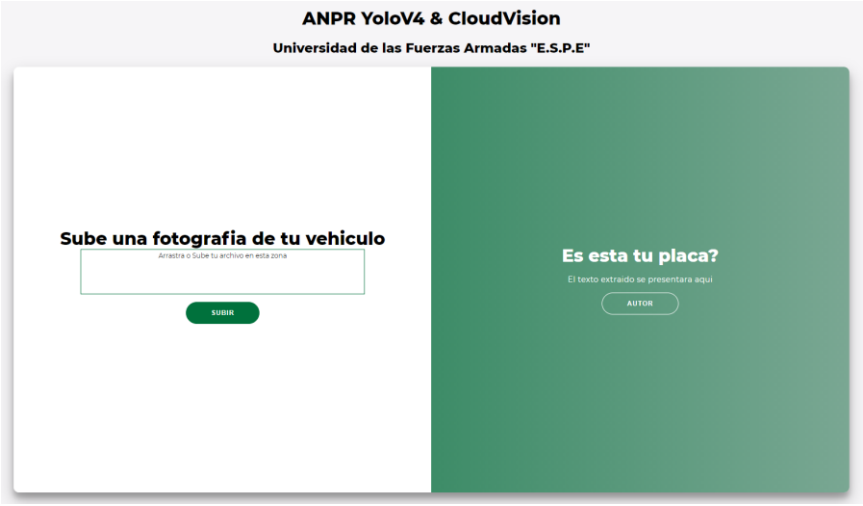
Se omitió el uso de reversión de colores a la imagen, debido a problemas del API para diferenciar ciertos caracteres. La respuesta del servicio es alojada en un arreglo de cadena de caracteres, de la cual se procede a depurar palabras como lo son “Ecuador” o “ANT”.

Interfaz Gráfica

La interfaz fue modificada para ser más amigable al usuario. Se reubicó el panel de presentación de la información y se crearon animaciones con créditos de autor, respetando los colores institucionales como se puede apreciar en la **Figura 55**.

Figura 55.

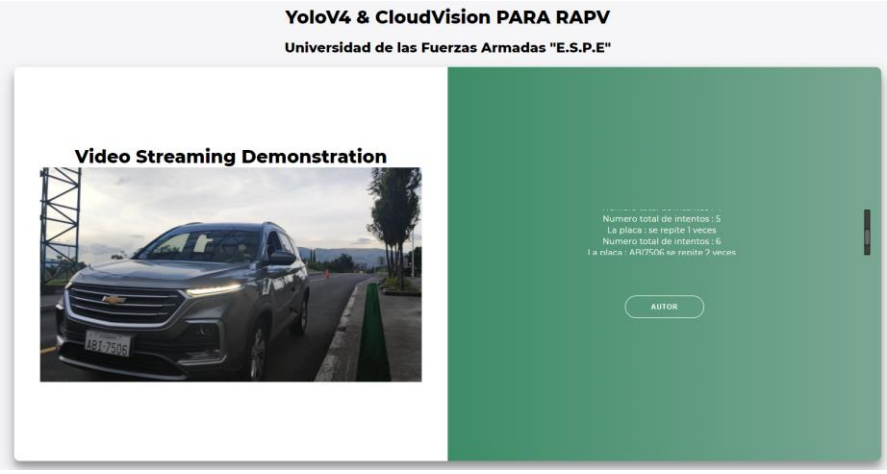
Interfaz para carga de imágenes



El panel izquierdo se encarga de la carga de la información, y el panel derecho presenta el análisis obtenido por la red neuronal. Mientras tanto para el URL de reconocimiento en video se presenta en el siguiente formato mostrado en la **Figura 56.** Los resultados del análisis se presentan en el panel derecho, colocando en negrillas el resultado final de la detección.

Figura 56.

Interfaz para el procesamiento de video en tiempo real



Pruebas Y Resultados

Se utilizó el mismo grupo de pruebas de las anteriores iteraciones, cuyos resultados se pueden verificar en la **Tabla 10**. Con respecto al trabajo sobre video en tiempo real, las detecciones e identificaciones se ejecutan con mayor precisión, los cuales son descritos a mayor detalle en la **Tabla 11**. A partir de estos resultados obtenidos, se pueden indicar:

- Google Cloud Vision es una solución fiable ante la perdida de datos que ocurría al utilizar Tesseract.
- El tiempo de respuesta del servicio Flask no se vio afectado, puesto que la respuesta de los servicios de Google es inmediata y no se procesan grandes cantidades de texto.
- La calidad de las imágenes debe manejar mejores escenarios iluminados a fin de mejorar los resultados de lectura de placas.

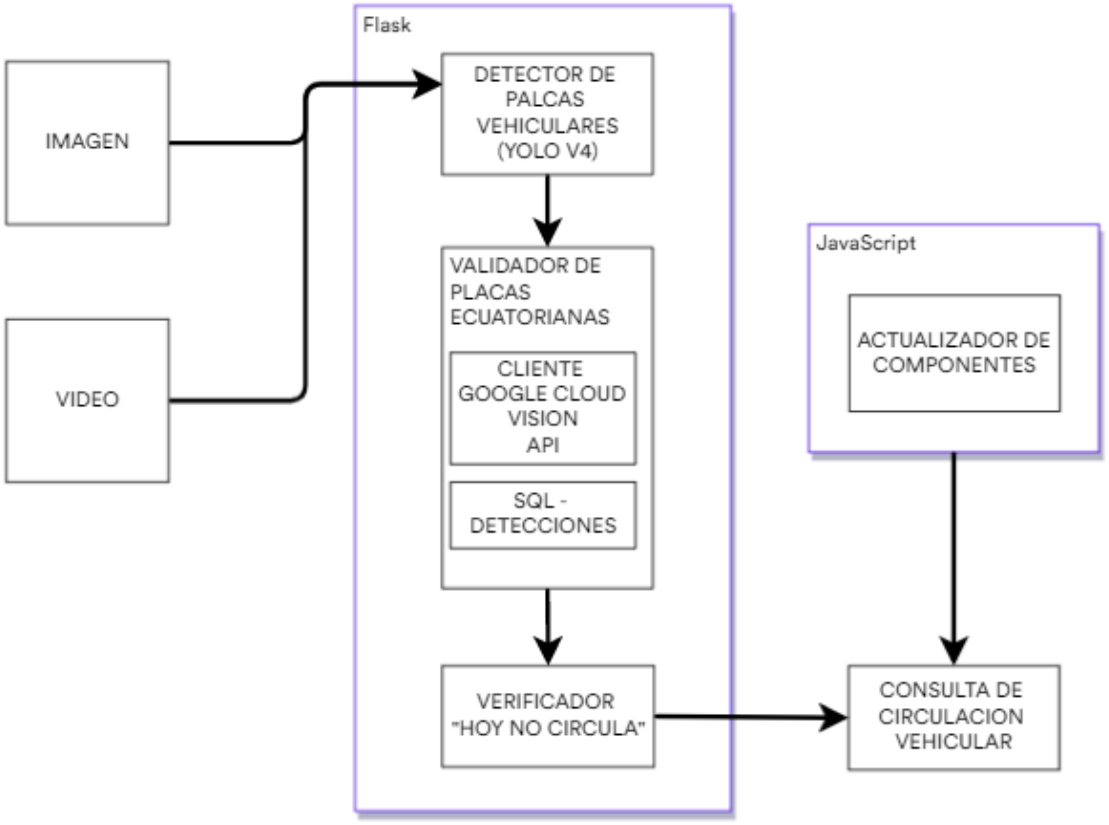
Implementación de la Plataforma de Servicio

Para la implementación de la plataforma de servicio, se optó por el caso de prueba de la normativa “Hoy no circula”, con el fin de validar si los vehículos de las instalaciones de la Universidad de las Fuerzas Armadas ESPE pueden circular en Quito, evitando multas o retención de vehículos en jornadas laborales.

Este proceso requirió la implementación de un módulo capaz de actualizar los componentes en tiempo real, para lo cual se agregaron funciones mediante JavaScript, como se puede observar en el modelo de arquitectura (**Figura 57**).

Figura 57.

Modelo de arquitectura del prototipo de software con caso de uso “Hoy No Circula”



Como se puede apreciar, se solicita una fotografía o se puede cargar una fuente de video (**Figura 58**), el algoritmo toma el tiempo y fecha actual, y lo contrasta contra las configuraciones de la restricción “Hoy no Circula” cargadas en la interfaz de configuración (ver **Figura 59**). Otra adición fue la inclusión de porcentajes de uso de componentes del servidor, al igual que información como tiempo de procesado de solicitud y nivel de confianza de la red neuronal, información que es respaldada en una base de datos, para su próximo análisis.

Figura 58.

Interfaz para consulta de circulación de vehículos mediante imagen o video

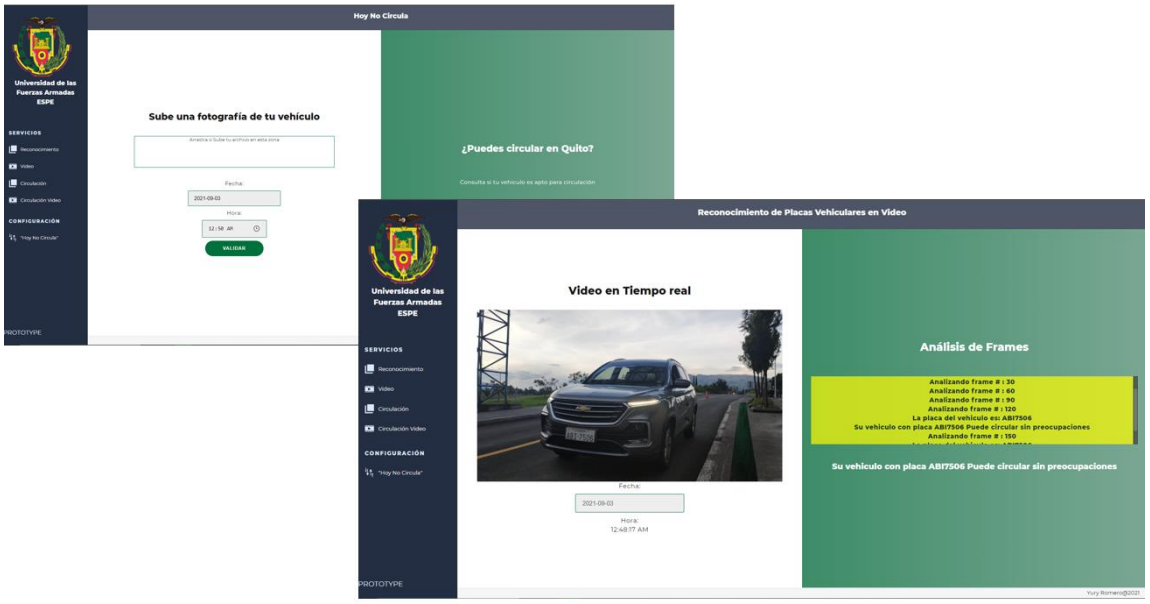
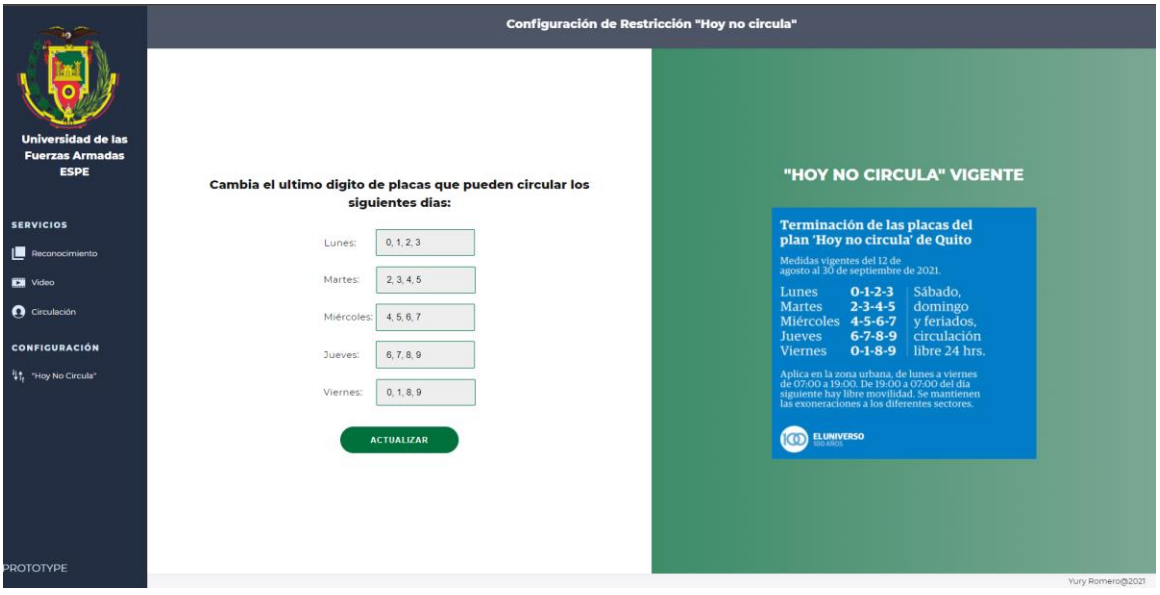


Figura 59.

Interfaz de configuración de restricciones de “Hoy no Circula”



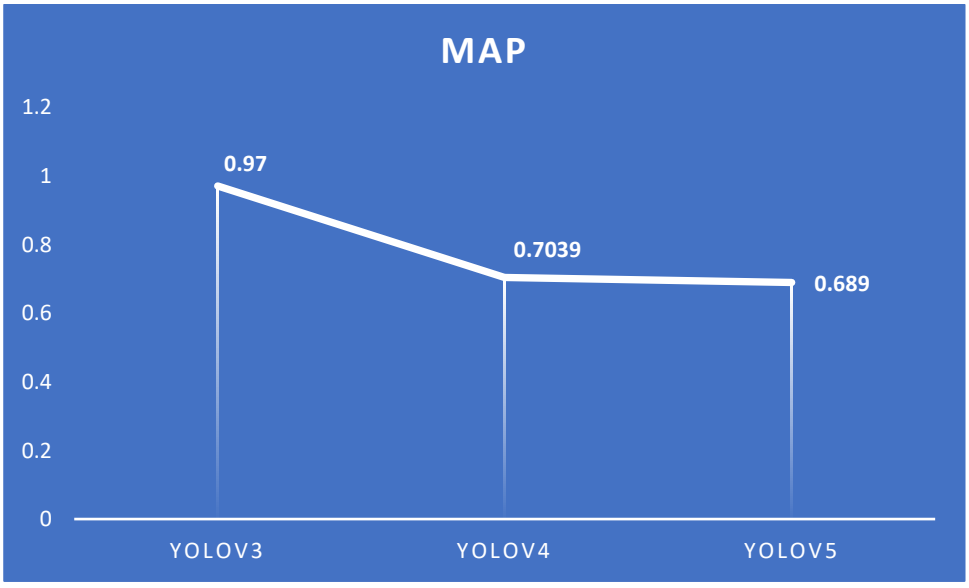
Capítulo 5

Análisis de Resultados

La metodología de desarrollo incremental aplicada en el desarrollo de este trabajo de titulación posibilitó la experimentación de distintos modelos de redes neuronales durante sus 3 primeras interacciones, lo que resultó en una base estable y confiable para la detección de placas vehiculares. En la **Figura 60** se puede observar los diferentes niveles de mAP (media de promedios de precisión) conseguidos con cada red neuronal convolucional.

Figura 60.

Comparación de mAP entre las distintas versiones de YOLO utilizadas



Pese que YoloV3 tiene un mejor mAP, YoloV4 tiene un mejor tiempo de respuesta de detecciones (ver los resultados de la **Tabla 3**). Por esta razón el prototipo siguió su desarrollo con el uso de YoloV4 al ofrecer una solución con un promedio de detección de 95% sobre el grupo de prueba.

Durante la cuarta y quinta iteración se abarcaron los problemas de reconocimiento de placas vehiculares, con algoritmos de validación de placas vehiculares ecuatorianas y delimitadores de área para los caracteres que conforman la placa vehicular, dando como resultado un promedio de lectura del 36% al hacer uso de Tesseract OCR, lo cual no se encuentra dentro de un rango aceptable y requirió de ciertas mejoras, como un validador de placas vehiculares(respetando el formato propuesto por la ANT) y la segmentación de caracteres dependiendo de la cantidad de pixeles que maneje la imagen.

Las imágenes utilizadas en este prototipo imitan un ambiente generalizado en aspectos de luz, posicionamiento de cámara y ángulos de inclinación a fin de cubrir un margen mayor de posibilidades. Para un óptimo funcionamiento, las placas vehiculares deben ser capturadas de manera frontal y con exposición a luz para evitar cualquier distorsión de imagen, esto permite facilitar al algoritmo de Otsu (Diferenciación de Planos) separar el fondo de la placa (fondo blanco) de las letras.

Dentro de la sexta iteración, el consumo del API de servicio de Google permitió la mejora sustancial del reconocimiento de caracteres, aumentando el margen de acierto de reconocimiento de caracteres de un 36% a un 73.17% dentro del grupo de datos de prueba utilizado.

La información de prueba utilizada durante la última iteración proviene de la entrada del parqueadero de la Universidad de Las Fuerzas Armadas ESPE campus Sangolquí, presentando resultados en un promedio de 2.1 segundos por imagen procesada.

En el procesamiento de video en “tiempo real” se utilizó dos opciones de corte, un corte cada 30 frames y otro cada 15 frames, siendo este último la mejor opción para

reconocimiento de placas vehiculares, pues al tener más información, el algoritmo de validación de placas tiene mayor cantidad de opciones para buscar la coincidencia.

Después de realizar varias pruebas, el OCR de Google obtuvo resultados sobresalientes ante Tesseract, al leer placas vehiculares con mayor acierto. El procesamiento realizando a las imágenes antes de ser enviadas a los OCR es similar, con diferencia de la reversión de colores de imágenes como se presenta en la **Figura 61.**

Figura 61.

Resultados de Binarización e Inversión de Colores

Al trabajar en la implementación del servicio, se encontraron problemas con el procesamiento de video en tiempo real, puesto que la disposición de FLASK para esta clase de tareas requiere hacer uso de tecnologías complementarias, como JavaScript para el manejo de componentes aledaños al video que se presentan en pantalla.

Capítulo 5

Conclusiones y Recomendaciones

Conclusiones

- Se realizó un estudio sobre el uso de visión por computadoras y si aplicación en el campo del reconocimiento de texto, encontrando que el uso de un OCR es fundamental para reconocimiento de texto en 4 de los 5 estudios primarios, a su vez 2 de estos estudios se orientan al trabajo con redes neuronales.
- Basado en el estudio de los métodos de identificación vehicular, se encontraron los métodos de delimitación, reconocimiento óptico de caracteres y el método de extracción para caracteres; se optó por el método de reconocimiento óptico de caracteres, al presentar mejores resultados al realizar el reconocimiento individual de cada carácter que compone la placa vehicular.
- Se logró desarrollar un prototipo de software que permite la identificación de placas vehiculares con el uso de una red neuronal convolucional YoloV4 para la detección de placas, y Cloud Vision API junto a OpenCV para el reconocimiento de caracteres.
- La red neuronal utilizada en el presente prototipo maneja un rango del 81% al 99% en niveles de confianza de detección, el porcentaje de acierto del OCR es de un 73.13% dentro del grupo de pruebas.
- Con el análisis de los diferentes tipos de pruebas, el porcentaje de acierto del OCR del 73.13% se debe a que dentro del grupo de pruebas existen imágenes cuyas placas no manejan la iluminación adecuada, o poseen altos niveles de distorsión afectando así el reconocimiento de

caracteres de la placa, para la identificación de la misma, el rango de confianza de la red neuronal varía a causa de la distancia a la que se encuentra la placa vehicular de la cámara, pues el entrenamiento no se llevó a cabo con imágenes donde la placa vehicular se encontrara muy alejada de la cámara.

Recomendaciones

- Hacer un estudio del entrenamiento de Tesseract OCR sobre el tipo de fuente manejado en placas vehiculares ecuatorianas, para mejorar sus de reconocimiento de texto.
- Tomar fotografías o videos de forma estable, a una distancia donde la placa vehicular sea legible de forma horizontal y con buenos niveles de luz.
- Separar el grupo de pruebas por características, tales como: iluminación, distorsión, ángulos de inclinación de placa.
- Incluir mallas curriculares en la carrera que tengan que ver con técnicas de inteligencia artificial.

Referencias

Akshayan, R., Vishnu Prashad, S., Soundarya, S., Malarvezhi, P., & Dayana, R. (2018). Review Paper: Licence Plate and Car Model Recognition. *Cognitive Informatics and Soft Computing*, 307-314.

Almache, M. (2017). GUIA-IA Universidad De lAs Fuerzs Armadas ESPE. *Inteligencia Artificial II*.

ANT. (2014). *Vehículos Matriculados – Serie Histórica 2008-2014.* Obtenido de Ecuador en Cifras: https://www.ecuadorencifras.gob.ec/documentos/web-inec/Estadisticas_Economicas/Estadistica%20de%20Transporte/Vehiculos_Matr_2008-2014/2008-2013_VehiculosMatriResultados.pdf

Bansal, S., Gupta, M., & Tyago, A. (2019). Building a Character Recognition System for Vehicle Application. *Advances in Decision Sciences, Image Processing, Security and Computer Vision*, 161-168.

Bismart. (2020). *Diferencias entre Machine Learning y Deep Learning*. Obtenido de Bismart Blog: https://blog.bismart.com/es/diferencia-machine-learning-deep-learning

Chaturvedi, A. (11 de Octubre de 2020). *Understanding Nvidia TensorRT for deep learning model optimization*. Obtenido de Medium: https://medium.com/@abhaychaturvedi_72055/understanding-nvidias-tensorrt-for-deep-learning-model-optimization-dad3eb6b26d9

Chithra, P., & Prashanthi, B. (2017). Feature Based Multiple Vehicle Licence Plate Detection and Video based Traffic Counting. *Smart and Innovative Trends in Nex Generation Computing Technologies*, 918-931.

Delgado, M. (1997). *La Inteligencia Rtificial Realidad de un mito moderno.* Barcelona: Universidad de Granada.

Deng.Li, & Yu, D. (2014). Deep Learning: Methods and Applications. *Now The Scsence of Knowledge*, 197-387.

Dennet, D. C. (1984). Cognitive wheels: the frame problem of AI. *Minds, Machines and Evolution*, 129-150.

Ekos. (11 de Marzo de 2019). *Quito y Guayaquil entre las ciudades con más tráfico de América Latin*. Obtenido de Eko Negocioes: https://www.ekosnegocios.com/articulo/quito-y-guayaquil-entre-las-ciudades-con-mas-trafico-de-america-latina

ElComercio. (19 de Febrero de 2019). *La automatización de Cadisan y la red de estacionamientos avanza en Quito.* Obtenido de El comercio: https://www.elcomercio.com/actualidad/automatizacion-cadisan-estacionamientos-quito-mantenimiento.html

ElComercio. (25 de Enero de 2019). *Nuevo sistema automatizará parqueaderos municipales de Quito*. Obtenido de El comercio: https://www.elcomercio.com/actualidad/quito-sistema-automatizacion-parqueaderos-municipio.html

ElComercio. (24 de Octubre de 2019). *Problemas para hallar estacionamiento en La Mariscal*. Obtenido de El comercio: https://www.elcomercio.com/actualidad/problemas-estacionamiento-mariscal-vehiculos-cuidadores.html

ElUniversoEC. (18 de Marzo de 2019). *Expectativa del mercado automotor de Ecuador para el 2019 en comparación con el 2018.* Obtenido de El Universo:

https://www.eluniverso.com/noticias/2019/03/18/nota/7240196/expectativa-mercado-automotor-ecuador-2019-comparacion-2018

Garcia, L. (2013). *¿Qué es OpenCV?* Obtenido de Un Poco de Java: https://unpocodejava.com/2013/10/09/que-es-opencv/

He, K., Zhang, X., Sun, J., & Ren, S. (2015). Deep Residual Learning for Image Recognition. *arXiv*.

Hùng, V. (10 de Agosto de 2020). *tensorflow yolov4 tflite*. Obtenido de GitHub: https://github.com/hunglc007/tensorflow-yolov4-tflite

IBM. (2016). *Computer Vision*. Obtenido de IBM: https://www.ibm.com/topics/computer-vision

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet Classification with Deep Convolutional. *Neurips*.

LeCun, Y., Bengio, Y., & & Hinton, G. (2015). Review Deep Learning. *Deep Learning Nature*, 436-444.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based learning Applied to Document Recognition. *I.E.E.E.*

Leszczuk, M., Janowski, L., Romaniak, P., Glowacz, A., & Mirek, R. (2011). Quality of experience, content distribution, real time(live) content distribution. *Multimedia Communications, Services and Security*, 11-18.

Lucero, K. (15 de Octubre de 2020). *Mientras el transporte publico sea deficiente, el parque automotor seguira engordando.* Obtenido de Revista Gestion: https://www.revistagestion.ec/sociedad-analisis/mientras-el-transporte-publico-sea-deficiente-el-parque-automotor-seguira

Mahalakshmi, S., & Sendhil, R. (2018). Smart Toll Collection Using Automatic Licence Plate Recognition Techniques. *Computing, Analytics and Networks*, 31-41.

Neli, F. (2018). Image Analysis and Computer Vision with OpenCV. *Pythoon Data Analytics*, 507-535.

Nemec, D., Janota, A., & Rastislav, P. (2017). System for Monitoring and Guarding Vehicles on Parking Areas. *Smart Solutions in Today's Transport*, 307-319.

Olah, C. (2014). *Convolutional Neural Networks: The Biologically-Inspired Model*. Obtenido de Code Mentor: https://www.codementor.io/@james_aka_yale/convolutional-neural-networks-the-biologically-inspired-model-iq6s48zms

OpenCV. (2013). *Background Subtraction.* Obtenido de Open Cv Python Tutorials: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html#background-subtraction

OpenCV. (2019). *OpenCV: How to Use Background Subtraction Methods.* Obtenido de Open CV: https://docs.opencv.org/master/d1/dc5/tutorial_background_subtraction.html

Puarungroj, W., & Boonsirisumpun, N. (2018). Thai License Plate Recognition Based on Deep Learning. *sciencedirect*, 8.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *CVPR*. Obtenido de https://pjreddie.com/media/files/papers/yolo_1.pdf

Rosenfeld, A. (1988). Computer Vision: Basic principles. *IEEE*, 863-868.

Rouhiainen, L. (2018). *Iteligencia artificial 101 cosas que debes saer hoy sobre nuestro futuro.* Barcelona: Planeta Libros.

Shokrolah Shirazi, M. &. (2015). Vision-Based Vehicle Counting with High Accuracy for Highways with Perspective View. *Lecture Notes in Computer Science (Vol. 9475)*.

Silca, S. M., & Jung, C. (2018). Licence Plate Detection and Recognition in Unconstrained Scenarios. *Computer Vision-ECCV 2018*, 593-609.

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR*.

Threat, R. (2017 de Noviembre de 2017). *Code to light*. Obtenido de Getting started with PyTorch for Deep Learning: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

Zambak, A. F. (2013). The Frame problem. *Philosophy and Theory of Artificial Intelligence. Studies in Applied Philosophy, Epistemology and Rational Ethics*, 307-319.

Zherzdev, S., & Gruzdex, A. (2018). LPENet: Licence Plate recognition via Deep Neural Networks. *Computer Vision and Pattern recognition*.

Anexos