# Prototipo de reconocimiento automático de placas vehiculares usando redes neuronales profundas sobre un microcontrolador

Juan José Vargas Estevez y William Fernando Moncada Carreño

Trabajo de grado en la modalidad de investigación Presentado para optar por el título de Ingeniero Electrónico

Director:

M.Eng. Jaime Guillermo Barrero Pérez

Maestría en ingeniería área de electrónica

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2021

## Contenido

**Pág.**

## Lista de Figuras

**Pág.**

## Lista de Tablas

**Pág.**

## Lista de Apéndices

**Pág.**

## Resumen

**Título**: Prototipo de reconocimiento automático de placas vehiculares usando redes neuronales profundas sobre un microcontrolador[*]

**Autores**: Juan José Vargas Estevez, William Fernando Moncada Carreño[**]

**Palabras Claves**: Procesamiento digital de imágenes, Raspberry, Sipeed MaixGo, Redes Neuronales convolucionales, Detección de objetos.

**Descripción**:

El presente trabajo de grado tuvo como propósito desarrollar e implementar un prototipo de un sistema que permitiera identificar una placa vehicular y extraer los caracteres de esta para una zona de estacionamiento vehicular.

Para comenzar se creó una base de datos de placas vehiculares, variando los parámetros de captura de la imagen, de tal forma que se pudiera obtener una base de datos robusta. Posterior a esto, se adquirieron los componentes necesarios para el desarrollo a nivel de hardware: una Raspberry Pi 3 B+ con una cámara de 5Mpx y una Sipeed MaixGo con una cámara de 2Mpx. Estos microcontroladores fueron los encargados de obtener la imagen y de su procesamiento. Por consiguiente, se crearon los algoritmos de detección y reconocimiento de placas vehiculares utilizando redes neuronales convolucionales y procesamiento de imágenes, haciendo uso de Python como lenguaje de programación y algunas librerías como OpenCV y Numpy. Por último, se implementa y prueba el desarrollo planteado para obtener el prototipo deseado.

[*] Proyecto de Grado

[**] Facultad de Ingenierías Fisicomecánicas Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones Director: M.Eng. Jaime Guillermo Barrero Pérez

## Abstract

**Title**: Prototype of automatic recognition of vehicular plates using deep neural networks on a microcontroller[*]

**Authors**: Juan José Vargas Estevez, William Fernando Moncada Carreño[**]

**Key Words**: Digital Image Processing, Raspberry, Sipeed MaixGo, Convolutional Neural Networks, Object Detection.

**Description**:

The purpose of this degree project was to develop and implement the prototype of a system that would allow identifying a license plate and extracting its characters for a vehicle parking area.

Firstly, there was created a vehicle license plate database, varying the image capture parameters, in such a way that a robust database could be obtained. After that, the necessary components for hardware-level development were acquired: a Raspberry Pi 3 B + with a 5Mpx camera and a Sipeed MaixGo with a 2Mpx camera. These microcontrollers were in charge of obtaining the image and its processing. Consequently, the vehicle license plate detection and recognition algorithms were created using convolutional neural networks and image processing, making use of Python as a programming language and some libraries such as OpenCV and Numpy. Finally, the proposed development was implemented and tested to obtain the desired prototype.

[*] Project of grade

[**] Facultad de Ingenierías Fisicomecánicas Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones Director: M.Eng. Jaime Guillermo Barrero Pérez

# Introducción

El continuo avance en el uso de la visión por computadora ha mejorado la detección de objetos como una técnica para localizar la posición de objetos en imágenes o videos. El objetivo de esta técnica es replicar la capacidad de los humanos cuando observamos imágenes o videos, reconociendo o localizando los objetos de interés. Esto se puede lograr adquiriendo cierto nivel de comprensión del contenido de una imagen mediante un dispositivo de cómputo. Lo anterior entra como un campo activo de lo que se conoce como *Machine Learning* y más específicamente *Deep Learning*.

La identificación de una placa vehicular se puede realizar de forma manual, o mediante un dispositivo; sin embargo, haciendo esto manualmente el sistema presenta falencias y retrasos debido a las personas que realizan dicha operación. Ahora dado que la visión por computadora es ampliamente usada, concretamente en la detección de objetos, los dispositivos que realizan esta función en el mercado actualmente para identificación y análisis de placas vehiculares tienen costos elevados para ser implementados en muchos parqueaderos públicos, privados y unidades multifamiliares.

Debido a esto se implementó un prototipo de un sistema de bajo costo, formado por un microprocesador y una cámara, capaz de detectar la placa vehicular en tiempo real y realizar la lectura de esta.

# 1. Objetivos

## 1.1 Objetivo General

Desarrollar e implementar un prototipo de captura, análisis y verificación en tiempo real utilizando redes neuronales para el reconocimiento de placas vehiculares.

## 1.2 Objetivos Específicos

- Seleccionar una arquitectura de red neuronal profunda para el procesamiento de imagen.
- Crear una base de datos de placas vehiculares para el desarrollo propuesto.
- Entrenar y validar la arquitectura seleccionada para el reconocimiento de placas vehiculares.
- Construir el prototipo de seguridad con base en el microcontrolador seleccionado.

# 2. Marco Teórico

## 2.1 Machine Learning

El *Machine Learning* o aprendizaje automático es una rama del campo de la inteligencia artificial que busca dotar a una máquina con la capacidad de "aprender", mediante la adaptación de ciertos algoritmos de su programación respecto a cierta entrada de datos en su sistema.

## 2.2 Tipos de machine learning

El procedimiento informático del aprendizaje automático utiliza experiencia y evidencia en forma de datos, a través de los cuales puede comprender patrones o comportamientos por sí mismo. De esta manera puede construir predicciones de escenarios o ejecutar operaciones como soluciones a tareas específicas.

A partir de una gran cantidad de ejemplos de casos, se puede desarrollar un modelo que pueda inferir y generalizar el comportamiento que se ha observado, y a partir de él realizar predicciones para casos totalmente nuevos.

Existen tres tipos principales de aprendizaje automático que se dividen en función de la salida de estos. Algunos tipos de algoritmos son:

### *2.2.1 Aprendizaje supervisado*

Este tipo de aprendizaje se basa en la denominada información de formación. Se entrena al sistema proporcionándole una cierta cantidad de datos y utilizando etiquetas para definir el sistema en detalle. Es decir, el algoritmo produce una función que mapea las entradas y salidas necesarias del sistema. Un ejemplo de este tipo de algoritmo es el problema de clasificación, donde el sistema de aprendizaje intenta etiquetar (clasificar) una serie de vectores usando una de varias categorías (clases). La base de conocimientos del sistema consta de ejemplos previamente marcados.

### *2.2.2 Aprendizaje no supervisado*

Todo el proceso de modelado se realiza en un conjunto de ejemplos que consta únicamente de entradas del sistema. No hay información sobre estas categorías de muestra. Por tanto, en este caso, el sistema debe ser capaz de reconocer patrones para poder marcar nuevas entradas.

### *2.2.3 Aprendizaje reforzado*

El algoritmo aprende observando el mundo que le rodea. Su información de entrada es el *feedback* o retroalimentación que obtiene del mundo exterior como respuesta a sus acciones. Por lo tanto, el sistema aprende a base de ensayo-error.

El aprendizaje por refuerzo es el más general entre las tres categorías. En lugar de que un instructor indique al agente qué hacer, el agente inteligente debe aprender cómo se comporta el entorno mediante recompensas (refuerzos) o castigos, derivados del éxito o del fracaso

respectivamente. El objetivo principal es aprender la función de valor que le ayude al agente inteligente a maximizar la señal de recompensa y así optimizar sus políticas de modo a comprender el comportamiento del entorno y a tomar buenas decisiones para el logro de sus objetivos formales.

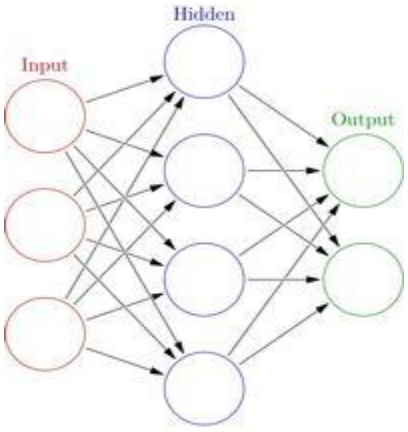
## 2.3 Deep learning

El *Deep Learning* o aprendizaje profundo es una técnica de aprendizaje automático que enseña a los ordenadores a hacer lo que resulta natural para las personas: aprender mediante ejemplos. Este consiste en neuronas artificiales en redes de múltiples capas que, a través de algoritmos, pueden enseñar al software a entrenarse para reconocer objetos como el lenguaje y las imágenes (Universidad Industrial de Santander, 2014). La capacidad de una máquina para "aprender" con experiencia en lugar de programación es parte de la inteligencia artificial.

Siguiendo el modelo de la ciencia del cerebro, el aprendizaje profundo implica alimentar una red neuronal con grandes cantidades de datos para entrenar a la máquina para la clasificación. La máquina recibe un objeto para identificar y lo procesará a través de varias capas de red. A medida que continúa el proceso, la máquina pasa de capas simples a capas más complicadas hasta que se llega a una respuesta. Los algoritmos instruyen a las neuronas sobre cómo responder para mejorar los resultados (MathWorks, n.d.).

**Figura 1.**

*Red neuronal artificial con coloración de capas*




Nota. Tomado de: Por Glosser.ca [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], vía Wikimedia Commons.

## 2.4 Computación en la nube

Suministro diseñado para proporcionar servidores, almacenamiento, bases de datos, redes, software, análisis e inteligencia a través de Internet ("nube") (Microsoft, n.d.). Este tiene como objetivo proporcionar una innovación más rápida, recursos flexibles y economías de escala.

## 2.5 Google Colab

Servicio en la nube basado en Jupyter Notebooks, que permite el uso gratuito de GPU y TPU de Google, y proporciona distintas bibliotecas tales como: Scikit-learn, PyTorch, TensorFlow, Keras y OpenCV. Google lanzó esta plataforma de aprendizaje automático que

proporciona servicios modernos de aprendizaje automático con modelos previamente entrenados y un servicio para generar modelos personalizables.

*Compute Engine* proporciona una unidad de procesamiento de gráficos (GPU) las cuales pueden agregar instancias de máquinas virtuales. Estas GPU pueden acelerar cargas de trabajo específicas en su instancia, como el aprendizaje automático y el procesamiento de datos (Google Cloud, 2020). Siendo asignada para el entrenamiento o procesamiento la NVIDIA® Tesla® T4nvidia-tesla-t4.

## 2.6 Tensor Flow

Consiste en una plataforma de código abierto de extremo a extremo para el aprendizaje automático. Esta cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que permiten impulsar un aprendizaje automático innovador y, por otra parte, compilar e implementar con facilidad aplicaciones con tecnología de AA (Aprendizaje Automático) (TensorFlow, 2019).

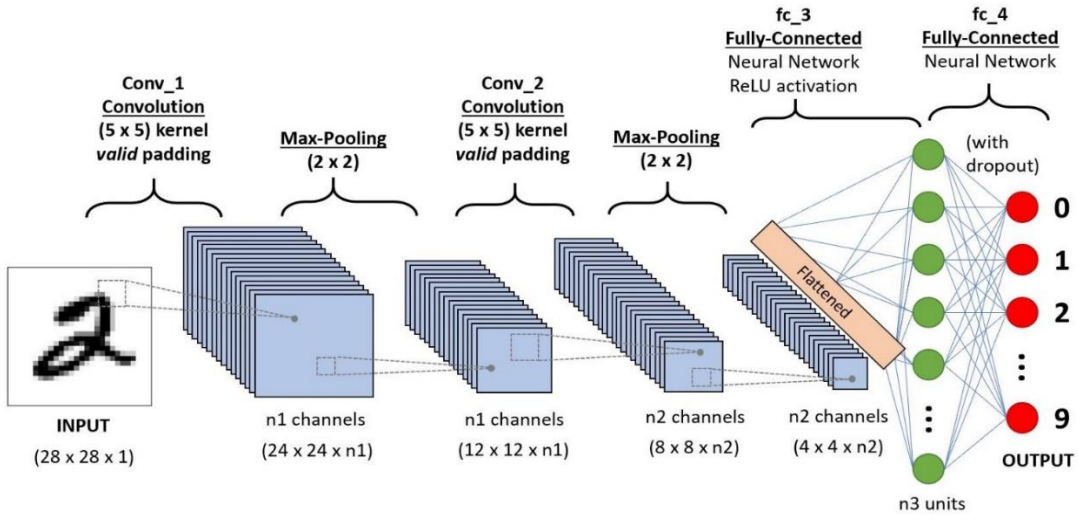
## 2.7 Redes neuronales convolucionales

Una red neuronal convolucional es una red neuronal artificial en la que las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) del cerebro biológico. Este tipo de red es una variante del perceptrón multicapa, pero dado que su aplicación se realiza en una matriz bidimensional, son muy efectivas para tareas de visión artificial (como clasificación y segmentación de imágenes) y otras aplicaciones.

El aprendizaje profundo de redes neuronales convolucionales (CNN) permite el análisis y el modelado automáticos de características abstractas y produce mejores resultados que otros algoritmos supervisados y no supervisados en visión por computadora. Estos tipos de redes son modelos de tipo *feed-forward* y tienen imágenes RGB como entrada, que se propagan hacia adelante a través de varias neuronas por medio de pesos y parámetros de aprendizaje. La arquitectura de estas redes se basa en módulos que extraen características y capas para su clasificación.

**Figura 2.**

*Una secuencia de CNN para clasificar dígitos escritos a mano.*




Nota. Tomado de: Por Sumit saha, 2018, [CC BY-SA 3.0 (https://miro.medium.com/max/2400/1*uAeANQIOQPqWZnnuH-VEyw.jpeg)].

Actualmente, existe una gran cantidad de arquitecturas CNN aplicadas a la Segmentación Semántica (SS). Asignan etiquetas de L = {$l1$, $l2$, ..., $ln$} a cada elemento de un conjunto de

variables de entrada aleatorias l, $X = \{x1 , x2, ..., xN\}$. Cada etiqueta l representa una clase u objeto con diferente significado semántico, como avión, vehículo, señal de tráfico o fondo. Por otro lado, la variable X es la imagen de la secuencia de video (Soto Orozco et al., 2019).

### *2.7.1 Arquitectura CNN*

Las redes neuronales convolucionales consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general, se añade una función para realizar un mapeo causal no-lineal.

Como red de clasificación, comienza con la etapa de extracción de características, que se compone de neuronas convolucionales y neuronas de reducción de muestras; por tanto, se encuentran neuronas de perceptrón simple al final de la red para realizar la clasificación final de las características extraídas. La etapa de extracción de características es similar al proceso de estimulación en las células de la corteza visual. Esta etapa consta de capas alternas de neuronas convolucionales y neuronas de reducción de muestras. A medida que los datos progresan en esta etapa, su dimensionalidad disminuye y las neuronas en la capa lejana son menos sensibles a la interferencia de los datos de entrada; pero al mismo tiempo son activadas por características cada vez más complejas.

Después de una o más etapas de extracción de características, los datos finalmente llegan a la etapa de clasificación. En ese momento, los datos se refinarán en una serie de características únicas de la imagen de entrada, y ahora la tarea de esta etapa final es poder clasificar estas características en una u otra etiqueta según el objetivo del entrenamiento.

## 2.8 Placas vehiculares de Colombia

En Colombia los vehículos automotores llevarán dos (2) placas iguales: una en el extremo delantero y otra en el extremo trasero. Los remolques, semirremolques y similares de transporte de carga tendrán una placa conforme a las características que determine el Ministerio de Transporte (ALCALDÍA MAYOR DE BOGOTÁ DC, 2012). Las motocicletas, motociclos y moto triciclos llevarán una sola placa reflectiva en el extremo trasero con base en las mismas características y seriado de las placas de los demás vehículos. Estas matriculas o placas sirven para identificar el vehículo, por lo tanto, deben estar registradas en el registro único nacional de tránsito (RUNT), lo que las convierte en información de acceso público (Secretaría General del Senado, 2020) (Corte Constitucional, 2011).

### *2.8.1 Contenido de la placa*

Para vehículos de servicio público, particular y oficial, llevará en forma horizontal seis (6) caracteres en relieve: tres (3) letras y tres (3) dígitos o cuatro (4) letras y dos (2) dígitos, en dos (2) grupos separados por el logotipo del ministerio de transporte: en la parte inferior el nombre del municipio donde se encuentra radicada la cuenta del vehículo, y un marco que bordea la placa,

### *2.8.2 Dimensiones de la placa*

Las dimensiones de las placas para vehículos automotores deberán tener un tamaño de 330 mm de ancho, por 160 mm de alto.

Las placas para motocicletas deberán tener un tamaño de 235 mm de ancho, por 105 mm de alto (Fiscalía General de la Nación, n.d.).

A excepción de las placas de servicio diplomático, consular y de misiones especiales las cuales para vehículos automotores deberán tener un tamaño de 303.6 mm de ancho, por 151.2 mm de alto y las placas para motocicletas deberán tener un tamaño de 210 mm de ancha, por 120 mm de alto.

### *2.8.3 Clasificación de las placas*

Las placas se clasifican, debido al servicio del vehículo, así: de servicio oficial, público, particular, diplomático, consular y de misiones especiales. Las placas de servicio diplomático, consular y de misiones especiales serán suministradas por el Ministerio de Transporte o por la entidad que delegue para tal fin, a través del Ministerio de Relaciones Exteriores. A continuación, se muestran los tres tipos de placas más comunes, y hacia los cuales se enfoca este proyecto.

Placa de servicio particular. Fondo: amarillo 124 C. Caracteres: negro C. Logotipo y marco: negro C.

**Figura 3.**

*Placa de automóvil de servicio particular.*

Placa para servicio público. Fondo: amarillo 124 C. Caracteres: negro C. Logotipo y marco: negro C.

***Figura 4.***

*Placa de automóvil de servicio público.*


Placa para motocicletas y similares. Fondo: Amarillo 124 C. Caracteres: Negro C. Logotipo y marco: Negro C.

**Figura 5.**

*Placa para motocicletas.*

# 3. Procedimiento

El proceso para la identificación de placas vehiculares se realizó en dos etapas. La primera consistió en la detección de la placa haciendo uso de redes neuronales convolucionales (CNN) utilizando TensorFlow. La segunda fue la lectura de caracteres de la placa, y esta se efectuó de dos formas. La primera se realizó a través de la segmentación de la placa y caracteres de la misma mediante OpenCV para su posterior lectura con Tesseract. Mientras que la segunda forma se realizó por medio de la API Cloud Vision proporcionada por Google. Este proceso se hizo de esta manera debido al fácil manejo del procedimiento y la complejidad de la tarea. Pues realizarlo de manera más directa (*End to End*) requeriría de una construcción de red neuronal mucho más compleja donde exista una o más de estas redes conectadas en cascada; asimismo, este tipo de proceso requiere una cantidad de datos demasiado elevada.

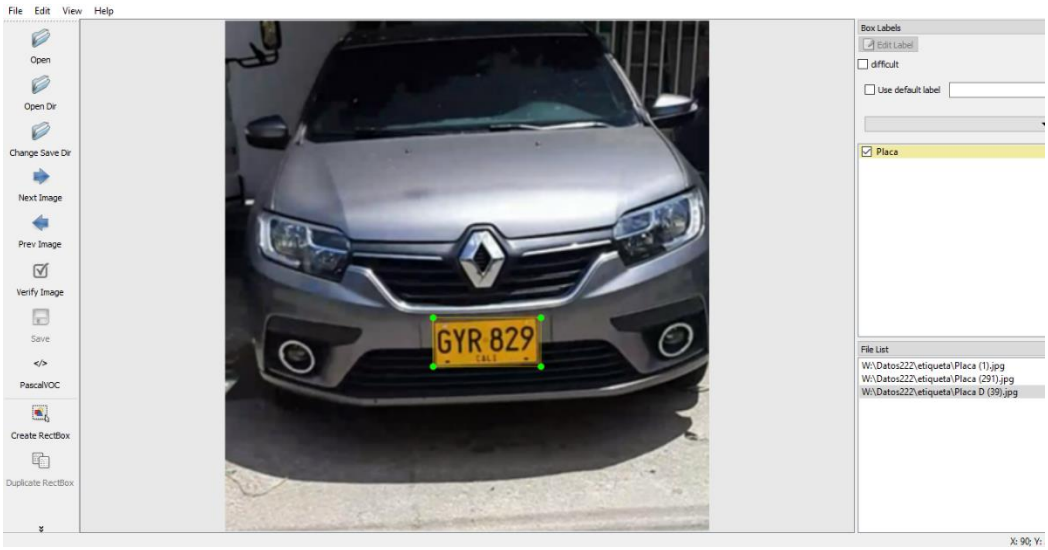
## 3.1 Primera Etapa

La primera etapa, cómo ya se mencionó, consistió en la detección de la placa para su posterior segmentación. Para esto hay múltiples formas de hacerlo y entornos de trabajo en los cuales se pueden desarrollar. Sin embargo, por comodidad y accesibilidad se hizo uso de TensorFlow como entorno de trabajo y de Google Colaboratory como compilador, este último es un servicio en la nube basado en los Notebooks de Jupyter y provee una GPU para el entrenamiento de la red neuronal (Bisong, 2019).

La Detección del Objeto (*Object Detection*) se da mediante la Localización del Objeto (*Object Localization*) y la Identificación del Objeto (*Object Identification*). La localización hace referencia a la ubicación del objeto en una imagen o video, mientras que la identificación se da con la asignación del objeto, es decir, su etiqueta (Elisha, 2020). Para realizar lo anterior se necesitan imágenes con su etiquetado correspondiente. Entonces, se creó una base de datos con aproximadamente 1000 imágenes, todas en formato JPG (esto debido a un conflicto con librerías de TensorFlow) y se usó LabelImg (tzutalin, 2021) para su etiquetado. Este último es un programa de código abierto que da facilidad a la hora de generar dichas etiquetas, y en este caso se usó el formato Pascal VOC (Renu Khandelwal, 2019) para el etiquetado. Lo anterior se puede ver en la Figura 1, en donde se establece el formato que se usará para la etiquetación, posteriormente se selecciona el contorno que representa la posición de la placa en la imagen, y finalmente se genera una etiqueta para su identificación. Cabe resaltar que para el entrenamiento la base de datos se debe dividir en dos conjuntos: entrenamiento y validación. De acuerdo con la literatura, las formas de dividir la base de datos dependen del tamaño de esta, llevándose la mayor parte el conjunto de entrenamiento. En este caso la mejor opción para tener una mayor precisión en el modelo se da dividiéndola en 70% para el conjunto de entrenamiento y 30% para el conjunto de validación (para la verificación del modelo se usaron imágenes o video en vivo que no están contenidas dentro de la base de datos). Luego se dejan los archivos XML de la base de datos en formato TFRecord, que es el tipo de formato con el que TensorFlow se ejecuta.

**Figura 6.**

*Generación de etiquetas en imágenes mediante LabelImg.*




Lo siguiente constituyó una parte fundamental, que es escoger la arquitectura de red adecuada. TensorFlow nos ofrece varios modelos ya preentrenados con diferentes arquitecturas de acuerdo con las características que se necesiten (tensorflow, 2020). Dado que el modelo debe ejecutarse y correr de manera óptima sobre un microcontrolador, se hace necesario que este (el modelo) se convierta posteriormente en formato de TensorFlowLite. Esto es porque está optimizado para aplicaciones móviles y perimetrales, y los pesos generados son reducidos; por consiguiente, la arquitectura que se escoja debe estar soportada por este formato. Con lo anterior se revisan los modelos soportados, los cuales son los SSD Models, y de estos se escogió la rama de MobileNet (Howard et al., 2017), más exactamente SSD MobileNet V2. De este último se eligió la versión cuantizada, que admite reducir la cantidad de memoria requerida y reduce el costo computacional, por lo que permite una ejecución más eficiente sobre la GPU (Nicholls, 2018).

A la hora de generar el entrenamiento se debe configurar el archivo .config de la arquitectura seleccionada. Este archivo contiene los parámetros de entrenamiento para la ejecución

de dicho entrenamiento. Debido a que la idea es generar *Transfer Learning* los pesos se dejaron tal cual, y sólo se modificaron la cantidad de clases, el *batch size* para generar una mayor estabilidad y las rutas de acceso; ver Apéndice 1.

Posteriormente se realizó la ejecución del entrenamiento, al cual se le hizo monitorización mediante Tensorboard. Este tuvo una duración aproximada de 12 horas y se detuvo de manera manual cuando se observó que la pérdida no variaba de manera significativa entre épocas y tiempo de entrenamiento, esto se dio en el paso 112.600. La función de pérdida generada por el conjunto de entrenamiento se puede observar en la Figura 2 con un estimado de 1.66, mientras que la función de pérdida total del modelo se observa en la Figura 3 con un 3.54. De la misma manera en la Figura 4 se observó la precisión promedio con un 66.29%; sin embargo, en la Figura 5 se vio que la precisión promedio para un 75% de IOU (*Interception Over Union*) es del 83.38%, lo que es bastante bueno. Con estos índices se congeló nuestro modelo y se hizo la conversión respectiva a TFLite.

**Figura 7.**

*Función de perdida para el conjunto de entrenamiento.*

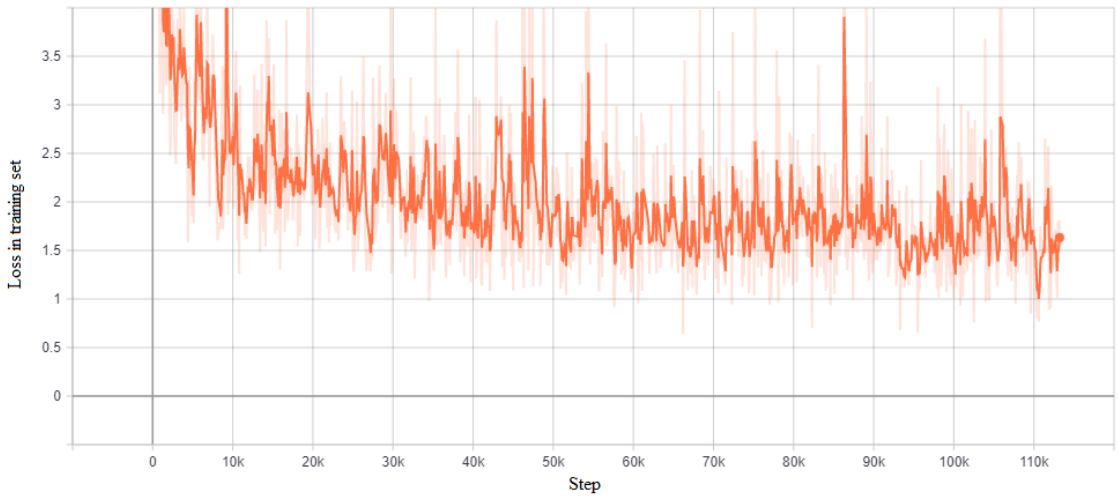


**Figura 8.**

*Función de perdida para el modelo.*

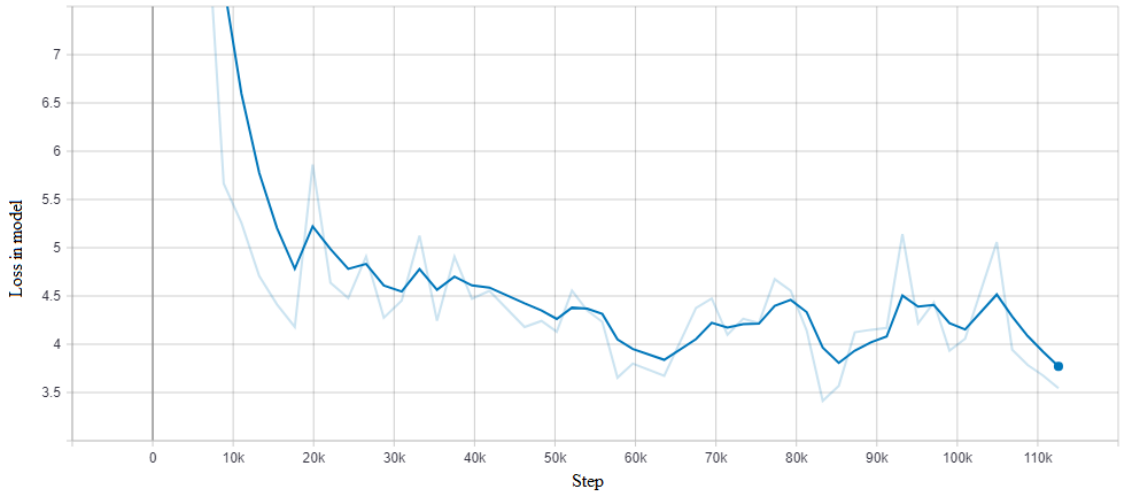



**Figura 9.**

*Precisión promedio para el modelo.*

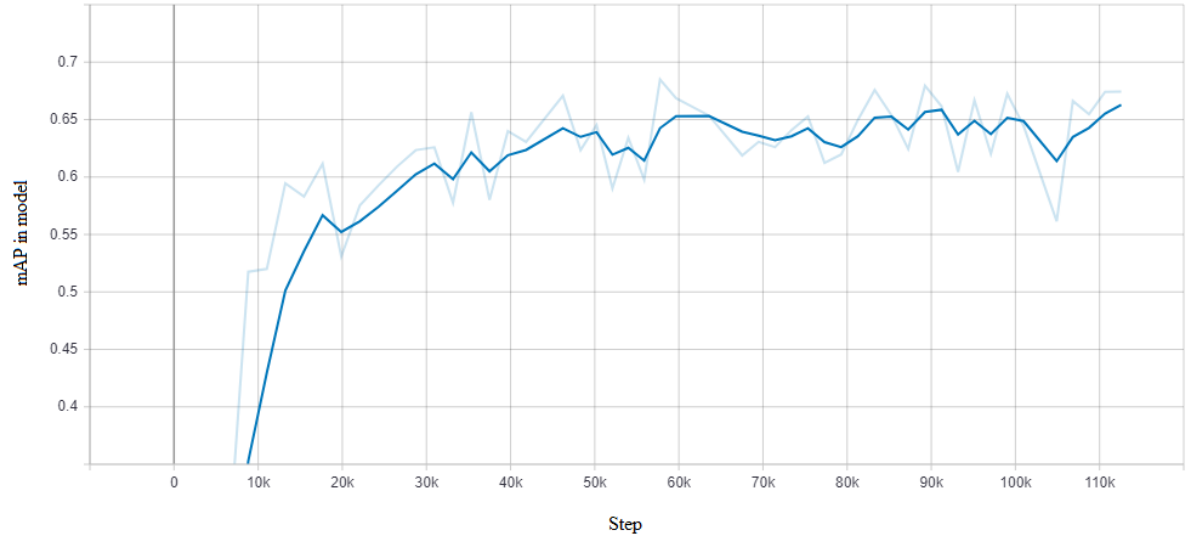
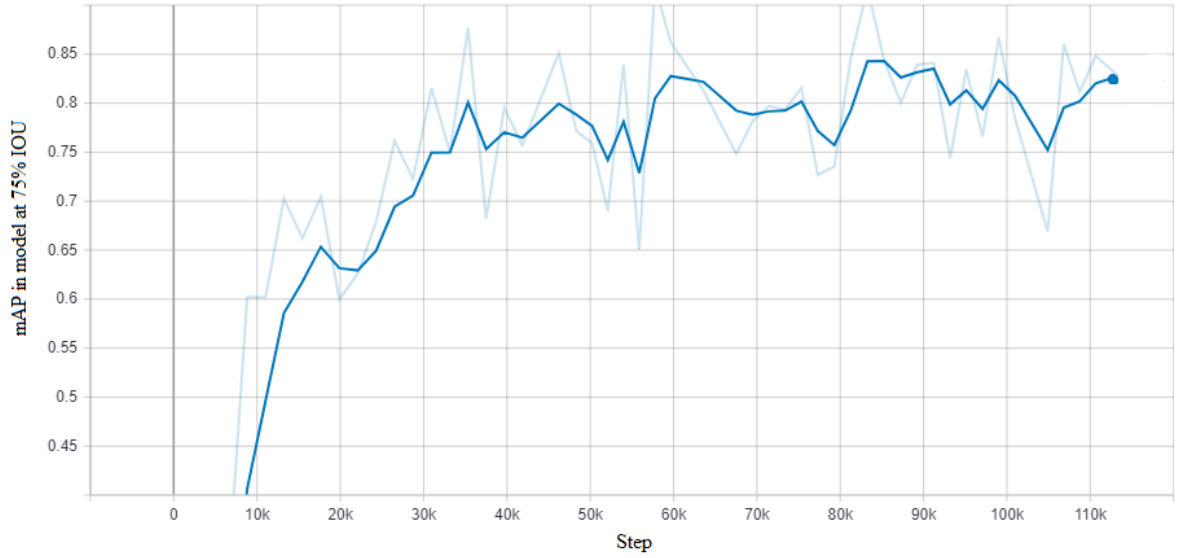


**Figura 10.**

*Precisión promedio para el modelo en un 75% de IOU.*




Para finalizar, Tensorboard da la posibilidad de ver cómo el modelo actúa sobre el conjunto de validación. En este caso se muestran ejemplos donde acierta (Figura 6); donde acierta, pero genera un falso positivo debido a la similitud (Figura 7); y, por último, uno donde es incapaz de reconocer (Figura 8), esto debido a que la imagen presenta mucho ruido y no hay datos similares suficientes en el conjunto de entrenamiento que soporten la verificación de este tipo de localización para la placa.

**Figura 11.**

*Ejecución del modelo para el conjunto de validación (acierto).*


**Figura 12.**

*Ejecución del modelo para el conjunto de validación (acierto con falso positivo).*

**Figura 13.**

*Ejecución del modelo para el conjunto de validación (desacierto).*


### *3.1.1 Selección del microcontrolador*

Una parte muy importante en el proceso de construcción del prototipo tiene que ver con el hardware, y en este caso se hizo uso de un microcontrolador como dispositivo de control. Lo ideal entonces fue encontrar uno que se acoplara a las necesidades de este proyecto, por lo cual en su desarrollo se revisaron dos en específico: Sipeed Maix Go y Raspberry Pi 3 B+.

Estos dos microcontroladores cuentan con especificaciones de diseño elevadas, las cuales son óptimas para la creación del prototipo. El Sipeed Maix Go por su parte, cuenta con todo un entorno de desarrollo para el trabajo en IA, ya que precisamente está diseñado para esto puesto que contiene un procesador dedicado para trabajo con redes neuronales convolucionales.

Adicionalmente, ofrece una portabilidad desde el inicio por su bajo consumo y batería incorporada, conectividad WiFi y diversos sensores que hacen de este un microcontrolador muy completo con un precio muy asequible (aproximadamente 40 USD). Sin embargo, al ser un hardware reciente (aproximadamente 3 años) su documentación es escasa y de difícil acceso (por el idioma), por lo cual su manejo implica un estudio amplio y existen dificultades a la hora de programar debido a que en su ecosistema algunas librerías no se pueden incluir todavía.

Por otra parte, tenemos la Raspberry Pi, en este caso el modelo 3 B+, un microcontrolador ampliamente conocido, con una potencia de cómputo alta y un manejo realmente sencillo. En este caso la mayor fortaleza proviene de la documentación y la amplia cantidad de librerías con las que se puede contar a la hora de programar. Además, cuenta con su propio sistema operativo con interfaz gráfica, lo cual facilita enormemente su manejo. Sus puntos negativos se basan en su costo (aproximadamente 60 USD) y en que viene en una versión con el hardware estándar; es decir, no viene cámara o algún otro sensor incorporado (a diferencia de la Maix) y su portabilidad depende del usuario al acoplarle una batería externa, lo cual incrementa costos. Lo ya expuesto respecto a los dos dispositivos se pueden resumir en la siguiente tabla comparativa.

**Tabla 1.**

*Comparación entre los dos dispositivos propuestos.*

| Especificaciones | Maix Go | Raspberry Pi 3B+ |
|---|---|---|
| Alimentación | 4.8V ~ 5.2V / ~600 mA | 5V / 2.5A |
| CPU | 28nm process, Dual-Core RISC-V 64bit IMAFDC / KPU K210 | Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC |
| Clock (GHz) | 0.4 | 1.4 |
| RAM(GB) | 0.008 | 1 |

| Especificaciones | Maix Go | Raspberry Pi 3B+ |
|---|---|---|
| Recursos para IA | KPU | Ninguno |
| Sistema Operativo /Lenguaje | -/Micropython | Raspian/Python |
| Conexión | WiFi | WiFi/Ethernet |
| Accesorios adicionales | Cámara DVP Lente M12<br>Antena IPEX<br>Pantalla LCD Táctil 2.8 pulgadas<br>Batería de Litio<br>Micrófono FPC10 | Cargador de 5V |
| Precio (Amazon) | ~$ 40 dólares | ~$ 60 dólares |

Para tomar una decisión respecto a cuál usar, primero se hizo una prueba de funcionalidad con base al modelo ya obtenido para observar su comportamiento en los dos microcontroladores. Cabe aclarar que el modelo que la Maix Go acepta está únicamente en el formato .kmodel, que es consecuencia de la arquitectura K210 del microcontrolador. Hacer la conversión es un proceso tedioso y algo complicado debido a que tiene sus propias estructuras y condiciones que son propias de los Edge IA devices (Klippel et al., 2020). Sin embargo, existe una plataforma de código abierto creada por Dmitry Maslov llamada aXeleRate (AIWintermuteAI, 2020) que consiste en una serie de scripts optimizados para ejecutarse en un Notebook de Jupyter sobre Google Colab. Estos scripts permiten un entrenamiento rápido con diferentes arquitecturas (YOLO, MobileNet, ResNet, entre otras) y una conversión automática de los modelos en el formato necesario TFLite o .kmodel para K210. Entonces, se usó aXeleRate para la conversión del modelo y poder ejecutarlo sobre la Maix Go.

Con base en lo anterior se realizaron las pruebas de rendimiento y viabilidad para video capturado en tiempo real. El sensor que se usó por parte de la Raspberry fue la PiCamera de 5 Mpx, mientras que el usado por la MaixGo fue una cámara DVP con lente M12 de 2Mpx. Los

resultados de la Raspberry se observan en la Figura 9 mientras que los arrojados por la Maix se ven en la Figura 10 (códigos en Apéndice 2). Como se observa, para un mismo vehículo la detección se da satisfactoriamente; sin embargo, es clara la superioridad de la MaixGo frente a la Raspberry, ya que su rendimiento en cuadros por segundo (FPS) es casi 20 veces mayor.

**Figura 14.**

*Modelo de detección de placas vehiculares funcionando sobre Raspberry Pi 3B+ (video en vivo).*

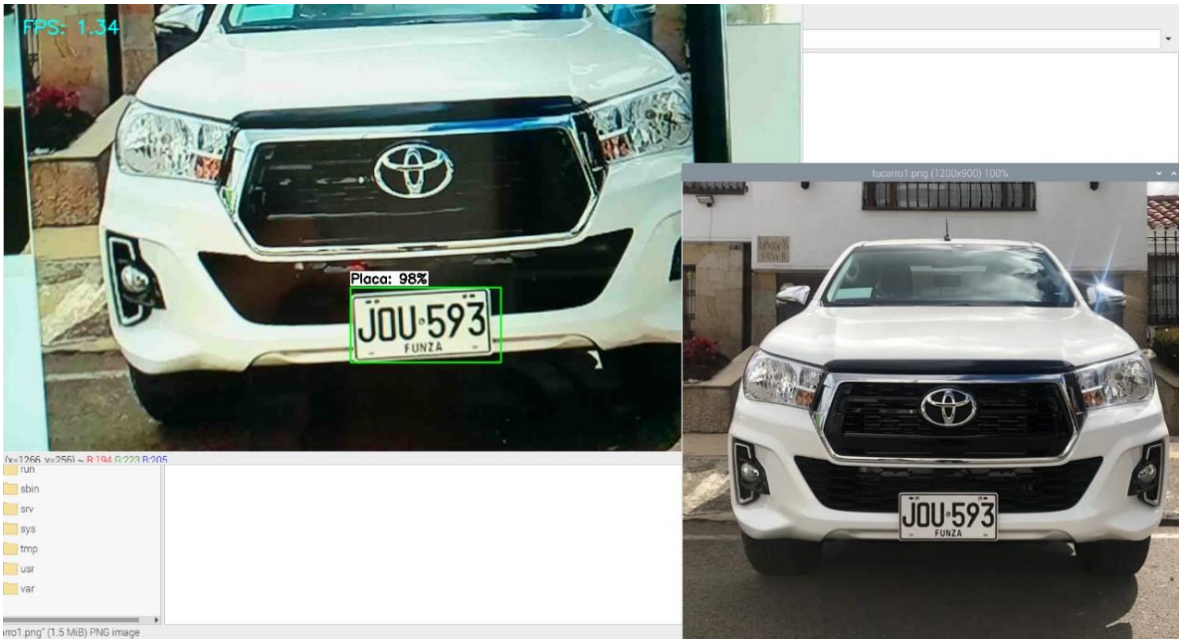


**Figura 15.**

*Modelo de detección de placas vehiculares funcionando sobre Sipeed Maix Go (video en vivo).*


Con base lo anterior lo lógico sería escoger la Sipeed Maix Go como microcontrolador para la continuación del desarrollo del prototipo. Sin embargo, como ya se mencionó anteriormente, este microcontrolador al ser relativamente nuevo no tiene una forma de adicionar y ejecutar librerías tales como OpenCV para el procesamiento de imágenes, las cuales son necesarias para poder realizar la segunda etapa con respecto a la segmentación y OCR (*Optical Character Recognition*). Entonces, por estas razones el microcontrolador seleccionado es la Raspberry Pi 3B+; sin embargo, se hará por medio de fotos fijas que serán tomadas por el hardware ya mencionado.

## 3.2 Segunda Etapa

### *3.2.1 Segmentación y Pytesseract*

Esta primera forma consta de la segmentación que se le hizo a la placa y a sus caracteres (después de ser reconocida) con el fin de que estos últimos sean reconocidos por Tesseract, el cual es un motor de reconocimiento óptico de caracteres (OCR) libre y desarrollado por Google. Este tiene una librería especificada para Python (Pytesseract) que soporta imágenes procesadas por OpenCV (Python, n.d.).

Lo primero fue segmentar la placa, para esto se extrajeron las coordenadas de la caja detectada por el modelo (lo que este considera una placa) y luego se recortó en esta misma posición; este proceso se observa en la Figura 11.

**Figura 16.**

*(a) Carro con placa a detectar. (b) Modelo ejecutado y placa detectada. (c) Segmentación de la placa detectada.*


(a) (b)

(c)

Para el reconocimiento de los caracteres de la placa, se hizo primeramente un procesamiento a la imagen con el fin de obtener la mayor precisión posible. Este se dio en el orden siguiente: primero se pasó la imagen a escala de grises y se ajustó el tamaño, con un escalado de 3 en los dos ejes aumentando así sus dimensiones. Luego se aplicó un filtro de desenfoque Gaussiano (Gaussian Blur) con el fin de reducir ruido de la imagen y seguidamente se binarizó (OpenCV, 2020a). Posterior a esto se aplicó una estructuración de la imagen para usar la operación

morfológica de dilatación (OpenCV, 2020b), estos pasos se muestran en la Figura 12 y se hacen necesarios para poder encontrar los contornos de las letras con mayor facilidad.

**Figura 17.**

*(a) Imagen en escala de grises. (b) Imagen con el filtro Gaussian Blur. (c) Imagen binarizada. (d) Imagen dilatada.*


Una vez realizados los pasos anteriores se pasó a detectar los contornos de la imagen como ya se mencionó. Posteriormente se procedió a colocar dichos contornos en cajas con el fin de tener un mejor control sobre ellos. Luego de ejecutar operaciones de manejo de áreas y relaciones de tamaño (tal como que las regiones que se tomaron debían ser de al menos 1/6 del tamaño total de

la imagen) se obtuvieron las regiones de interés (ROI) tomadas de manera correcta, y que posteriormente serían segmentadas. Todo este proceso se puede ver en la Figura 13.

**Figura 18.**

*(a) Imagen con los contornos encontrados. (b) Imagen con los contornos convertidos en cajas. (c) Imagen con la delimitación de las ROI.*


Finalmente aplicamos estas regiones sobre la dilatación ya hecha, invertimos los colores y segmentamos. Todo esto con el fin de pasar letra por letra (como se observa en la Figura 14) al ejecutor del OCR, que en este caso está hecho por Pytesseract; ver Apéndice 3.

**Figura 19.**

*(a) Caracteres segmentados con inversión de color. (b) Salida de la consola donde se reconoce la placa.*

(a)

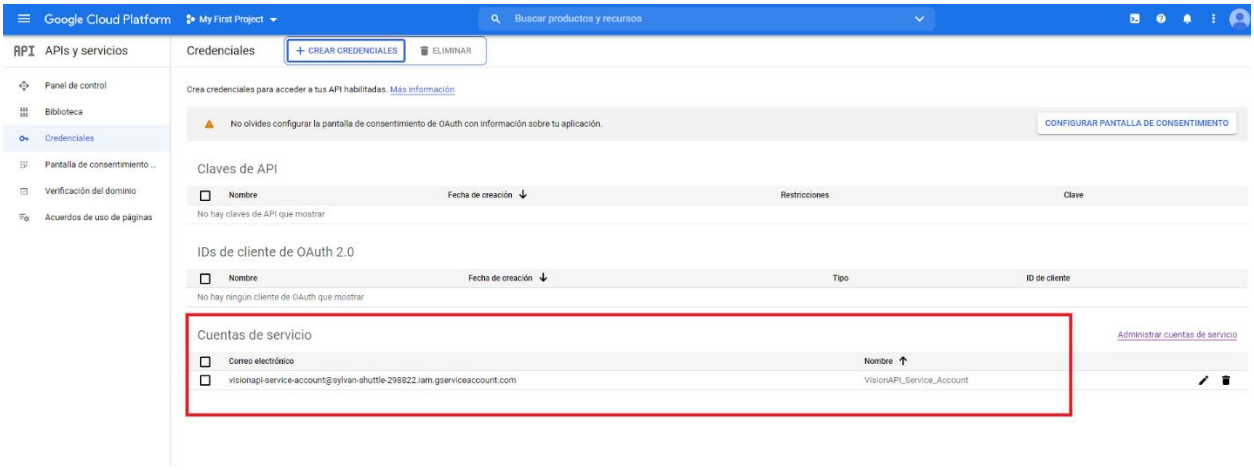

(b)

### *3.2.2 Cloud Vision API*

La segunda forma consiste en el reconocimiento por medio de la API de Google. Cloud Vision es una plataforma en la nube proporcionada por Google que permite integrar con facilidad aplicaciones de detección de objetos y reconocimiento de caracteres (Google, 2020). Para usar este servicio se hizo un registro en la plataforma indicada, y en este se tuvo que especificar cuál era la API a usar. Esto es debido a que cada aplicación que maneja Google Cloud tiene credenciales de acceso específicas. En el recuadro rojo de la Figura15 se pueden ver el registro para usar la API y en esta misma sección es donde se descarga la llave de enlace, que es un archivo .json.

**Figura 20.**

*Generación de credenciales en la Plataforma Google Cloud.*




Dicha llave de acceso es necesaria para la comunicación de extremo a extremo entre la API y el dispositivo que se manejó, en nuestro caso Raspberry Pi 3B+. Al tener las credenciales se descargó al dispositivo la librería que soporta la conexión y los comandos de trabajo específicos para la API. Posteriormente se hizo uso de la segmentación mostrada en la Figura 11(c). Con la placa vehicular ya segmentada se aplicó un cambio de tamaño para que la lectura fuera más favorable. La respuesta que genera la API es también un archivo .json, que contiene todos los caracteres que ha encontrado y sus posiciones en la imagen. Para este trabajo se hizo uso de la librería Pandas para navegar y extraer únicamente los caracteres que se necesitaban, ver Apéndice 4. En la Figura 16 podemos observar el procedimiento del proceso.

**Figura 21.**

*(a) Identificación de la placa vehicular. (b) Segmentación de la placa. (c) Respuesta de caracteres por parte de Cloud Vision. (d) Respuesta final después de procesamiento mediante Pandas.*


(a)


(b)

```
In [7]: df['description']
Out[7]:
0    DNM 627\nBOGOTA D.C.\n
1                       DNM
2                       627
3                    BOGOTA
4                      D.C.
Name: description, dtype: object
```

(c)

```
La placa es:  DNM627
```

(d)

# 4. Pruebas y resultados

Para validar y medir el nivel de rendimiento del prototipo implementado, se realizaron pruebas a una muestra de 30 vehículos distintos a los utilizados en el entrenamiento y validación de sistema. De ellos se toman 3 imágenes por vehículo: frontal, trasera y diagonal (ángulo horizontal).

Lo anterior se realizó con el objetivo de medir el nivel de detección y lectura de las placas. Dado que para las diferentes pruebas no se tiene control sobre los parámetros de captura de las imágenes con las que se realizaron, los resultados se presentan en dos diferentes categorías en las que se midieron niveles de confianza y de aciertos, tanto para el reconocimiento como la lectura de la placa.

Para la lectura de caracteres en la placa se compararon las dos formas de reconocimiento ya descritas (segmentación /Pytesseract y Cloud Vision API). Se verificó el número de aciertos, es decir, si los caracteres se encuentran en la posición indicada.

**Reconocimiento de la placa**:

Se realizaron las siguientes pruebas:

- Reconocimiento de placa frontal
- Reconocimiento de placa trasera.
- Reconocimiento de placa diagonal.

**Lectura de la placa**:

Se realizaron las siguientes pruebas:

- Lectura de placa.

## 4.1 Reconocimiento de la placa

Para la primera prueba se tomaron el grupo de imágenes que contenían vehículos con la placa frontal (Figura 22), placa trasera (Figura 24) y placa diagonal (Figura 26). Se obtuvieron los siguientes resultados:

**Figura 22.**

*(a) Vehículo con placa frontal.*


**Tabla 2.**

*Número de placas delanteras detectadas.*

| Porcentaje de placas detectadas | | | | |
|---|---|---|---|---|
| | Detectadas | No detectadas | Totales | Porcentaje de acierto |
| Delanteras | 27 | 3 | 30 | 90% |

**Tabla 3.**

*Nivel de confianza de placas delanteras detectadas.*

| Nivel de confianza placas delanteras | |
|---|---|
| Placas | Delantera (%) |
| 30 | 79,16 |

**Figura 23.**

*Nivel de confianza para placa frontal.*

**Figura 24.**

*Vehículo con placa trasera.*


**Tabla 4.**

*Número de placas traseras detectadas.*

| Porcentaje de placas detectadas | | | | |
|---|---|---|---|---|
| | Detectadas | No detectadas | Totales | Porcentaje de acierto |
| Traseras | 27 | 3 | 30 | 90% |

**Tabla 5.**

*Nivel de confianza de placas traseras detectadas.*

| Nivel de confianza placas traseras | |
|---|---|
| Placas | Trasera (%) |
| 30 | 79,16 |

**Figura 25.**

*Nivel de confianza para placa trasera.*


**Figura 26.**

*Vehículo con placa diagonal.*

**Tabla 6.**

*Número de placas diagonales detectadas.*

| Porcentaje de placas detectadas | | | | |
|---|---|---|---|---|
| | Detectadas | No detectadas | Totales | Porcentaje de acierto |
| Diagonal | 19 | 11 | 30 | 63.33% |

**Tabla 7.**

*Nivel de confianza de placas detectadas.*

| Nivel de confianza placas diagonales | |
|---|---|
| Placas | Diagonales (%) |
| 30 | 55.23 |

**Figura 27.**

*Nivel de confianza para placa diagonal.*

## 4.2 Lectura de la placa

Una vez detectada la placa se realizó la respectiva lectura de esta a partir de las placas detectadas. Por lo tanto, se modificó el número de la muestra, ya que solo se podría realizar la lectura de la placa en cuyas imágenes se detectó alguna. Se dividieron los resultados según la cantidad de caracteres obtenidos, y si estos se encontraban en la posición correcta. Se obtuvieron los siguientes datos:

**Tabla 8.**

*Lectura de placas con Pytesseract.*

| **Segmentación & Pytesseract (S/P)** | | | |
|---|---|---|---|
| **Porcentaje de aciertos** | **Placas delanteras** | **Placas traseras** | **Placas diagonales** |
| 100% | 8 | 8 | 0 |
| 83.33% | 11 | 10 | 2 |
| 66.66% | 6 | 1 | 2 |
| 50% | 1 | 2 | 3 |
| 33.33% | 1 | 0 | 5 |
| 16.66% | 0 | 0 | 0 |
| 0% | 0 | 6 | 7 |
| Totales | 27 | 27 | 19 |

Con base en lo anterior, para la lectura de placas con el sistema desarrollado, de las 30 placas por cada grupo se detectaron: 27 delanteras, 27 traseras y 19 diagonales. Ahora, tomando estos datos como el 100% de la muestra, para las 27 placas delanteras, se reconocieron completamente los caracteres de 8 (ejemplo, Figura 28(a)). De la misma muestra 11 placas

presentaron 5 de 6 caracteres (ejemplo, Figura 28(b)), los cuales se encontraban en el orden y posición indicada. Por tanto, tomando como mínimo 5 caracteres reconocidos, el modelo tiene un 70.37% de precisión para placas delanteras.

**Figura 28.**

*(a) Lectura de la placa 100%. (b) Lectura de la placa 83.33%.*


(a) (b)

Ahora bien, siguiendo con el análisis anterior para las placas traseras se obtuvo la lectura completa en 8 placas y lectura de 5 caracteres para 11 placas, presentando así una precisión de 66.66%. Por último, para placas en diagonal y siguiendo la estructura de análisis mencionada se obtuvo una precisión de 10.52%.

Se realizó el mismo proceso con Cloud Vision obteniendo los siguientes resultados:

**Tabla 9.**

*Lectura de placas con Cloud Vision.*

| Cloud Vision (CV) | | | |
|---|---|---|---|
| **Porcentaje de aciertos** | **Placas delanteras** | **Placas traseras** | **Placas diagonales** |
| 100% | 26 | 25 | 14 |

| Cloud Vision (CV) | | | |
|---|---|---|---|
| **Porcentaje de aciertos** | **Placas delanteras** | **Placas traseras** | **Placas diagonales** |
| 83.33% | 1 | 2 | 2 |
| 66.66% | 0 | 0 | 2 |
| 50% | 0 | 0 | 1 |
| 33.33% | 0 | 0 | 0 |
| 16.66% | 0 | 0 | 0 |
| 0% | 0 | 0 | 0 |
| Totales | 27 | 27 | 19 |

En este caso obtenemos una precisión del 99% en la lectura de placas delanteras, 99% en la lectura de placas traseras y 84.21% en la lectura de placas en diagonal.

## 5. Conclusiones

- Las bases de datos en el entrenamiento de redes neuronales son un punto de partida muy importante. Dependiendo de la cantidad y forma de obtención de estas los resultados pueden variar enormemente para el punto deseado. A pesar de que existan bancos de imágenes dedicados a la creación de estas, es fundamental entender que los datos tendrán más relevancia cuando estos pertenecen a una misma distribución y toman condiciones específicas para lo que se desea realizar.
- Integrar una red neuronal a un sistema embebido es un proceso un poco riguroso, ya que tanto la arquitectura del modelo como el peso que este genere deben ser tomados en cuenta para un desarrollo óptimo. Como se evidenció, el microcontrolador Maix Go tiene una enorme

ventaja en el desarrollo de este tipo de procesos debido a su procesador neuronal dedicado, por lo que para aplicaciones *End to End* este hardware es indicado. No obstante, como se constató en el desarrollo del presente proyecto, este microcontrolador aún está en una etapa temprana y tiene limitaciones de documentación.

- Debido a funciones de programación se hizo necesario usar el microcontrolador Raspberry Pi 3B+, ya que tiene un sistema más cómodo para trabajar en el procesamiento de imágenes. Aunque este no tiene el poder que la Maix Go ofrece, genera un entorno más completo para diversas aplicaciones. Lo anterior no deja de lado que el microcontrolador creado por Sipeed sea más asequible y con el paso del tiempo probablemente tendrá más relevancia en aplicaciones con IA.

- Puesto que los resultados para la detección de la placa presentan un alto valor de confianza en cuanto a la identificación de esta, verificamos la efectividad del modelo implementado, aun cuando los parámetros de la muestra no son controlados, este responde de manera satisfactoria.

- En base a los porcentajes obtenidos al realizar las pruebas, podemos validar la eficacia del sistema desarrollado. Dado que, para el resultado final, este se encuentra entre el 60 y 80 porciento de precisión, sin controlarse propiedades como la luz, distancia y ángulo de captura.

# 6. Recomendaciones

- Cloud Vision es una opción bastante fiable y recomendada para generar el reconocimiento de caracteres en este proyecto. Sin embargo, como la finalidad es dejar un sistema completo y no solo este prototipo, se propone la creación de una red neuronal de clasificación de imágenes. Para esto se hace necesario crear una base de datos de caracteres propios de las placas vehiculares colombianas y hacerles un procesamiento a las mismas.
- Adicionalmente, se recomienda incrementar el número de imágenes en la base de datos, para reiniciar o continuar el modelo ya presentado, esto con la finalidad de adquirir mayor precisión en la detección de la placa.
- Se recomienda enormemente estar pendiente de las futuras actualizaciones del microcontrolador Maix Go, ya que este es un candidato muy fuerte para generar no solo un prototipo, sino un producto completamente terminado de reconocimiento debido a su poder de cómputo.

## Referencias Bibliográficas

AIWintermuteAI. (2020, Diciembre 30). *AIWintermuteAI/aXeleRate*. GitHub. ConsultadoSeptiembre 8, 2020 de https://github.com/AIWintermuteAI/aXeleRate

ALCALDÍA MAYOR DE BOGOTA DC. (2012, Enero 10). *Decreto 019 de 2012 Nivel Nacional*. Alcaldía de Bogotá. Consultado Diciembre 22, 2020, de https://www.alcaldiabogota.gov.co/sisjur/normas/Norma1.jsp?i=45322

Bisong, E. (2019). Google Colaboratory. *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, 59–64. https://doi.org/10.1007/978-1-4842-4470-8_7

Corte Constitucional. (2011, Mayo 26). *Sentencia T-451/11*. Corte Constitucional. https://www.corteconstitucional.gov.co/relatoria/2011/T-451-11.htm

Elisha, O. (2020, Septiembre 17). *Real-time Object Detection using SSD MobileNet V2 on Video Streams*. Medium; Heartbeat. https://heartbeat.fritz.ai/real-time-object-detection-using-ssd-mobilenet-v2-on-video-streams-3bfc1577399c

Fiscalía General de la Nación. (n.d.). *FICHA TÉCNICA MT 001 PLACA ÚNICA NACIONAL*. Fiscalía General de La Nación. Consultado Diciembre 20, 2020, de https://www.fiscalia.gov.co/colombia//wp-content/uploads/policiajudicial/DOCPJFISCALIA/PLACA%20ANICA%20NACIONAL.pdf

Google Cloud. (2020, Diciembre 16). *GPUs on Compute Engine | Compute Engine Documentation*. Google Cloud. https://cloud.google.com/compute/docs/gpus

Google. (2020). *Documentación de Cloud Vision | API de Cloud Vision | Google Cloud*. Google Cloud. https://cloud.google.com/vision/docs?hl=es-419#use-cases

Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., & Andreetto, M. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. https://arxiv.org/pdf/1704.04861.pdf

Klippel, E., Oliveira, R., Maslov, D., Bianchi, A., Silva, S. E., & Garrocho, C. (2020). Towards to an Embedded Edge AI Implementation for Longitudinal Rip Detection in Conveyor Belt. *Anais Estendidos Do Simpósio Brasileiro de Engenharia de Sistemas Computacionais (SBESC Estendido 2020)*. https://doi.org/10.5753/sbesc_estendido.2020.13096

MathWorks. (n.d.). *Deep Learning: Tres cosas que es necesario saber*. La.Mathworks.com. Consultado Diciembre 16, 2020, de https://la.mathworks.com/discovery/deep-learning.html

Microsoft. (n.d.). *¿Qué es la informática en la nube? Guía para principiantes | Microsoft Azure*. Microsof Azure. Consultado Enero 7, 2021, de https://azure.microsoft.com/es-es/overview/what-is-cloud-computing/.

Nicholls, J. (2018, Agosto 13). *Quantization in Deep Learning*. Medium; Medium. https://medium.com/@joel_34050/quantization-in-deep-learning-478417eab72b#:~:text=Quantization%20for%20deep%20learning%20is,cost%20of%20using%20neural%20networks.

OpenCV. (2020a). *Image Thresholding*. Opencv.org. https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html

OpenCV. (2020b). *Morphological Transformations*. Opencv.org. https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html

Python. (n.d.). *Pytesseract*. PyPI. Consultado Diciembre15, 2020, de https://pypi.org/project/pytesseract/

Renu Khandelwal. (2019, Diciembre 7). *COCO and Pascal VOC data format for Object detection*. Medium; Towards Data Science. https://towardsdatascience.com/coco-data-format-for-object-detection-a4c5eaf518c5

Secretaría General del Senado. (2020, Diciembre 31). *Ley 769 de 2020*. Secretaría General Del Senado. http://www.secretariasenado.gov.co/senado/basedoc/ley_0769_2002.html

Soto Orozco, O., Corral Sáenz, A. D., RojoGonzález, C., & Ramírez Quintana, J. A. (2019). *ANÁLISIS DEL DESEMPEÑO DE REDES NEURONALES PROFUNDAS PARA SEGMENTACIÓN SEMÁNTICA EN HARDWARE LIMITADO*. http://recibe.cucei.udg.mx/revista/es/vol8-no2/computacion06.pdf

TensorFlow. (2019). *TensorFlow*. TensorFlow. https://www.tensorflow.org/

Tensorflow. (2020, Julio 28). *tensorflow/models*. GitHub. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md

tzutalin. (n.d.). *tzutalin/labelImg*. GitHub. Consultado Octubre 10, 2020, de https://github.com/tzutalin/labelImg

Universidad Industrial de Santander. (2014). *Deep Learning*. Biblioteca BASES DE DATOS. Consultado Diciembre 27, 2020, de https://bibliotecavirtual.uis.edu.co/login?qurl=http://eds.a.ebscohost.com%2feds%2fdetail%2fdetail%3fvid%3d1%26sid%3de4c8864c-7b4f-43ce-8468-b3162cb38eb4%2540sdc-v-sessmgr03%26bdata%3dJmxhbmc9ZXMmc2l0ZT1lZHMtbGl2ZQ%253d%253d#AN=119214353&db=ers

# Apéndices

## Apéndice A. Entrenamiento de detección para placas vehiculares.

El siguiente Colab está enfocado para entrenar arquitecturas de redes neuronales convolucionales enfocadas a la detección de objetos.

**Importar librerias y seleccionar versión de TF.**

```
from google.colab import drive

drive.mount('/gdrive')
# the project's folder
%cd /gdrive/'My Drive'/Object_Detection
```

```
Mounted at /gdrive
/gdrive/My Drive/Object_Detection
```

```
%tensorflow_version 1.x
```

```
TensorFlow 1.x selected.
```

```
import tensorflow
print(tensorflow.__version__)
```

```
1.15.2
```

```
!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk
!pip install -qq Cython contextlib2 pillow lxml matplotlib pycocotools

from __future__ import division, print_function, absolute_import

import pandas as pd
import numpy as np
import csv

import re
import os
import io
import glob
import shutil
import urllib.request
import tarfile
import xml.etree.ElementTree as ET
```

```
import tensorflow.compat.v1 as tf
import cv2

from PIL import Image
from collections import namedtuple, OrderedDict

from google.colab import files

!ls

data  models
```

**Conversión de archivos XML a CSV**

```
#adjusted from: https://github.com/datitran/raccoon_dataset
def xml_to_csv(path):
  classes_names = []
  xml_list = []

  for xml_file in glob.glob(path + '/*.xml'):
    tree = ET.parse(xml_file)
    root = tree.getroot()
    for member in root.findall('object'):
      classes_names.append(member[0].text)
      value = (root.find('filename').text ,
               int(root.find('size')[0].text),
               int(root.find('size')[1].text),
               member[0].text,
               int(member[4][0].text),
               int(member[4][1].text),
               int(member[4][2].text),
               int(member[4][3].text))
      xml_list.append(value)
  column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xma
x', 'ymax']
  xml_df = pd.DataFrame(xml_list, columns=column_name)
  classes_names = list(set(classes_names))
  classes_names.sort()
  return xml_df, classes_names

for label_path in ['train_labels', 'test_labels']:
  image_path = os.path.join(os.getcwd(), label_path)
  xml_df, classes = xml_to_csv(label_path)
  xml_df.to_csv(f'{label_path}.csv', index=None)
  print(f'Successfully converted {label_path} xml to csv.')

label_map_path = os.path.join("label_map.pbtxt")
pbtxt_content = ""

for i, class_name in enumerate(classes):
    pbtxt_content = (
```

```
        pbtxt_content
        + "item {{\n    id: {0}\n    name: '{1}'\n}}\n\n".format(i + 1, class
_name)
    )
pbtxt_content = pbtxt_content.strip()
with open(label_map_path, "w") as f:
    f.write(pbtxt_content)
```

```
Successfully converted train_labels xml to csv.
Successfully converted test_labels xml to csv.
```

```
%cd ..
```

```
/gdrive/My Drive/Object_Detection
```

**Importar TensorFlow y gestionar la ruta (PATH)**

```
# downloads the models
!git clone --q https://github.com/tensorflow/models.git
```

```
%cd models/research/
```

```
/gdrive/My Drive/Object_Detection/models/research
```

```
!pip install tf_slim
```

```
Requirement already satisfied: tf_slim in /usr/local/lib/python3.6/dist-packa
ges (1.1.0)
Requirement already satisfied: absl-py>=0.2.2 in /usr/local/lib/python3.6/dis
t-packages (from tf_slim) (0.10.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
(from absl-py>=0.2.2->tf_slim) (1.15.0)
```

```
# compils the proto buffers
!protoc object_detection/protos/*.proto --python_out=.
# exports PYTHONPATH environment var with research and slim paths
os.environ['PYTHONPATH'] += ':./:./slim/'
```

**Se comprueba que los modelos se ejecuten de manera correcta.**

```
# testing the model builder
!python3 object_detection/builders/model_builder_test.py
```

**Conversión de archivos CSV a TFRecord**

```
#adjusted from: https://github.com/datitran/raccoon_dataset
from object_detection.utils import dataset_util


#change this to the base directory where your data/ is
data_base_url = '/gdrive/My Drive/Object_Detection/data/'

#location of images
image_dir = data_base_url +'images/'
```

```
def class_text_to_int(row_label):
  if row_label == 'Placa':
    return 1
  else:
    None

def split(df, group):
  data = namedtuple('data', ['filename', 'object'])
  gb = df.groupby(group)
  return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.ke
ys(), gb.groups)]

def create_tf_example(group, path):
    with tf.io.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), '
rb') as fid:
      encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size
    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_te
xt),
```

```
        'image/object/class/label': dataset_util.int64_list_feature(classes),
        }))
    return tf_example
#creates tfrecord for both csv's
for csv in ['train_labels', 'test_labels']:
  writer = tf.io.TFRecordWriter(data_base_url + csv + '.record')
  path = os.path.join(image_dir)
  examples = pd.read_csv(data_base_url + csv + '.csv')
  grouped = split(examples, 'filename')
  for group in grouped:
    tf_example = create_tf_example(group, path)
    writer.write(tf_example.SerializeToString())

  writer.close()
  output_path = os.path.join(os.getcwd(), data_base_url + csv + '.record')
  print('Successfully created the TFRecords: {}'.format(data_base_url +csv +
'.record'))
```

```
Successfully created the TFRecords: /gdrive/My Drive/Object_Detection/data/tr
ain_labels.record
Successfully created the TFRecords: /gdrive/My Drive/Object_Detection/data/te
st_labels.record
```

**Selección de arquitectura a usar en el modelo**

Estos modelos están dados por TensorFlow. Para más información, visitar https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md

```
MODELS_CONFIG = {
    'ssd_mobilenet_v1_quantized': {
        'model_name': 'ssd_mobilenet_v1_quantized_300x300_coco14_sync_2018_07
_18',
    },
    'faster_rcnn_inception_v2': {
        'model_name': 'faster_rcnn_inception_v2_coco_2018_01_28',
    },
}

# Select a model from `MODELS_CONFIG`.
# I chose ssd_mobilenet_v2 for this project, you could choose any
selected_model = 'ssd_mobilenet_v1_quantized'

#the distination folder where the model will be saved
#change this if you have a different working dir
DEST_DIR = '/gdrive/My Drive/Object_Detection/models/research/pretrained_mode
l'

# Name of the object detection model to use.
MODEL = MODELS_CONFIG[selected_model]['model_name']
```

```
#selecting the model
MODEL_FILE = MODEL + '.tar.gz'

#creating the downlaod link for the model selected
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'

#checks if the model has already been downloaded, download it otherwise
if not (os.path.exists(MODEL_FILE)):
    urllib.request.urlretrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)

#unzipping the model and extracting its content
tar = tarfile.open(MODEL_FILE)
tar.extractall()
tar.close()

# creating an output file to save the model while training
os.remove(MODEL_FILE)
if (os.path.exists(DEST_DIR)):
    shutil.rmtree(DEST_DIR)
os.rename(MODEL, DEST_DIR)
```

```
!ls
```

```
a3c_blogpost        delf              pcl_rl
adversarial_text    efficient-hrl         pretrained_model
attention_ocr       lfads             README.md
audioset        lstm_object_detection     rebar
autoaugment     marco             seq_flow_lite
cognitive_planning  ngrok             setup.py
cvt_text        ngrok-stable-linux-amd64.zip  slim
deeplab         nst_blogpost          training
deep_speech     object_detection          vid2depth
```

```
!mkdir training
```

**Selección y configuración del archivo .config**

```
#path to the config file
!cat object_detection/samples/configs/ssd_mobilenet_v1_quantized_300x300_coco
14_sync.config
```

```
#path to the config file
%%writefile object_detection/samples/configs/ssd_mobilenet_v1_quantized_300x3
00_coco14_sync_2018_07_18.config
# SSD with Mobilenet v1 with quantized training.
# Trained on COCO, initialized from Imagenet classification checkpoint

# Achieves 18.2 mAP on coco14 minival dataset.

# This config is TPU compatible
```

```
model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 1
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    encode_background_as_zeros: true
    anchor_generator {
      ssd_anchor_generator {
        num_layers: 6
        min_scale: 0.2
        max_scale: 0.95
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        aspect_ratios: 3.0
        aspect_ratios: 0.3333
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
    box_predictor {
      convolutional_box_predictor {
        min_depth: 0
        max_depth: 0
```

```
    num_layers_before_predictor: 0
    use_dropout: false
    dropout_keep_probability: 0.8
    kernel_size: 1
    box_code_size: 4
    apply_sigmoid_to_scores: false
    class_prediction_bias_init: -4.6
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        random_normal_initializer {
          stddev: 0.01
          mean: 0.0
        }
      }
      batch_norm {
        scale: true,
        center: true,
        decay: 0.97,
        epsilon: 0.001,
      }
    }
  }
}
feature_extractor {
  type: 'ssd_mobilenet_v1'
  min_depth: 16
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
      l2_regularizer {
        weight: 0.00004
      }
    }
    initializer {
      random_normal_initializer {
        stddev: 0.01
        mean: 0.0
      }
    }
    batch_norm {
      scale: true,
      center: true,
      decay: 0.97,
```

```
          epsilon: 0.001,
        }
      }
      override_base_feature_extractor_hyperparams: true
    }
    loss {
      classification_loss {
        weighted_sigmoid_focal {
          alpha: 0.75,
          gamma: 2.0
        }
      }
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    normalize_loc_loss_by_codesize: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
  fine_tune_checkpoint: "/gdrive/My Drive/Object_Detection/models/research/pr
etrained_model/model.ckpt"
  batch_size: 6
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
  optimizer {
```

```
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: 0.004
          total_steps: 200000
          warmup_learning_rate: 0.001
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
}
train_input_reader: {
  tf_record_input_reader {
    input_path: "/gdrive/My Drive/Object_Detection/data/train_labels.record"
  }
  label_map_path: "/gdrive/My Drive/Object_Detection/data/label_map.pbtxt"
}

eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  num_examples: 8000
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "/gdrive/My Drive/Object_Detection/data/test_labels.record"
  }
  label_map_path: "/gdrive/My Drive/Object_Detection/data/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}

graph_rewriter {
  quantization {
    delay: 48000
    activation_bits: 8
    weight_bits: 8
  }
}
```

```
Writing object_detection/samples/configs/ssd_mobilenet_v1_quantized_300x300_c
oco14_sync_2018_07_18.config
```

**Importar y generar espacio de visualización en TensorBoard**

```
!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip -o ngrok-stable-linux-amd64.zip
```

```
--2021-01-05 00:12:30--  https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-li
nux-amd64.zip
Resolving bin.equinox.io (bin.equinox.io)... 34.193.233.154, 34.205.198.58, 3
4.232.108.170, ...
Connecting to bin.equinox.io (bin.equinox.io)|34.193.233.154|:443... connecte
d.
HTTP request sent, awaiting response... 200 OK
Length: 13773305 (13M) [application/octet-stream]
Saving to: ‘ngrok-stable-linux-amd64.zip.7’

ngrok-stable-linux- 100%[===================>]  13.13M  53.6MB/s    in 0.2s

2021-01-05 00:12:30 (53.6 MB/s) - ‘ngrok-stable-linux-amd64.zip.7’ saved [137
73305/13773305]

Archive:  ngrok-stable-linux-amd64.zip
  inflating: ngrok
```

```
#the logs that are created while training
LOG_DIR = "training/"
get_ipython().system_raw(
    'tensorboard --logdir {} --host 0.0.0.0 --port 6006 &'
    .format(LOG_DIR)
)
get_ipython().system_raw('./ngrok http 6006 &')
#The link to tensorboard.
#works after the training starts.
!curl -s http://localhost:4040/api/tunnels | python3 -c \
    "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url']
)"
```

```
https://750b12644e9e.ngrok.io
```

**Entrenamiento y congelamiento de modelos**

**Entrenar el modelo**

```
!python3 object_detection/model_main.py \
    --pipeline_config_path=/gdrive/My\ Drive/Object_Detection/models/research
/object_detection/samples/configs/ssd_mobilenet_v1_quantized_300x300_coco14_s
ync.config \
    --model_dir=training
```

```
!pip install lvis
```

```
Collecting lvis
  Downloading https://files.pythonhosted.org/packages/72/b6/1992240ab48310b53
60bfdd1d53163f43bb97d90dc5dc723c67d41c38e78/lvis-0.5.3-py3-none-any.whl
Requirement already satisfied: cycler>=0.10.0 in /usr/local/lib/python3.6/dis
```

```
t-packages (from lvis) (0.10.0)
Requirement already satisfied: numpy>=1.18.2 in /usr/local/lib/python3.6/dist
-packages (from lvis) (1.19.4)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.6/dist-p
ackages (from lvis) (1.15.0)
Requirement already satisfied: kiwisolver>=1.1.0 in /usr/local/lib/python3.6/
dist-packages (from lvis) (1.3.1)
Requirement already satisfied: pyparsing>=2.4.0 in /usr/local/lib/python3.6/d
ist-packages (from lvis) (2.4.7)
Requirement already satisfied: Cython>=0.29.12 in /usr/local/lib/python3.6/di
st-packages (from lvis) (0.29.21)
Requirement already satisfied: matplotlib>=3.1.1 in /usr/local/lib/python3.6/
dist-packages (from lvis) (3.2.2)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/pytho
n3.6/dist-packages (from lvis) (2.8.1)
Requirement already satisfied: opencv-python>=4.1.0.25 in /usr/local/lib/pyth
on3.6/dist-packages (from lvis) (4.1.2.30)
Installing collected packages: lvis
Successfully installed lvis-0.5.3
```

**Escoger el último modelo entrenado**

```
# by Chengwei
#dir where the model will be saved
output_directory = './TFLite_model'

lst = os.listdir('training')
lst = [l for l in lst if 'model.ckpt-' in l and '.meta' in l]
steps=np.array([int(re.findall('\d+', l)[0]) for l in lst])
last_model = lst[steps.argmax()].replace('.meta', '')

last_model_path = os.path.join('training', last_model)

last_model
```

```
'model.ckpt-111310'
```

**Congelar el modelo para TF**

```
!python /gdrive/'My Drive'/Object_Detection/models/research/object_detection/
export_inference_graph.py \
    --input_type=image_tensor \
    --pipeline_config_path=/gdrive/My\ Drive/Object_Detection/models/research
/object_detection/samples/configs/ssd_mobilenet_v1_quantized_300x300_coco14_s
ync.config \
    --output_directory={output_directory} \
    --trained_checkpoint_prefix={last_model_path}
```

**Congelar el modelo para TFLite**

```
!python /gdrive/'My Drive'/Object_Detection/models/research/object_detection/
export_tflite_ssd_graph.py \
    --pipeline_config_path=/gdrive/My\ Drive/Object_Detection/models/research
/object_detection/samples/configs/ssd_mobilenet_v1_quantized_300x300_coco14_s
ync.config \
    --output_directory={output_directory} \
    --trained_checkpoint_prefix={last_model_path} \
    --add_postprocessing_op=true
```

## Apéndice B. Reconocimiento de placa en video en vivo

### *Video para Raspberry Pi 3B+*

```
import sensor,image,lcd
import KPU as kpu

lcd.init()
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.set_windowing((224, 224))
sensor.set_vflip(1)

sensor.run(1)
classes = ["Placa"]
task = kpu.load(0x200000) #change to "/sd/name_of_the_model_file.kmodel" if loading fro
m SD card
a = kpu.set_outputs(task, 0, 7,7,30)   #the actual shape needs to match the last layer
shape of your model(before Reshape)
anchor = (0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77
052, 9.16828)
a = kpu.init_yolo2(task, 0.3, 0.3, 5, anchor) #tweak the second parameter if you're get
ting too many false positives
clock=time.clock()
while(True):
    clock.tick()
    img = sensor.snapshot()
    code = kpu.run_yolo2(task, img)
    if code:
        fps = clock.fps()
        print(fps)
        img.draw_string(2,2, ( "FPS: %2.1f" % (fps)), color=(0,128,0),scale=2)
        for i in code:
            a=img.draw_rectangle(i.rect(),color = (0, 255, 0))
            a = img.draw_string(i.x(),i.y(), classes[i.classid()], color=(255,0,0), sca
le=3)
        a = lcd.display(img)
    else:
        a = lcd.display(img)
a = kpu.deinit(task)
```

### *Video para Raspberry Pi 3B+*

```
######## Webcam Object Detection Using Tensorflow-trained Classifier #########
#
# Author: Evan Juras
# Date: 10/27/19
# Description:
# This program uses a TensorFlow Lite model to perform object detection on a live webca
m
# feed. It draws boxes and scores around the objects of interest in each frame from the
```

```
# webcam. To improve FPS, the webcam object runs in a separate thread from the main program.
# This script will work with either a Picamera or regular USB webcam.
#
# This code is based off the TensorFlow Lite image classification example at:
# https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/examples/python/label_image.py
#
# I added my own method of drawing boxes and labels using OpenCV.

# Import packages
import os
import argparse
import cv2
import numpy as np
import sys
import time
from threading import Thread
import importlib.util

# Define VideoStream class to handle streaming of video from webcam in separate processing thread
# Source - Adrian Rosebrock, PyImageSearch: https://www.pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/
class VideoStream:
    """Camera object that controls video streaming from the Picamera"""
    def __init__(self,resolution=(640,480),framerate=30):
        # Initialize the PiCamera and the camera image stream
        self.stream = cv2.VideoCapture(0)
        ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
        ret = self.stream.set(3,resolution[0])
        ret = self.stream.set(4,resolution[1])

        # Read first frame from the stream
        (self.grabbed, self.frame) = self.stream.read()

    # Variable to control when the camera is stopped
        self.stopped = False

    def start(self):
    # Start the thread that reads frames from the video stream
        Thread(target=self.update,args=()).start()
        return self

    def update(self):
        # Keep looping indefinitely until the thread is stopped
        while True:
            # If the camera is stopped, stop the thread
            if self.stopped:
                # Close camera resources
                self.stream.release()
                return

            # Otherwise, grab the next frame from the stream
            (self.grabbed, self.frame) = self.stream.read()

    def read(self):
    # Return the most recent frame
        return self.frame

    def stop(self):
```

```
    # Indicate that the camera and thread should be stopped
        self.stopped = True

# Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                    default='TFLite_model')
parser.add_argument('--graph', help='Name of the .tflite file, if different than detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different than labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for displaying detected objects',
                    default=0.5)
parser.add_argument('--resolution', help='Desired webcam resolution in WxH. If the webcam does not support the resolution entered, errors may occur.',
                    default='1280x720')
parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up detection',
                    action='store_true')

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
min_conf_threshold = float(args.threshold)
resW, resH = args.resolution.split('x')
imW, imH = int(resW), int(resH)
use_TPU = args.edgetpu

# Import TensorFlow libraries
# If tflite_runtime is installed, import interpreter from tflite_runtime, else import from regular tensorflow
# If using Coral Edge TPU, import the load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
            from tflite_runtime.interpreter import load_delegate
    else:
        from tensorflow.lite.python.interpreter import Interpreter
        if use_TPU:
            from tensorflow.lite.python.interpreter import load_delegate

    # If using Edge TPU, assign filename for Edge TPU model
    if use_TPU:
        # If user has specified the name of the .tflite file, use that name, otherwise use default 'edgetpu.tflite'
        if (GRAPH_NAME == 'detect.tflite'):
            GRAPH_NAME = 'edgetpu.tflite'

    # Get path to current working directory
    CWD_PATH = os.getcwd()

    # Path to .tflite file, which contains the model that is used for object detection
    PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)
```

```
117.
118.       # Path to label map file
119.       PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)
120.
121.       # Load the label map
122.       with open(PATH_TO_LABELS, 'r') as f:
123.           labels = [line.strip() for line in f.readlines()]
124.
125.       # Have to do a weird fix for label map if using the COCO "starter model" from
126.       # https://www.tensorflow.org/lite/models/object_detection/overview
127.       # First label is '???', which has to be removed.
128.       if labels[0] == '???':
129.           del(labels[0])
130.
131.       # Load the Tensorflow Lite model.
132.       # If using Edge TPU, use special load_delegate argument
133.       if use_TPU:
134.           interpreter = Interpreter(model_path=PATH_TO_CKPT,
135.                                     experimental_delegates=[load_delegate('libedgetpu.
    so.1.0')])
136.           print(PATH_TO_CKPT)
137.       else:
138.           interpreter = Interpreter(model_path=PATH_TO_CKPT)
139.
140.       interpreter.allocate_tensors()
141.
142.       # Get model details
143.       input_details = interpreter.get_input_details()
144.       output_details = interpreter.get_output_details()
145.       height = input_details[0]['shape'][1]
146.       width = input_details[0]['shape'][2]
147.
148.       floating_model = (input_details[0]['dtype'] == np.float32)
149.
150.       input_mean = 127.5
151.       input_std = 127.5
152.
153.       # Initialize frame rate calculation
154.       frame_rate_calc = 1
155.       freq = cv2.getTickFrequency()
156.
157.       # Initialize video stream
158.       videostream = VideoStream(resolution=(imW,imH),framerate=30).start()
159.       time.sleep(1)
160.
161.       #for frame1 in camera.capture_continuous(rawCapture, format="bgr",use_video_port
    =True):
162.       while True:
163.
164.           # Start timer (for calculating frame rate)
165.           t1 = cv2.getTickCount()
166.
167.           # Grab frame from video stream
168.           frame1 = videostream.read()
169.
170.           # Acquire frame and resize to expected shape [1xHxWx3]
171.           frame = frame1.copy()
172.           frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
173.           frame_resized = cv2.resize(frame_rgb, (width, height))
174.           input_data = np.expand_dims(frame_resized, axis=0)
175.
```

```
176.           # Normalize pixel values if using a floating model (i.e. if model is non-
    quantized)
177.           if floating_model:
178.               input_data = (np.float32(input_data) - input_mean) / input_std
179.
180.           # Perform the actual detection by running the model with the image as input

181.           interpreter.set_tensor(input_details[0]['index'],input_data)
182.           interpreter.invoke()
183.
184.           # Retrieve detection results
185.           boxes = interpreter.get_tensor(output_details[0]['index'])[0] # Bounding box
     coordinates of detected objects
186.           classes = interpreter.get_tensor(output_details[1]['index'])[0] # Class inde
    x of detected objects
187.           scores = interpreter.get_tensor(output_details[2]['index'])[0] # Confidence
    of detected objects
188.           #num = interpreter.get_tensor(output_details[3]['index'])[0]  # Total number
     of detected objects (inaccurate and not needed)
189.
190.           # Loop over all detections and draw detection box if confidence is above min
    imum threshold
191.           for i in range(len(scores)):
192.               if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):
193.
194.                   # Get bounding box coordinates and draw box
195.                   # Interpreter can return coordinates that are outside of image dimen
    sions, need to force them to be within image using max() and min()
196.                   ymin = int(max(1,(boxes[i][0] * imH)))
197.                   xmin = int(max(1,(boxes[i][1] * imW)))
198.                   ymax = int(min(imH,(boxes[i][2] * imH)))
199.                   xmax = int(min(imW,(boxes[i][3] * imW)))
200.
201.                   cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)
202.
203.                   # Draw label
204.                   object_name = labels[int(classes[i])] # Look up object name from "la
    bels" array using class index
205.                   label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example: 'p
    erson: 72%'
206.                   labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLE
    X, 0.7, 2) # Get font size
207.                   label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw la
    bel too close to top of window
208.                   cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-
    10), (xmin+labelSize[0], label_ymin+baseLine-
    10), (255, 255, 255), cv2.FILLED) # Draw white box to put label text in
209.                   cv2.putText(frame, label, (xmin, label_ymin-
    7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text
210.
211.           # Draw framerate in corner of frame
212.           cv2.putText(frame,'FPS: {0:.2f}'.format(frame_rate_calc),(30,50),cv2.FONT_HE
    RSHEY_SIMPLEX,1,(255,255,0),2,cv2.LINE_AA)
213.
214.           # All the results have been drawn on the frame, so it's time to display it.

215.           cv2.imshow('Object detector', frame)
216.
217.           # Calculate framerate
218.           t2 = cv2.getTickCount()
219.           time1 = (t2-t1)/freq
```

```
            frame_rate_calc= 1/time1

            # Press 'q' to quit
            if cv2.waitKey(1) == ord('q'):
                break

        # Clean up
        cv2.destroyAllWindows()
        videostream.stop()
```

## Apéndice C. Código de detección usando TensorFlow y Pytesseract

```
######## Detector de Placas Vehiculares usando TensorFLow y una Camara Pi #########
#
# Autores : Juan Vargas & William Moncada
# Fecha: 10/08/2020
# Descripción:
#El siguiente programa usa un modelo de detección de objetos mediante TensorFLow Lite
#para reconocer una placa vehicular. Asi mismo se hace la lectura de carácteres mediant
e
#procesamiento de imagenes con segmentación y Pytesseract.

#
# Este código está basado en :
# https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-
Raspberry-Pi/blob/master/TFLite_detection_image.py
#
#Se agregaron los métodos de captura,clasificación correspondientes y OCR.

# Importar paquetes
import os
import argparse
import cv2
import numpy as np
import sys
import glob
import importlib.util
import pytesseract
import re


cam = cv2.VideoCapture(0)

cv2.namedWindow("test")

img_counter = 0

while True:
    ret, frame = cam.read()
    if not ret:
        print("Fallo al tomar la foto")
        break
    cv2.imshow("Camara", frame)

    k = cv2.waitKey(1)
    if k%256 == 27:
```

```
        # ESC presionado
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # Espacio presionado
        img_name = "Imagen{}.jpg".format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} Guardada!".format(img_name))
        img_counter += 1


cam.release()

cv2.destroyAllWindows()


# Definir y analizar argumentos de entrada Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                    default='TFLite_model')
parser.add_argument('--
graph', help='Name of the .tflite file, if different than detect.tflite',
                    default='detect.tflite')
parser.add_argument('--
labels', help='Name of the labelmap file, if different than labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--
threshold', help='Minimum confidence threshold for displaying detected objects',
                    default=0.5)
parser.add_argument('--
image', help='Name of the single image to perform detection on. To run detection on mul
tiple images, use --imagedir',
                    default='Imagen0.jpg')
parser.add_argument('--
imagedir', help='Name of the folder containing images to perform detection on. Folder m
ust contain only images.',
                    default=None)
parser.add_argument('--
edgetpu', help='Use Coral Edge TPU Accelerator to speed up detection',
                    action='store_true')

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
min_conf_threshold = float(args.threshold)
use_TPU = args.edgetpu

# Analizar el nombre y el directorio de la imagen de entrada Parse input image name and
 directory.
IM_NAME = args.image
IM_DIR = args.imagedir

# Si se especifican tanto una imagen como una carpeta, arroja un error If both an image
 AND a folder are specified, throw an error
if (IM_NAME and IM_DIR):
    print('¡Error! Utilice únicamente el argumento --image o --imagedir, no ambos')
    sys.exit()

```

```
# Si no se especifica ni una imagen ni una carpeta, por defecto se usa 'test1.jpg' para el nombre de la imagen If neither an image or a folder are specified, default to using 'test1.jpg' for image name
if (not IM_NAME and not IM_DIR):
    IM_NAME = 'test1.jpg'

# Importar TensorFlow librerias

        pkg = importlib.util.find_spec('tflite_runtime')
        if pkg:
            from tflite_runtime.interpreter import Interpreter
            if use_TPU:
                from tflite_runtime.interpreter import load_delegate
        else:
            from tensorflow.lite.python.interpreter import Interpreter
            if use_TPU:
                from tensorflow.lite.python.interpreter import load_delegate

        # Si usa Edge TPU, asigne un nombre de archivo para el modelo de Edge TPU If using Edge TPU, assign filename for Edge TPU model
        if use_TPU:
            # Si el usuario ha especificado el nombre del archivo .tflite, use ese nombre; de lo contrario, use el predeterminado 'edgetpu.tflite' If user has specified the name of the .tflite file, use that name, otherwise use default 'edgetpu.tflite'
            if (GRAPH_NAME == 'detect.tflite'):
                GRAPH_NAME = 'edgetpu.tflite'


        # Obtener la ruta al directorio de trabajo actual
        CWD_PATH = os.getcwd()

        # Defina la ruta a las imágenes y tome todos los nombres de archivo de imagen
        if IM_DIR:
            PATH_TO_IMAGES = os.path.join(CWD_PATH,IM_DIR)
            images = glob.glob(PATH_TO_IMAGES + '/*')

        elif IM_NAME:
            PATH_TO_IMAGES = os.path.join(CWD_PATH,IM_NAME)
            images = glob.glob(PATH_TO_IMAGES)

        # Ruta al archivo .tflite, que contiene el modelo que se utiliza para la detección de objetos
        PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

        # Ruta al archivo label map
        PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

        # Cargar the label map
        with open(PATH_TO_LABELS, 'r') as f:
            labels = [line.strip() for line in f.readlines()]


        if labels[0] == '???':
            del(labels[0])

        # Cargar el modelo de Tensorflow Lite.
        # Si usa Edge TPU, use el argumento especial load_delegate
        if use_TPU:
            interpreter = Interpreter(model_path=PATH_TO_CKPT,
                                      experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
```

```
148.            print(PATH_TO_CKPT)
149.        else:
150.            interpreter = Interpreter(model_path=PATH_TO_CKPT)
151.
152.        interpreter.allocate_tensors()
153.
154.        # Obtener detalles del modelo
155.        input_details = interpreter.get_input_details()
156.        output_details = interpreter.get_output_details()
157.        height = input_details[0]['shape'][1]
158.        width = input_details[0]['shape'][2]
159.
160.        floating_model = (input_details[0]['dtype'] == np.float32)
161.
162.        input_mean = 127.5
163.        input_std = 127.5
164.
165.        # Recorrer cada imagen y realice la detección
166.        for image_path in images:
167.
168.            # Cargue la imagen y cambie el tamaño a la forma esperada [1xHxWx3]
169.            image = cv2.imread(image_path)
170.            image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
171.            imH, imW, _ = image.shape
172.            image_resized = cv2.resize(image_rgb, (width, height))
173.            input_data = np.expand_dims(image_resized, axis=0)
174.
175.            # Normalice los valores de los píxeles si utiliza un modelo flotante (es dec
    ir, si el modelo no está cuantificado)
176.            if floating_model:
177.                input_data = (np.float32(input_data) - input_mean) / input_std
178.
179.            # Realice la detección real ejecutando el modelo con la imagen como entrada

180.            interpreter.set_tensor(input_details[0]['index'],input_data)
181.            interpreter.invoke()
182.
183.            # Recuperar resultados de detección
184.            boxes = interpreter.get_tensor(output_details[0]['index'])[0] # Bounding box
     coordenadas de los objetos detectados
185.            classes = interpreter.get_tensor(output_details[1]['index'])[0] # Índice de
    clase de objetos detectados
186.            scores = interpreter.get_tensor(output_details[2]['index'])[0] # Confianza d
    e los objetos detectados
187.
188.            # Recorra todas las detecciones y dibuje el cuadro de detección si la confia
    nza está por encima del umbral mínimo
189.            for i in range(len(scores)):
190.                if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):
191.
192.                    # Obtener las coordenadas del cuadro delimitador y dibujarlo
193.                    ymin = int(max(1,(boxes[i][0] * imH)))
194.                    xmin = int(max(1,(boxes[i][1] * imW)))
195.                    ymax = int(min(imH,(boxes[i][2] * imH)))
196.                    xmax = int(min(imW,(boxes[i][3] * imW)))
197.
198.
199.
200.                    cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)
201.                    crop_img = image[ymin:ymax, xmin:xmax] #Segmentar la placa
202.
```

```
                    img=crop_img


                    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Pasar a escala de gris
es
                    gray = cv2.resize(gray, None, fx = 3, fy = 3, interpolation = cv2.IN
TER_CUBIC)#Cambiar el tamaño
                    imCopy= gray.copy()#Copia de seguridad
                    blur = cv2.GaussianBlur(gray, (7,7),0) #Filtro Gausiano para el ruid
o
                    ret, thresh = cv2.threshold(blur, 180, 255, cv2.THRESH_OTSU | cv2.TH
RESH_BINARY_INV)#Binarización de la imagen


                    rect_kern = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))#Crear e
structura para la dilatación
                    dilation = cv2.dilate(thresh, rect_kern, iterations = 1)# Dilatar la
 imagen

                    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE + cv2.R
ETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)#Encontrar contornos dentro de la imagen

                    def sort_contours(cnts,reverse = False):
                        i = 0
                        boundingBoxes = [cv2.boundingRect(c) for c in cnts]
                        (cnts, boundingBoxes) = zip(*sorted(zip(cnts, boundingBoxes),
                                                            key=lambda b: b[1][i], rever
se=reverse))
                        return cnts

                    sorted_contours = sorted(contours, key=lambda ctr: cv2.boundingRect(
ctr)[0])# Ordenar contornos

                    cv2.drawContours(imCopy, contours, -1, (0,255,0))

                    crop_characters = []
                    plate_num = ""
                    for cnt in sorted_contours: #Condiciones de segmentación de de carac
teres
                        x,y,w,h = cv2.boundingRect(cnt)
                        height, width = imCopy.shape

                        if height / float(h) > 6: continue
                        ratio = h / float(w)
                        if ratio < 1.5: continue
                        if width / float(w) > 15: continue
                        area = h * w
                        if area < 100: continue
                        rect=cv2.rectangle(imCopy,(x,y),(x+w,y+h),(0,255,0),2)

                        roi = dilation[y-7:y+h+7, x-
7:x+w+7] #Recorte de imagenes sobre dilatación
                        roi = cv2.bitwise_not(roi) # Invertir colores
                        try:
                            roi = cv2.medianBlur(roi, 5)
                        except:
                            roi=roi

                        crop_characters.append(roi) #Guardar caracteres en lista
                        try:
                            #Configuración Pytesseract
```

```
                            text = pytesseract.image_to_string(roi, config='-c tessedit_char_whitelist=0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ --psm 10 --oem 3')

                            clean_text = re.sub('[\W_]+', '', text)# Limpiar el texto de Pytesseract
                            plate_num += clean_text
                        except:
                            text = None
                    if plate_num != None:
                        print("La placa es #: ", plate_num)

                    try:
                        #Generar imagenes para revisión
                        cv2.imshow("Placa",img)
                        cv2.imshow("Rects",rect)
                        cv2.imshow("Dilation",dilation)
                    except NameError:
                        print("No se reconoció ninguna placa")
                        break



                    # Dibujar etiqueta
                    object_name = labels[int(classes[i])] # Usando classes buscamos la etiqueta
                    label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Generamos la precisión de detección
                    labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Creamos un fondo
                    label_ymin = max(ymin, labelSize[1] + 10) # Aegurar no dibujar la etiqueta demasiado cerca de la parte superior de la ventana
                    cv2.rectangle(image, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Dibujar una caja blanca para poner el texto
                    cv2.putText(image, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Dibujar la etiqueta de texto



            # Todos los resultados se han dibujado en la imagen, ahora muestra la imagen.
            cv2.imshow('Object detector', image)
            try:
                cv2.imshow("Placa",crop_img)
                cv2.imwrite("Platee.jpg",crop_img)
            except NameError:
                print("No se detectó ninguna placa")
                break

            #cv2.imwrite("Platee.jpg",crop_img)

            # Presionar q para cerrar
            if cv2.waitKey(0) == ord('q'):
                break

        # Limpiar
        cv2.destroyAllWindows()
        # Import packages
        import os
        import argparse
        import cv2
```

```
304.        import numpy as np
305.        import sys
306.        import time
307.        from threading import Thread
308.        import importlib.util
309.
310.        # Define VideoStream class to handle streaming of video from webcam in separate
    processing thread
311.        # Source - Adrian Rosebrock, PyImageSearch: https://www.pyimagesearch.com/2015/1
    2/28/increasing-raspberry-pi-fps-with-python-and-opencv/
312.        class VideoStream:
313.            """Camera object that controls video streaming from the Picamera"""
314.            def __init__(self,resolution=(640,480),framerate=30):
315.                # Initialize the PiCamera and the camera image stream
316.                self.stream = cv2.VideoCapture(0)
317.                ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG
    '))
318.                ret = self.stream.set(3,resolution[0])
319.                ret = self.stream.set(4,resolution[1])
320.
321.                # Read first frame from the stream
322.                (self.grabbed, self.frame) = self.stream.read()
323.
324.            # Variable to control when the camera is stopped
325.                self.stopped = False
326.
327.            def start(self):
328.            # Start the thread that reads frames from the video stream
329.                Thread(target=self.update,args=()).start()
330.                return self
331.
332.            def update(self):
333.                # Keep looping indefinitely until the thread is stopped
334.                while True:
335.                    # If the camera is stopped, stop the thread
336.                    if self.stopped:
337.                        # Close camera resources
338.                        self.stream.release()
339.                        return
340.
341.                    # Otherwise, grab the next frame from the stream
342.                    (self.grabbed, self.frame) = self.stream.read()
343.
344.            def read(self):
345.            # Return the most recent frame
346.                return self.frame
347.
348.            def stop(self):
349.            # Indicate that the camera and thread should be stopped
350.                self.stopped = True
351.
352.        # Define and parse input arguments
353.        parser = argparse.ArgumentParser()
354.        parser.add_argument('--
    modeldir', help='Folder the .tflite file is located in',
355.                            default='TFLite_model')
356.        parser.add_argument('--
    graph', help='Name of the .tflite file, if different than detect.tflite',
357.                            default='detect.tflite')
358.        parser.add_argument('--
    labels', help='Name of the labelmap file, if different than labelmap.txt',
```

```
                            default='labelmap.txt')
        parser.add_argument('--threshold', help='Minimum confidence threshold for displaying detected objects',
                            default=0.5)
        parser.add_argument('--resolution', help='Desired webcam resolution in WxH. If the webcam does not support the resolution entered, errors may occur.',
                            default='1280x720')
        parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up detection',
                            action='store_true')

        args = parser.parse_args()

        MODEL_NAME = args.modeldir
        GRAPH_NAME = args.graph
        LABELMAP_NAME = args.labels
        min_conf_threshold = float(args.threshold)
        resW, resH = args.resolution.split('x')
        imW, imH = int(resW), int(resH)
        use_TPU = args.edgetpu

        # Import TensorFlow libraries
        # If tflite_runtime is installed, import interpreter from tflite_runtime, else import from regular tensorflow
        # If using Coral Edge TPU, import the load_delegate library
        pkg = importlib.util.find_spec('tflite_runtime')
        if pkg:
            from tflite_runtime.interpreter import Interpreter
            if use_TPU:
                from tflite_runtime.interpreter import load_delegate
        else:
            from tensorflow.lite.python.interpreter import Interpreter
            if use_TPU:
                from tensorflow.lite.python.interpreter import load_delegate

        # If using Edge TPU, assign filename for Edge TPU model
        if use_TPU:
            # If user has specified the name of the .tflite file, use that name, otherwise use default 'edgetpu.tflite'
            if (GRAPH_NAME == 'detect.tflite'):
                GRAPH_NAME = 'edgetpu.tflite'

        # Get path to current working directory
        CWD_PATH = os.getcwd()

        # Path to .tflite file, which contains the model that is used for object detection
        PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

        # Path to label map file
        PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

        # Load the label map
        with open(PATH_TO_LABELS, 'r') as f:
            labels = [line.strip() for line in f.readlines()]

        # Have to do a weird fix for label map if using the COCO "starter model" from
        # https://www.tensorflow.org/lite/models/object_detection/overview
        # First label is '???', which has to be removed.
        if labels[0] == '???':
```

```
413.             del(labels[0])
414.
415.         # Load the Tensorflow Lite model.
416.         # If using Edge TPU, use special load_delegate argument
417.         if use_TPU:
418.             interpreter = Interpreter(model_path=PATH_TO_CKPT,
419.                                       experimental_delegates=[load_delegate('libedgetpu.
    so.1.0')])
420.             print(PATH_TO_CKPT)
421.         else:
422.             interpreter = Interpreter(model_path=PATH_TO_CKPT)
423.
424.         interpreter.allocate_tensors()
425.
426.         # Get model details
427.         input_details = interpreter.get_input_details()
428.         output_details = interpreter.get_output_details()
429.         height = input_details[0]['shape'][1]
430.         width = input_details[0]['shape'][2]
431.
432.         floating_model = (input_details[0]['dtype'] == np.float32)
433.
434.         input_mean = 127.5
435.         input_std = 127.5
436.
437.         # Initialize frame rate calculation
438.         frame_rate_calc = 1
439.         freq = cv2.getTickFrequency()
440.
441.         # Initialize video stream
442.         videostream = VideoStream(resolution=(imW,imH),framerate=30).start()
443.         time.sleep(1)
444.
445.         #for frame1 in camera.capture_continuous(rawCapture, format="bgr",use_video_port
    =True):
446.         while True:
447.
448.             # Start timer (for calculating frame rate)
449.             t1 = cv2.getTickCount()
450.
451.             # Grab frame from video stream
452.             frame1 = videostream.read()
453.
454.             # Acquire frame and resize to expected shape [1xHxWx3]
455.             frame = frame1.copy()
456.             frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
457.             frame_resized = cv2.resize(frame_rgb, (width, height))
458.             input_data = np.expand_dims(frame_resized, axis=0)
459.
460.             # Normalize pixel values if using a floating model (i.e. if model is non-
    quantized)
461.             if floating_model:
462.                 input_data = (np.float32(input_data) - input_mean) / input_std
463.
464.             # Perform the actual detection by running the model with the image as input

465.             interpreter.set_tensor(input_details[0]['index'],input_data)
466.             interpreter.invoke()
467.
468.             # Retrieve detection results
```

```
            boxes = interpreter.get_tensor(output_details[0]['index'])[0] # Bounding box coordinates of detected objects
            classes = interpreter.get_tensor(output_details[1]['index'])[0] # Class index of detected objects
            scores = interpreter.get_tensor(output_details[2]['index'])[0] # Confidence of detected objects
            #num = interpreter.get_tensor(output_details[3]['index'])[0]  # Total number of detected objects (inaccurate and not needed)

            # Loop over all detections and draw detection box if confidence is above minimum threshold
            for i in range(len(scores)):
                if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):

                    # Get bounding box coordinates and draw box
                    # Interpreter can return coordinates that are outside of image dimensions, need to force them to be within image using max() and min()
                    ymin = int(max(1,(boxes[i][0] * imH)))
                    xmin = int(max(1,(boxes[i][1] * imW)))
                    ymax = int(min(imH,(boxes[i][2] * imH)))
                    xmax = int(min(imW,(boxes[i][3] * imW)))

                    cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

                    # Draw label
                    object_name = labels[int(classes[i])] # Look up object name from "labels" array using class index
                    label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example: 'person: 72%'
                    labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Get font size
                    label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label too close to top of window
                    cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw white box to put label text in
                    cv2.putText(frame, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text

            # Draw framerate in corner of frame
            cv2.putText(frame,'FPS: {0:.2f}'.format(frame_rate_calc),(30,50),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,0),2,cv2.LINE_AA)

            # All the results have been drawn on the frame, so it's time to display it.

            cv2.imshow('Object detector', frame)

            # Calculate framerate
            t2 = cv2.getTickCount()
            time1 = (t2-t1)/freq
            frame_rate_calc= 1/time1

            # Press 'q' to quit
            if cv2.waitKey(1) == ord('q'):
                break

        # Clean up
        cv2.destroyAllWindows()
        videostream.stop()
```

## Apéndice D. Código de detección usando TensorFlow y CloudVision

```
1.  ######## Detector de Placas Vehiculares usando TensorFLow y una Camara Pi #########
2.  #
3.  # Autores : Juan Vargas & William Moncada
4.  # Fecha: 21/12/2020
5.  # Descripción:
6.  #El siguiente programa usa un modelo de detección de objetos mediante TensorFLow Lite
7.  #para reconocer una placa vehicular. Asi mismo se hace la lectura de carácteres mediant
    e
8.  #Googl Vision.
9.
10. #
11. # Este código está basado en :
12. #https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-
    Raspberry-Pi/blob/master/TFLite_detection_image.py#
13. # Se agregaron los métodos de captura,clasificación correspondientes y OCR.
14.
15. # Importar paquetes
16. import os, io
17. import argparse
18. import cv2
19. import numpy as np
20. import sys
21. import glob
22. import importlib.util
23. from google.cloud import vision
24. from google.cloud.vision import types
25. import pandas as pd
26. import re
27.
28.
29. cam = cv2.VideoCapture(0)
30.
31. cv2.namedWindow("test")
32.
33. img_counter = 0
34.
35. while True:
36.     ret, frame = cam.read()
37.     if not ret:
38.         print("Fallo al tomar la foto")
39.         break
40.     cv2.imshow("Camara", frame)
41.
42.     k = cv2.waitKey(1)
43.     if k%256 == 27:
44.         # ESC presionado
45.         print("Escape hit, closing...")
46.         break
47.     elif k%256 == 32:
48.         # Espacio presionado
49.         img_name = "Imagen{}.jpg".format(img_counter)
50.         cv2.imwrite(img_name, frame)
51.         print("{} Guardada!".format(img_name))
52.         img_counter += 1
53.
```

```
54.
55. cam.release()
56.
57. cv2.destroyAllWindows()
58.
59.
60. # Definir y analizar argumentos de entrada Define and parse input arguments
61. parser = argparse.ArgumentParser()
62. parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
63.                     default='TFLite_model')
64. parser.add_argument('--
    graph', help='Name of the .tflite file, if different than detect.tflite',
65.                     default='detect.tflite')
66. parser.add_argument('--
    labels', help='Name of the labelmap file, if different than labelmap.txt',
67.                     default='labelmap.txt')
68. parser.add_argument('--
    threshold', help='Minimum confidence threshold for displaying detected objects',
69.                     default=0.5)
70. parser.add_argument('--
    image', help='Name of the single image to perform detection on. To run detection on mul
    tiple images, use --imagedir',
71.                     default='Imagen0.jpg')
72. parser.add_argument('--
    imagedir', help='Name of the folder containing images to perform detection on. Folder m
    ust contain only images.',
73.                     default=None)
74. parser.add_argument('--
    edgetpu', help='Use Coral Edge TPU Accelerator to speed up detection',
75.                     action='store_true')
76.
77. args = parser.parse_args()
78.
79. MODEL_NAME = args.modeldir
80. GRAPH_NAME = args.graph
81. LABELMAP_NAME = args.labels
82. min_conf_threshold = float(args.threshold)
83. use_TPU = args.edgetpu
84.
85. # Analizar el nombre y el directorio de la imagen de entrada Parse input image name and
     directory.
86. IM_NAME = args.image
87. IM_DIR = args.imagedir
88.
89. # Si se especifican tanto una imagen como una carpeta, arroja un error If both an image
     AND a folder are specified, throw an error
90. if (IM_NAME and IM_DIR):
91.     print('¡Error! Utilice únicamente el argumento --image o --imagedir, no ambos')
92.     sys.exit()
93.
94. # Si no se especifica ni una imagen ni una carpeta, por defecto se usa 'test1.jpg' para
     el nombre de la imagen If neither an image or a folder are specified, default to using
     'test1.jpg' for image name
95. if (not IM_NAME and not IM_DIR):
96.     IM_NAME = 'test1.jpg'
97.
98. # Importar TensorFlow librerias
99.
100.        pkg = importlib.util.find_spec('tflite_runtime')
101.        if pkg:
102.            from tflite_runtime.interpreter import Interpreter
```

```
            if use_TPU:
                from tflite_runtime.interpreter import load_delegate
        else:
            from tensorflow.lite.python.interpreter import Interpreter
            if use_TPU:
                from tensorflow.lite.python.interpreter import load_delegate

        # Si usa Edge TPU, asigne un nombre de archivo para el modelo de Edge TPU If usi
    ng Edge TPU, assign filename for Edge TPU model
        if use_TPU:
            # Si el usuario ha especificado el nombre del archivo .tflite, use ese nombr
    e; de lo contrario, use el predeterminado 'edgetpu.tflite' If user has specified the na
    me of the .tflite file, use that name, otherwise use default 'edgetpu.tflite'
            if (GRAPH_NAME == 'detect.tflite'):
                GRAPH_NAME = 'edgetpu.tflite'


        # Obtener la ruta al directorio de trabajo actual
        CWD_PATH = os.getcwd()

        # Defina la ruta a las imágenes y tome todos los nombres de archivo de imagen
        if IM_DIR:
            PATH_TO_IMAGES = os.path.join(CWD_PATH,IM_DIR)
            images = glob.glob(PATH_TO_IMAGES + '/*')

        elif IM_NAME:
            PATH_TO_IMAGES = os.path.join(CWD_PATH,IM_NAME)
            images = glob.glob(PATH_TO_IMAGES)

        # Ruta al archivo .tflite, que contiene el modelo que se utiliza para la detecci
    ón de objetos
        PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

        # Ruta al archivo label map
        PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

        # Cargar the label map
        with open(PATH_TO_LABELS, 'r') as f:
            labels = [line.strip() for line in f.readlines()]


        if labels[0] == '???':
            del(labels[0])

        # Cargar el modelo de Tensorflow Lite.
        # Si usa Edge TPU, use el argumento especial load_delegate
        if use_TPU:
            interpreter = Interpreter(model_path=PATH_TO_CKPT,
                                      experimental_delegates=[load_delegate('libedgetpu.
    so.1.0')])
            print(PATH_TO_CKPT)
        else:
            interpreter = Interpreter(model_path=PATH_TO_CKPT)

        interpreter.allocate_tensors()

        # Obtener detalles del modelo
        input_details = interpreter.get_input_details()
        output_details = interpreter.get_output_details()
        height = input_details[0]['shape'][1]
        width = input_details[0]['shape'][2]
```

```
159.
160.       floating_model = (input_details[0]['dtype'] == np.float32)
161.
162.       input_mean = 127.5
163.       input_std = 127.5
164.
165.       # Recorrer cada imagen y realice la detección
166.       for image_path in images:
167.
168.           # Cargue la imagen y cambie el tamaño a la forma esperada [1xHxWx3]
169.           image = cv2.imread(image_path)
170.           image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
171.           imH, imW, _ = image.shape
172.           image_resized = cv2.resize(image_rgb, (width, height))
173.           input_data = np.expand_dims(image_resized, axis=0)
174.
175.           # Normalice los valores de los píxeles si utiliza un modelo flotante (es dec
    ir, si el modelo no está cuantificado)
176.           if floating_model:
177.               input_data = (np.float32(input_data) - input_mean) / input_std
178.
179.           # Realice la detección real ejecutando el modelo con la imagen como entrada

180.           interpreter.set_tensor(input_details[0]['index'],input_data)
181.           interpreter.invoke()
182.
183.           # Recuperar resultados de detección
184.           boxes = interpreter.get_tensor(output_details[0]['index'])[0] # Bounding box
     coordenadas de los objetos detectados
185.           classes = interpreter.get_tensor(output_details[1]['index'])[0] # Índice de
    clase de objetos detectados
186.           scores = interpreter.get_tensor(output_details[2]['index'])[0] # Confianza d
    e los objetos detectados
187.
188.           # Recorra todas las detecciones y dibuje el cuadro de detección si la confia
    nza está por encima del umbral mínimo
189.           for i in range(len(scores)):
190.               if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):
191.
192.                   # Obtener las coordenadas del cuadro delimitador y dibujarlo
193.                   ymin = int(max(1,(boxes[i][0] * imH)))
194.                   xmin = int(max(1,(boxes[i][1] * imW)))
195.                   ymax = int(min(imH,(boxes[i][2] * imH)))
196.                   xmax = int(min(imW,(boxes[i][3] * imW)))
197.
198.                   cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)
199.                   crop_img = image[ymin:ymax, xmin:xmax] #Segmentar la placa
200.
201.
202.                   crop_img = cv2.resize(crop_img, None, fx = 2, fy = 2, interpolation
    = cv2.INTER_CUBIC)
203.
204.
205.
206.
207.
208.                   # Dibujar etiqueta
209.                   object_name = labels[int(classes[i])] # Usando classes buscamos la e
    tiqueta
210.                   label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Generamos l
    a precisión de detección
```

```
                    labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Creamos un fondo
                    label_ymin = max(ymin, labelSize[1] + 10) # Aegurar no dibujar la etiqueta demasiado cerca de la parte superior de la ventana
                    cv2.rectangle(image, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Dibujar una caja blanca para poner el texto
                    cv2.putText(image, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Dibujar la etiqueta de texto

                # Todos los resultados se han dibujado en la imagen, ahora muestra la imagen.
            try:
                cv2.imshow('Object detector', image)
                cv2.imshow("Placa",crop_img)
                cv2.imwrite("Platee.jpg",crop_img)

            except NameError:
                print("No se detectó ninguna placa")
                break

            #Colocar la llave para acceder a Cloud Vision
            os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = r'Key_VisionAPI.json'

            client= vision.ImageAnnotatorClient()

            #Generar la ruta de acceso a la imagen detectada y recortada
            FILE_NAME='Platee.jpg'
            FOLDER_PATH=r'C:\tensorflow1\models\research\object_detection'
            #Leer la imagen
            with io.open(os.path.join(FOLDER_PATH,FILE_NAME),'rb') as image_file:
                content=image_file.read()

            image= vision.types.Image(content=content)
            response= client.text_detection(image=image)
            texts=response.text_annotations
            #Configurar el archivo recibido
            df=pd.DataFrame(columns=['locale','description'])

            for text in texts:
                df=df.append(dict(locale=text.locale,description=text.description),ignore_index=True)

            if len(df['description'][1]) >=6:
                clean_text = re.sub('[\W_]+', '', df['description'][1])
                print('La placa es: ',clean_text)

            elif len(df['description'][2]) >3:
                clean_text = re.sub('[\W_]+', '', df['description'][1])
                print('La placa es: ',clean_text)
            else:
                clean_text = re.sub('[\W_]+', '', df['description'][1]+df['description'][2])
                print('La placa es: ',clean_text)


            # Presionar q para cerrar
            if cv2.waitKey(0) == ord('q'):
                break

        # Limpiar
```

```
264.         cv2.destroyAllWindows()
```