PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Trabajo de Titulación

TEMA:

Entrenamiento del modelo YOLO para detección de una placa vehicular previamente capturada en imagen o video y aplicación de OCR para obtención de sus caracteres

AUTOR:

Steven Danny Peñafiel Falcón

DIRECTOR:

Luis Oswaldo Espinosa Viteri

QUITO DM, DICIEMBRE DE 2022

DEDICATORIA

A mi madre Mercedes, por ser un apoyo no solo en mi proceso educativo, si no también, en mi vida. Por siempre guiar y soportar las decisiones que tome.

A mi padre José, por siempre estar presente y ser un consejero en cada situación complicada que se me presento.

A mi compañera de vida Emily, por brindarme una nueva perspectiva en mi modo de vida y mejorar cada día con su presencia.

A mis hermanos Michael y Nick, por estar siempre presentes a pesar de las dificultades.

AGRADECIMIENTOS

A mis padres, José y Mercedes, por hacer posible la culminación de mis estudios, con su apoyo económico, pero sobre todo su apoyo moral y sus consejos.

A mi pareja Emily, por darme la motivación necesaria para continuar en cada decisión tomada.

A mis hermanos, Michael y Nick, por incentivarme a ser una mejor persona.

A mi tutor, Luis, por guiar el presente trabajo.

RESUMEN

Los avances en el campo del aprendizaje profundo resultado de la cantidad de datos que se generan de forma diaria y de la capacidad de procesamiento de estos, han permitido desarrollar modelos pre-entrenados de código abierto, a los cuales se les puede aplicar transferencia de aprendizaje para realizar tareas específicas. Claro ejemplo de esto es el modelo YOLO para la detección de objetos, área que tradicionalmente usa tecnologías de visión artificial, pero que con el uso de aprendizaje profundo ha presentado mejoras en la extracción de características, dado que, este proceso se realiza de forma automática. Por lo que, el presente estudio entrena al modelo YOLO en su séptima versión para detectar placas vehiculares capturadas con anterioridad en ficheros tipo imagen o video, y lo complementa con la tecnología conocida como reconocimiento óptico de caracteres (OCR) para extraer y retornar los caracteres de la misma.

Palabras claves: aprendizaje profundo, YOLO, reconocimiento óptico de caracteres, OCR.

Tabla de contenidos

DEDICATORIA i

AGRADECIMIENTOS ii

RESUMEN iii

ABSTRACT iv

Tabla de figuras v

1. INTRODUCCIÓN 1

1.1. Tema 1

1.2. Justificación 1

1.3. Planteamiento del problema 2

1.4. Objetivos Generales 2

1.4.1. Objetivos Específicos 2

1.5. Alcance 3

2. MARCO TEÓRICO 4

2.1. Datos 4

2.1.1. Tipos de datos 4

2.2. Aprendizaje de Maquina 5

2.2.1. Tipos de aprendizaje 5

2.3. Aprendizaje Profundo 6

2.4. Detección de Objetos 7

2.5. YOLO 8

2.6. Reconcomiendo Óptico de Caracteres 10

2.7. Sistemas de Reconocimiento de Placas Vehiculares 10

2.8. Herramientas 11

2.8.1. Jupyter 11

2.8.2. Google Colaboratory 11

2.8.3. Python 12
2.8.4. PyTorch 12
2.8.5. OpenCV 13
2.8.6. EasyOCR 13
2.8.7. YOLOv7 13
3. Entrenamiento del modelo YOLOv7 y aplicación de OCR 15
3.1. Instalación y configuración 15
3.2. Entrenamiento modelo YOLOv7 17
3.2.1. Entrenamiento 17
3.2.2. Pruebas modelo YOLOv7 20
3.2.3. Evaluación 21
3.3. Reconocimiento óptico de caracteres 25
3.4. Pruebas 30
4. Resultados 32
5. Conclusiones, Recomendaciones y Líneas Futuras 35
5.1. Conclusiones 35
5.2. Recomendaciones 35
6. BIBLIOGRAFÍA 36

Índice de figuras

Figura 1. Funcionamiento de un modelo de aprendizaje profundo. Adaptado de Chollet, F. (2017). Deep learning with python. New York, NY: Manning Publications. 7

Figura 2. Arquitectura del modelo YOLO. Tomado de Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. arxiv. 9

Figura 3 . Arquitectura del modelo YOLOv4 y YOLOv7. Tomado de Bochkovskiy A., Wang C., Liao M. (23 de Abril de 2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arxiv. 14
Figura 4. Pantalla de inicio Google Colab 15
Figura 5. Sistema operativo 16
Figura 6. Estructura de los ficheros del conjunto de datos 16
Figura 7. Estructura del fichero creado al clonar el repositorio YOLOv7 17
Figura 8. Instalación con la herramienta pip de las bibliotecas a utilizar por el modelo YOLOv7 y el reconocedor óptico de caracteres 17
Figura 9. Ejemplo fichero de datos para el modelo YOLOv7 18
Ilustración 10. Información de la GPU 19
Figura 11. Comando para entrenar el modelo YOLOv7 19
Figura 12. Comando para probar el modelo YOLOv7 20
Figura 13. Comando para detectar una imagen con el modelo YOLOv7 20
Figura 14. Detección de la placa vehicular del comando presentado en la figura 13 .. 20
Ilustración 15. Matriz de confusión modelo YOLOv7 entrenado 21
Figura 16. Grafica del valor de precisión respecto al valor de confianza 23
Figura 17. Grafica del valor sensibilidad respecto al valor de confianza 24
Figura 18. Grafica del valor de precisión respecto al valor de sensibilidad 24
Figura 19. Importación de librerías e inicialización de variables globales. 26
Figura 20. Función main 26
Figura 21. Función detect_image 27
Figura 22. Función detect_license_plate 27
Figura 23. Función plot_boxes 28
Figura 24. Función get_characters 28
Figura 25. Función filter_text 29
Figura 26. Función detect_video 29
Figura 27. Llamada a la función main para detectar ficheros tipo imagen y video. 30
Figura 28. Porcentaje de detecciones correctas e incorrectas en imágenes. 32
Figura 29. Detección del segundo fichero tipo imagen ingresado 33
Figura 30. Detección del quinto fichero tipo imagen ingresado 33
Figura 31. Porcentaje de detecciones correctas e incorrectas en videos 34

Índice de tablas

Tabla 1. Detecciones en ficheros tipo imagen 30
Tabla 2. Detecciones en ficheros tipo video. 34

1. INTRODUCCIÓN

1.1. Tema

Entrenamiento del modelo YOLO para detección de una placa vehicular previamente capturada en imagen o video y aplicación de OCR para obtención de sus caracteres.

1.2. Justificación

El reconocimiento automático de placas hace referencia a la tecnología capaz de obtener los caracteres de la placa de un vehículo, que puede ser usada para extraer información del automóvil, como su localización o modelo. Muchos de los sistemas hacen uso de técnicas de visión por computadora tradicionales para identificar el posicionamiento de la placa. No obstante, en la actualidad se tienen diferentes métodos para cumplir el mismo objetivo que suelen presentar mejores resultados en la mayoría de los casos.

Actualmente existe una gran cantidad de datos, que gracias a los modelos de aprendizaje automático pueden ser transformados en información, siendo importante mencionar que estos modelos se entrenan de mejor manera en datos estructurados, por ejemplo, datos organizados en filas y columnas. Sin embargo, los datos no siempre tienen una estructura, en otras palabras, existen datos no estructurados, como lo son las imágenes o videos, que se procesan mediante aprendizaje profundo, un subcampo del aprendizaje automático basado en redes neuronales artificiales donde los modelos se entrenan y aprenden de forma distinta, siendo capaces de extraer características de forma automática de datos no estructurados (International Business Machines [IMB], 2020). Teniendo aplicaciones en el campo de la detección de objetos y en específico para este estudio en la detección de la placa del automóvil.

Un ejemplo de esto es YOLO un modelo capaz de ser entrenado en imágenes completas y de optimizar el rendimiento de detección de forma simultánea, logrando así reconocer los objetos en un menor tiempo de respuesta. No obstante, para la obtención de los caracteres de la placa previamente detectada es necesario hacer uso del proceso conocido como reconocimiento óptico de caracteres, que es usado para convertir una imagen a un formato de tipo texto descifrable para un computador. (Amazon Web Services [AWS], s.f.)

1.3. Planteamiento del problema

Como se mencionó anteriormente gran parte de los sistemas de reconocimiento de placas vehiculares hacen uso de técnicas tradicionales de visión por computadora. Lo cual puede generar resultados imprecisos cuando se cambia el tipo de placa o la ubicación de esta.

Las mejoras en los dispositivos, por ejemplo, la capacidad en memoria, el poder computacional, la resolución en los sensores de imágenes, etc., y los avances en el campo del aprendizaje profundo permiten obtener resultados más precisos, puesto que, las redes neuronales artificiales pueden ser entrenadas de forma no supervisada en tareas como la clasificación de imágenes o detección de objetos. Mientras que por el otro lado las técnicas tradicionales de visión por computadora tienen que ser programadas (O' Mahony, y otros, 2019), lo cual presenta los conflictos presentados en el párrafo anterior.

En función de esta problemática se plantea la siguiente pregunta principal de investigación:

- ¿Cómo se realizarán los procesos de entrenamiento del modelo YOLO para la detección de una placa vehicular previamente capturada en una imagen o video y de aplicación de OCR para obtener sus caracteres?

Y las preguntas secundarias:

- ¿Qué versión del modelo YOLO será usada?
- ¿Cuáles son las herramientas que se emplearan para entrenar y complementar el modelo YOLO para el reconocimiento de placas vehiculares?
- ¿Qué métricas serán usadas para evaluar los resultados?

1.4. Objetivos Generales

Entrenar el modelo YOLO para detectar una placa vehicular previamente capturada en imagen o video y aplicar OCR para obtener sus caracteres.

1.4.1. Objetivos Específicos

- Establecer la versión del modelo YOLO a usarse.
- Determinar las herramientas que se emplearan para entrenar y complementar el modelo YOLO para el reconocimiento de placas vehiculares.
- Establecer las métricas a usarse para evaluar los resultados.

1.5. Alcance

El presente estudió tiene como alcance el reconocimiento de los caracteres de una placa vehicular previamente capturada en imagen o video mediante el entrenamiento del modelo YOLO y aplicación del proceso de OCR para obtener sus caracteres.

Finalmente, como resultado de esta investigación se entregará un modelo capaz de detectar una placa vehicular en base a una imagen o video y su obtención de caracteres mediante OCR.

2. MARCO TEÓRICO

2.1. Datos

En la actualidad el uso de dispositivos electrónicos, por ejemplo, celulares, computadores, televisores inteligentes, entre otros, trajo consigo una generación de datos constante, por lo que, la palabra dato, proveniente del latín dado o proporcionado, ha adquirido un papel fundamental tanto para las empresas como para la sociedad en general. Prada (2008) define los datos como el mecanismo mediante el cual le es posible al ser humano identificar algún aspecto de la realidad, mientras que Gutiérrez (s.f.) menciona que los datos son símbolos (letras, números, etc.) que representan una descripción, palabra, medida o cantidad.

En base a estas definiciones los datos se pueden interpretar como el recurso mediante el cual la humildad ha logrado usar la tecnología para mejorar varios procesos haciendo uso de sistemas informáticos, ya que, simbolizan algún hecho, situación o valor que puede ser almacenados e incluso procesados para generar información relevante que puede ser usado en futuros análisis.

2.1.1. Tipos de datos

2.1.1.1. Datos estructurados

Son aquellos datos organizados de forma clara y reconocible que por lo general se almacenan mediante tablas, es decir, las columnas se usan para identificar lo que quiere representar el valor y las filas para almacenar el valor en sí. Bermúdez (2002) hace referencia a estos como datos con una estructura regida y bien definida por el lugar en el que son almacenados, así también, menciona a las bases de datos relacionales como un ejemplo de fuente estructurada que puede ser consultada de forma precisa mediante el lenguaje SQL.

2.1.1.2. Datos no estructurados

Como su nombre lo indica este tipo de datos son el opuesto de los datos estructurados, en otras palabras, no pueden ser almacenados en bases de datos relacionales o tablas, pues, no están organizados de forma explícita y por lo general es difícil establecer una estructura.

Por esto para Snadaker y Rima (2014) el problema de estos datos esta dado por la dificultad al momento de ordenarlos, manejarlos y organizarlos, dando como ejemplos imágenes, videos, archivos de audios, etc.

2.1.1.3. Datos semiestructurados

De acuerdo con Ambika (2020) comparten características de los datos estructurados y no estructurados, esto quiere decir que se los puede definir como la combinación de los dos. Algunos ejemplos son los archivos tipo csv y json, mientras que una de las fuentes más comunes para encontrar este tipo de datos son las bases de datos no relacionales.

2.2. Aprendizaje de Maquina

Woolf (2009) se refiere al aprendizaje de máquina, o aprendizaje automático, como la habilidad que tienen los sistemas para mejorar mediante la obtención de nuevo conocimiento basado en observación de nuevos datos, en otras palabras, se considera como aprendizaje de maquina a los sistemas o modelos con la aptitud de auto aprender nuevas relaciones de un conjunto de datos. Por lo general, se lo divide en la fase de entrenamiento, en la que el modelo recibe datos de entrada para aprender patrones y la fase de pruebas, en la que se ingresan datos diferentes a la fase anterior con la finalidad de comprobar la capacidad del modelo.

Del mismo modo, para los autores Chanal et al. (2021) la inteligencia artificial en la actualidad tiene aplicaciones prácticas gracias al aprendizaje automático, por lo que, pasa a ser un subcampo de la misma. Así mismo, Subasi (2020) menciona que el aprendizaje de maquina es un conjunto de ideas adaptadas de varias disciplinas en un periodo de tiempo, dando como resultado un campo multidisciplinario e interdisciplinario en el que la estadística, el cálculo multivariable, los métodos de optimización matemáticos, la algebra lineal, entre otros, son necesarios para el desarrollo de nuevos modelos que generalicen reglas a partir de una muestra. Alugos ejemplos de algoritmos de aprendizaje automático son K-means, redes bayesianas, arboles de decisión, regresión lineal.

En la actualidad, es usado por varias empresas para transformar los datos generados por los usuarios en información que se empela en la toma decisiones o la mejora en la experiencia de sus clientes. Existen diferentes metodologías que involucran aprendizaje de máquina para realizar este procedimiento, tal es el caso de KDD, Knowledge Discovery in Databases o Descubrimiento de conocimiento en bases de datos.

2.2.1. Tipos de aprendizaje

2.2.1.1. Aprendizaje supervisado

Se hace referencia al aprendizaje supervisado cuando el conjunto de datos, que se utilizó para entrenar un modelo, tiene una variable de salida o una variable objetivo, dicho de otra

manera, se puede controlar el proceso de aprendizaje del modelo, ya que, se conoce la respuesta que debe ser entregada. Un ejercicio en el que se aplica aprendizaje supervisado es en la predicción del precio de casas, pues, el modelo se entrena con un conjunto de datos que contenga el precio de las casas, la variable de salida u objetivo, y los factores que influyen en este, las variables de entrada o características.

2.2.1.2. Aprendizaje no supervisado

El aprendizaje no supervisado contempla los modelos que se entrenan en conjuntos de datos que no contienen una variable de salida u objetivo, dificultando el proceso de inspección del entrenamiento. La agrupación, o clustering, es un ejemplo de aprendizaje no supervisado donde el modelo encuentra o crea grupos de los datos de entrada mediante interpretación.

2.3. Aprendizaje Profundo

También conocido en inglés como deep learning, es un subcampo del aprendizaje de máquina que ha adquirido una gran popularidad en la actualidad gracias a su funcionamiento y a los campos donde está siendo usado, como lo son: el manejo de vehículos sin conductor, asistentes virtuales, reconocimiento de objetos, la mejora en la conversión de texto a voz y reconocimiento de voz.

Los autores Mao et al. (2019) destacan una de las características más usadas para describir el aprendizaje profundo que es su similitud con el funcionamiento del cerebro, pues, los modelos hacen uso de redes neuronales en diferentes capas para aprender, siendo importante mencionar que la relación es figurativa, ya que, los procesos subyacentes son diferentes, pero existe una conexión y activación de neuronas de forma similar al cerebro.

El funcionamiento de un modelo de aprendizaje profundo se realiza mediante el bucle de aprendizaje y comienza con el ingreso de los datos, de los cuales se almacenará un número que representa lo que se hizo con ellos en lo que conoce como peso o parámetros que es único para cada capa. Para el control de la predicción dada por la capa se hace uso de la función de perdida, o loss function, que calcula la diferencia entre el valor de salida y el valor deseado, conocido como valor de perdida, que ayuda a medir cuan acertada fue la respuesta. El valor calculado por la función de perdida es usado como retroalimentación para ajustar el valor almacenado en el peso o parámetro de cada capa respectivamente, con el fin de disminuir el resultado de la función de perdida en la siguiente iteración, y se lo realiza mediante el optimizador que hace

uso de un algoritmo de retro propagación, el algoritmo central en el aprendizaje profundo (Chollet, 2017).

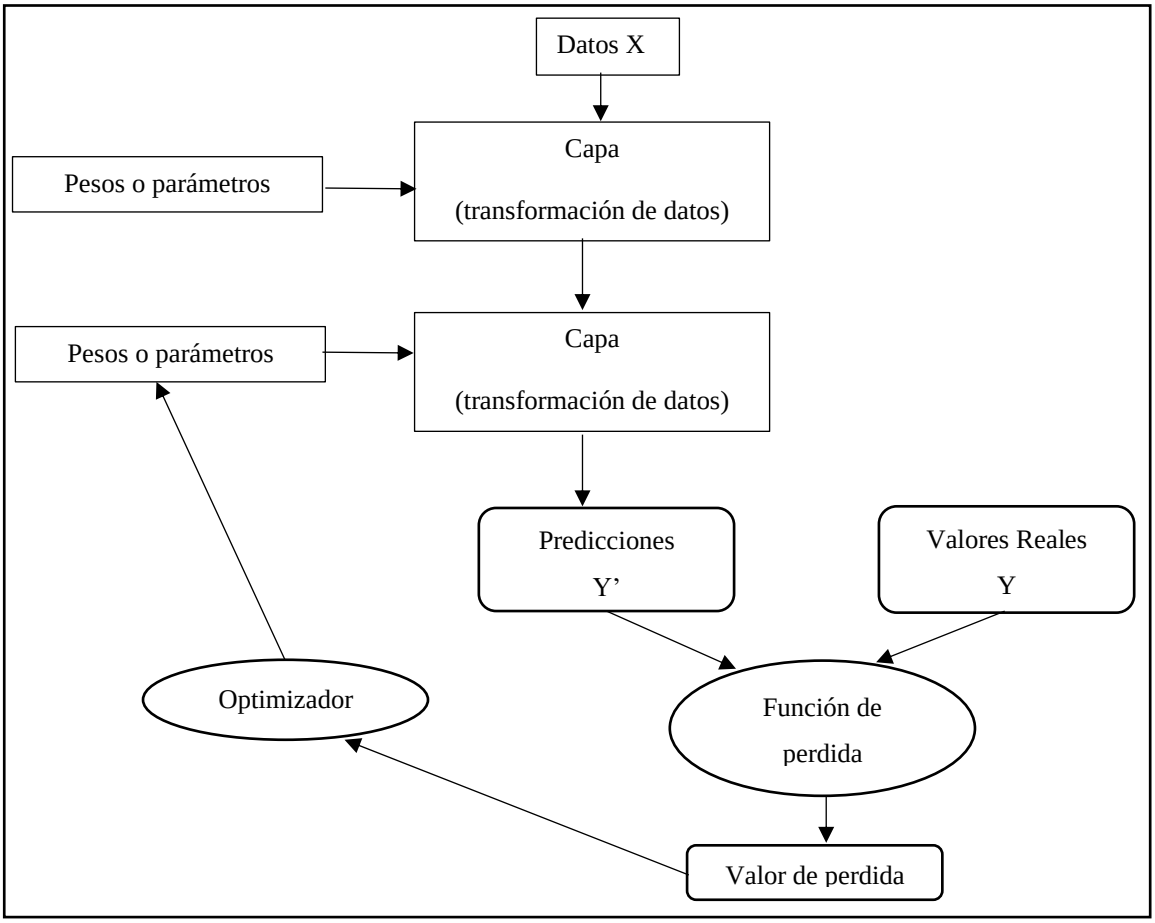


Figura 1. Funcionamiento de un modelo de aprendizaje profundo. Adaptado de Chollet, F. (2017). Deep learning with python. New York, NY: Manning Publications.

2.4. Detección de Objetos

Se la considera un subcampo de la visión por computador el cual ha adquirido una gran popularidad en los últimos años gracias a los avances en el campo del aprendizaje profundo y a las capacidades de los dispositivos actuales. Zou et al. (2019) definen la detección de objetos como una tarea de la visión por computadora que tiene como objetivo detectar instancias de objetos visuales de cierta clase, que pueden ser humanos, animales, carros, entre otros, en imágenes digitales. Del mismo, categorizan la detección de objetos en dos grupos de investigación, la detección de objetos general y las aplicaciones de la detección de objetos. En la primera categoría se buscar desarrollar métodos que simulen el conocimiento y la visión humana para la detección de diferentes tipos de objetos, por otra parte, la segunda hace

referencia a las aplicaciones de estos métodos en escenarios específicos, es decir, detección facial, detección de texto, etc.

Para Zheng et al. (2019) los principales retos de la detección de objetos están en detectar donde se encuentran ubicados los objetos en la imagen, nombrado localización de objetos, y su categorización, denominado clasificación de objetos. Dividiendo así el proceso de los modelos tradicionales en tres etapas: selección de región informativa, se escanea la imagen en busca de las posiciones; extracción de características, se extraen las características que permitirán reconocer los objetos; clasificación, se categoriza cada objeto en categorías antes definidas, haciendo así la representación más jerárquica e informativa para el reconocimiento visual.

Los avances del aprendizaje profundo han permitido el desarrollo de modelos de detección de objetos que realizan un proceso un poco diferente a los modelos tradicionales, dicho de otra forma, los métodos de entrenamiento y las arquitecturas de los modelos de aprendizaje profundo actuales logran aprender características más complejas, sin la necesidad de un diseño o delimitación manual. Lo cual genera un beneficio en la detección, ya que, en la etapa de extracción de característica los modelos tradicionales usan métodos que se ven afectados por el cambio en la iluminación, el fondo, la apariencia del entorno, entre otros, mientras que, los modelos que hacen uso de aprendizaje profundo funcionan de mejor manera frente a cambios del entorno.

2.5. YOLO

You Only Look Once (YOLO) es un modelo presentando en un artículo científico en el año 2016 el cual tuvo y tiene una gran aceptación, pues, su funcionamiento introducía un nuevo método para la detección de objetos que era más rápido en comparación a los modelos DPM y R-CNN, usados en ese año. Por ejemplo, el modelo R-CNN reconoce objetos haciendo uso de métodos conocidos como region proposal para la generación de cuadros delimitadores (zonas de interés en la imagen), que pasan por un clasificador encargado de calcular y asignar a los objetos encontrados un valor probabilístico de la clase a la que podría pertenecer respectivamente, para finalizar con un proceso que incluye su depuración, la eliminación de detecciones duplicadas y un ajuste al valor de la probabilidad asignado anteriormente, en base a las otras detecciones (Girshick, Donahue, Darrell, & Malik, 2013). Mientras que, para el desarrollo del modelo se planteó la detección de objetos como un problema de regresión simple, es decir, a diferencia del proceso descrito anteriormente YOLO realiza todas las fases en una

sola, dando así origen a su nombre, ya que, solo se “mira” una vez la imagen para realizar todo el proceso de detección.

El modelo esta implementado como una red neuronal convolucional, donde las veinte y cuatro primeras capas convolucionales extraen las características de la imagen y las 2 ultimas capas completamente conectadas predicen las probabilidades y las coordinadas de los objetos. Si bien YOLO toma como referencia el modelo GoogLeNet, se diferencia ya que hace uso de capas de reducción con dimensiones 1x1 seguidas de capas convolucionales con dimensiones 3x3.

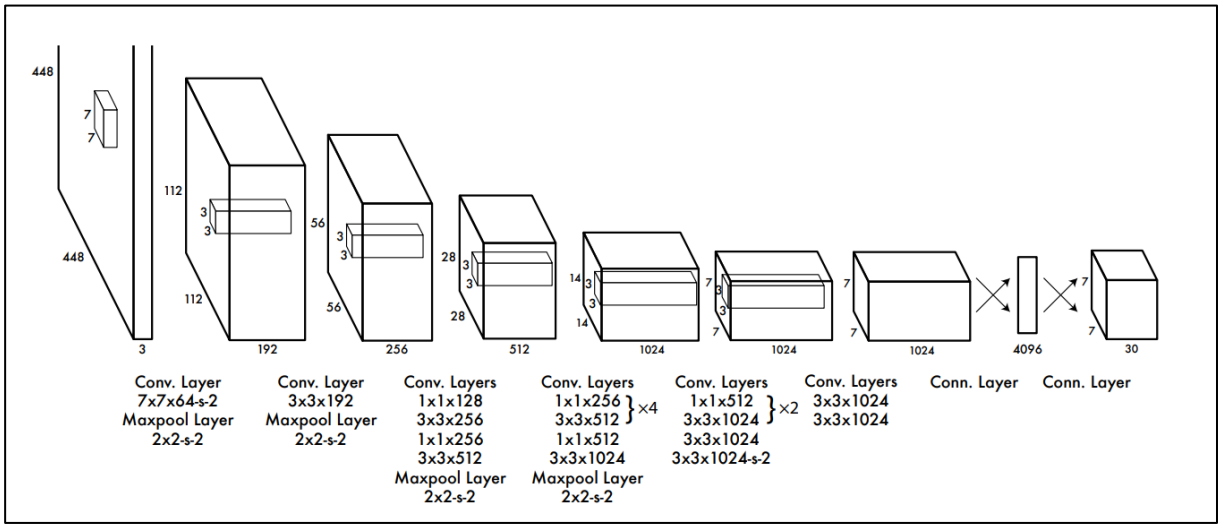


Figura 2. Arquitectura del modelo YOLO. Tomado de Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. arxiv.

En base a lo anterior descrito, Redmon et al. (2015) presentan los tres beneficios que tiene YOLO, por un lado, es extremadamente rápido, dado que, al realizar el proceso de forma simultánea se eliminan las complejas tuberías o pipelines usadas para conectar las fases, logrando así optimizar el tiempo de respuesta y duplicar la precisión promedio de otros sistemas en tiempo real. Por el otro, esto le permite tener una representación global de la imagen, por lo que, presenta mejores resultados al codificar el contexto y la apariencia de la clase u objeto reduciendo a la mitad el número de errores dados por el entorno en comparación con el modelo Fast R-CNN. Y, por último, el modelo YOLO supera a modelos como DPM, pues, generaliza las representaciones de los objetos, lo cual le permite detectar objetos en imágenes inesperadas con una probabilidad menor de fallo. Siendo importante mencionar que se puede añadir una cuarta ventaja, pues, todo el código de entrenamiento y prueba hace uso de la licencia GNU General Public License v3.0.

2.6. Reconocimiento Óptico de Caracteres

Conocida como OCR del inglés Optical Character Recognition, es el proceso mediante el cual se extrae y procesa el texto de una imagen o documento en formato legible para el ordenador haciendo uso tanto de software como de hardware (IBM, 2022). Por ejemplo, en caso de tener capturada la placa de un vehículo en una imagen se puede aplicar OCR para extraer los caracteres en formato texto y, de ser necesario, realizar las ediciones necesarias, siendo importante mencionar que la efectividad de la extracción dependerá de la calidad de la imagen y en especial de la calidad donde se ubican de los caracteres de placa.

Por lo general el funcionamiento de un software de OCR esta dado por las siguientes fases: adquisición de imagen, se lee y convierte la imagen en datos binarios, clasificando las áreas claras como fondo y a las áreas oscuras como texto; preprocesamiento, se limpia la imagen solucionando problemas de inclinación que se pudieron dar en el escaneo, eliminando el ruido y suavizando los bordes, limpiando los cuadros y líneas, aplicando scripts de multilenguaje para soporte los diferentes caracteres de los idiomas; reconocimiento de texto, los dos algoritmos principales son la coincidencia de patrones y la extracción de características, el primero separa cada carácter y lo compara con los glifos almacenados, por lo que solo funciona cuando tienen una fuente similar. El segundo divide los glifos, en características que pueden ser las direcciones e intersecciones de las líneas, bucles cerrados, etc., y las usa para encontrar la mejor coincidencia en los glifos almacenados; post-procesamiento, se convierte el texto extraído en un archivo. (Amazon Web Services [AWS], s.f.)

2.7. Sistemas de Reconocimiento de Placas Vehiculares

Se pueden definir como un conjunto de componentes que permiten reconocer los caracteres de la placa de un vehículo capturado en formato imagen o video de manera automática, es decir, el sistema es capaz de localizar la ubicación de la placa en la imagen o video y de extraer los caracteres en forma texto. Por lo que, es usado en detectores de velocidad, en parqueaderos para la gestión del ingreso, localización y obtención de información de vehículos mediante su placa, entre otros.

Su funcionalidad se puede dividir en dos fases: la localización de la placa vehicular y el reconocimiento óptico de caracteres. Los métodos usados para cada fase dependen de cada sistema, por ejemplo, para la localización de la placa vehicular en los sistemas tradicionales se hace uso de técnicas de visión por computadora, sin embargo y como se mencionó con

anterioridad, los avances en el campo del aprendizaje automático permiten entrenar modelos para el reconocimiento de objetos, los cuales por lo general tienden a disminuir el porcentaje de errores relacionados a las predicciones con imagen de entrada que tiene cambios en la iluminación, posición de la placa, etc. Mientras que para el reconocimiento óptico de caracteres se usan métodos como los descritos en la sección anterior.

Si bien este tipo de sistemas no son actuales, pues, el prototipo de EMI Electronics Computer Recognition System desarrollado y usado en los años setenta para detectar vehículos hurtados (Durán, 2021) se considera el primer sistema de este tipo. Siguen tienen gran presencia en diferentes ámbitos en la actualidad, ya que, los avances tanto en el software como en el hardware requerido para su desarrollo permiten mejorar y optimizar muchos de los procesos de sistemas ya existentes o nuevos. Dado que es una tecnología usada a lo largo del tiempo por diferentes empresas u organizaciones resulta importante mencionar que se la conoce de diferentes maneras, entre las que están, Automatic License-Plate Recognition (ALPR), Automatic Number Plate Recognition (ANPR), Car-Plate Recognition (CPR), etc.

2.8. Herramientas

2.8.1. Jupyter

Es un proyecto de código abierto y sin fines de lucro que se originó a partir de la tecnología IPython de la cual se usa terminal, y a su vez funciona como su kernel, puesto que, permite usar computación interactiva otorgando diferentes funcionalidades como un shell interactivo, herramientas GUI, etc. (IPython, n.d.)

Del mismo modo, otra de las partes fundamentales de su arquitectura son los cuadernos Jupyter que se pueden definir como datos estructurados en formato JSON y con la extensión .ipynb, usados para representar código, metadatos, contenido y respuestas. Los cuales cuentan con una interfaz para la visualización de código, multimedia, permitir la colaboración, entre otros. (Jupyter, n.d.)

2.8.2. Google Colaboratory

También conocido como Google Colab es un producto de Google Research, el cual permite la escritura y ejecución del lenguaje de programación Python en el navegador de forma rápida e intuitiva haciendo uso del proyecto de código abierto Jupyter, es decir, otorga a sus usuarios la posibilidad de usar y compartir cuadernos Jupyter sin la necesidad de configurarlos. (Google, n.d.)

Si bien puede ser usada de forma gratuita se debe tener en cuenta que sus recursos no están garantizados ni son ilimitados, por ejemplo, cuando se hace demasiado uso de las GPUs se restringe las ejecuciones del cuaderno o incluso de la cuenta hasta actualizar a algunos de los planes de paga. Así pues, en este estudio fue necesario el uso del plan Colab Pro, ya que, a diferencia de las GPUs Nvidia T4 que tienen cierto límite en el plan gratuito se puede hacer uso de las GPUs Nvidia V100 o A100, lo cual resulta de gran utilidad al momento de entrenar el modelo YOLO para la detección de la placa el automóvil, sin embargo, es importante aclarar que el uso tampoco es ilimitado.

2.8.3. Python

Es un lenguaje de programación de alto nivel e interpretado, es decir, su sintaxis es bastante similar al idioma inglés y no necesita compilar sus instrucciones a lenguaje de máquina, si no que las ejecuta directamente.

En la actualidad se encuentra presente en la mayoría de las áreas, en especial en la ciencia de datos, pues, es amigable y fácil de aprender tanto para usuarios avanzados como para usuarios nuevos gracias a su sintaxis, comunidad, documentación, bibliotecas. Del mismo modo, al ser de código abierto es de libre uso y distribución con una licencia administrada por Python Software Foundation.

El repositorio PyPI, Python Package Index, es un repositorio que permite encontrar y descargar software desarrollado por la comunidad de Python para aplicarlo en diferentes disciplinas, por ejemplo, programación web, acceso a bases de datos, ciencia, educación, desarrollo de videojuegos, inteligencia artificial, etc.

2.8.4. PyTorch

Un framework código abierto que busca facilitar el proceso desarrollo de modelos de aprendizaje automático incluyendo su puesta en producción, del mismo modo, esta optimizado para el procesamiento de tensores, o tensors, con GPUs y CPUs en modelos de aprendizaje profundo.

Entre sus características están: entrenamiento distribuido, tiene la capacidad de entrenar modelos de forma distribuida, optimizando su rendimiento; soporte en la nube, se encuentra disponible en las plataformas en la nube principales, por ejemplo, Amazon Web Services, Google Cloud Platform, Microsoft Azure; ecosistema completo, es decir, cuenta con diferentes

herramientas y bibliotecas que permiten el desarrollo de procesamiento de lenguaje natural (NLP, Natural Language Processing), visión por computadora, entre otros. (PyTorch, s.f.)

2.8.5. OpenCV

Open Source Computer Vision Library, es una biblioteca de código abierto que incluye diversos algoritmos de visión por computadora y que tienen una estructura modular, por lo que, contiene bibliotecas compartidas o estáticas, entre las que se encuentran: core, módulo que define estructuras de datos básicas; imgproc, módulo de procesamiento de imágenes, por ejemplo, filtrado de imágenes lineales y no lineales, transformaciones de imágenes geométricas, conversión de espacio de color, histogramas, etc.; 3d, algoritmos básicos de geometría de vista múltiple, estimación de pose de objetos y elementos de reconstrucción 3D; video, módulo para análisis de video que incluye sustracción de fondo, estimación de movimiento y algoritmos de seguimiento de objetos; objdetect, detección de objetos e instancias de clases predefinidas, por ejemplo, personas, automóviles; entre otras. (OpenCV, s.f.)

2.8.6. EasyOCR

Como su nombre lo indica, es un paquete de Python desarrollado por la empresa Jaided AI para el reconocimiento óptico de caracteres que esta implementada en PyTorch y que puede hacer uso de la tecnología CUDA en las GPU Nvidia, en caso de contar con una, para acelerar la detección de texto. En su repositorio de GitHub se presenta como un OCR listo para usar con soporte para más de ochenta lenguajes y todos los scripts de escritura populares, por ejemplo, latín, mandarín, arábico, etc. (JaidedAI, s.f.)

2.8.7. YOLOv7

Desde el lanzamiento del modelo YOLO han existido diferentes mejoras al mismo, las cuales se publican mediante versiones que incluyen un paper en el que se detallan los métodos utilizados y los resultados, la versión utilizada en este documento es la séptima. Conocida como YOLOv7 fue publicada el 6 del Julio del presente año, por los autores Chien-Yao Wang, Alexey Bochkovskiy y Hong-Yuan Mark Liao quienes a su vez han trabajado en otras versiones de YOLO, por ejemplo, YOLOv4, YOLOv5, YOLOR, entre otros. La arquitectura esta basada en la versión cuatro de YOLO.

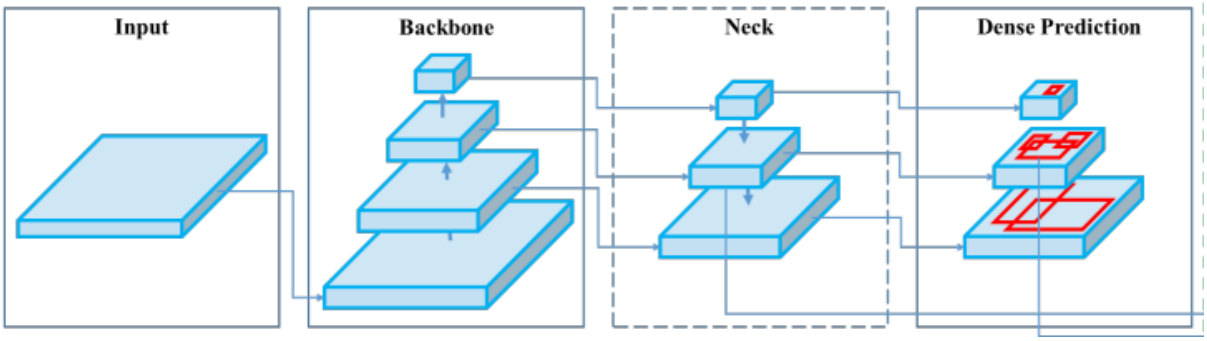


Figura 3 . Arquitectura del modelo YOLOv4 y YOLOv7. Tomado de Bochkovskiy A., Wang C., Liao M. (23 de Abril de 2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arxiv.

Como se puede observar en la figura 3, la arquitectura comienza con el input o el ingreso de la imagen que puede ser un array de dos dimensiones que representa los colores RGB de la imagen, dado que un video es un conjunto de imágenes este también es un formato valido de entrada. Para extraer las características principales de la imagen se hace uso del backbone que es una red neuronal compuesta por varias capas convolucionales y que por lo general es entrenada con redes neuronales preentrenadas, en el caso de YOLOv7 se usaron VoVNET, CSPVoVNET, ELAN, E-LAN. Antes de presentar los resultados el modelo hace uso de la capa conocida como neck, la cual se usa para unir las soluciones de los diferentes estados de la capa backbone, para la cual se usaron los métodos FPN, RFB, PAN. Para finalizar el modelo utiliza la capa head o dense prediction que se encarga de realizar las predicciones localizando y clasificando los objetos.

Estas implementaciones le permitieron superar sus anteriores versiones al obtener una precisión promedio de 56.8% en videos mayores a 30fps, así mismo, redujo el tiempo computacional en un 36%, pues, necesita 75% menos parámetros que el modelo YOLOv4. (Wang, Bochkovskiy, & Liao, 2022)

3. Entrenamiento del modelo YOLOv7 y aplicación de OCR

3.1. Instalación y configuración

Para el desarrollo del presente trabajo se hace uso de un cuaderno de Jupyter en la plataforma Google Colab, por lo que, las configuraciones son las predeterminadas. Recordando que el plan usado es Google Colab Pro por los beneficios relacionados a la tarjeta gráfica mencionados anteriormente. Antes de realizar los siguientes pasos, es necesario contar con una cuenta de Google, en caso de no contar con una realizar el registro correspondiente.

Como primer paso se ingresa a la plataforma de Google Colab, para lo que se puede hacer uso de un buscador web o mediante la url https://colab.research.google.com/, donde se presenta la pantalla de la figura 3 y se selecciona la opción nuevo notebook.

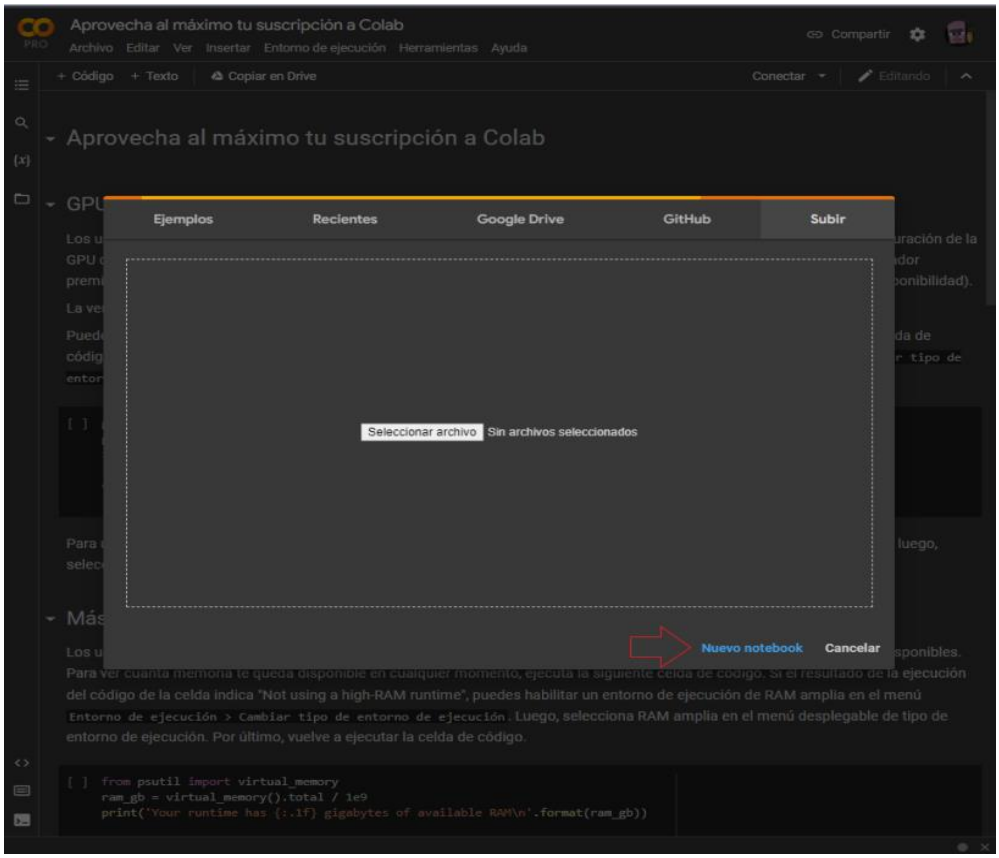


Figura 4. Pantalla de inicio Google Colab

Una vez creado el cuaderno de Jupyter en Google Colab se ejecuta el comando !cat /etc/*release con el fin de conocer el sistema operativo, en este caso Ubuntu en su versión 18.04.6 LTS.

```
!cat /etc/*release

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.6 LTS"
```

Figura 5. Sistema operativo

El conjunto de datos se obtuvo de repositorio públicos de Kaggle, sin embargo, fue necesario ordenar los datos en imágenes jpg con un archivo txt en formato tabla donde la primera columna representa la etiqueta de la clase, la segunda y tercera el centro en el eje x y el centro en el eje y del objeto, la cuarta y la quinta el ancho y el alto. Así mismo, los datos fueron estructurados por ficheros como se muestra en la figura 6, ya que, es la estructura que el modelo YOLOv7 necesita. Los datos están alojados en el servicio Google Drive, razón por la que se importa la clase drive desde la biblioteca google.colab para montar los archivos a la ruta /content/gdrive y copiar el conjunto de datos desde la ruta /content/gdrive/MyDrive/alpr/dataset a la ruta /content.

Figura 6. Estructura de los ficheros del conjunto de datos

Para hacer uso del modelo yolov7 se clona el repositorio de GitHub donde está alojado con el comando !git clone https://github.com/WongKinYiu/yolov7.git, el cual creara un directorio con el nombre yolov7 que contendrá los archivos presentados en la figura 7.

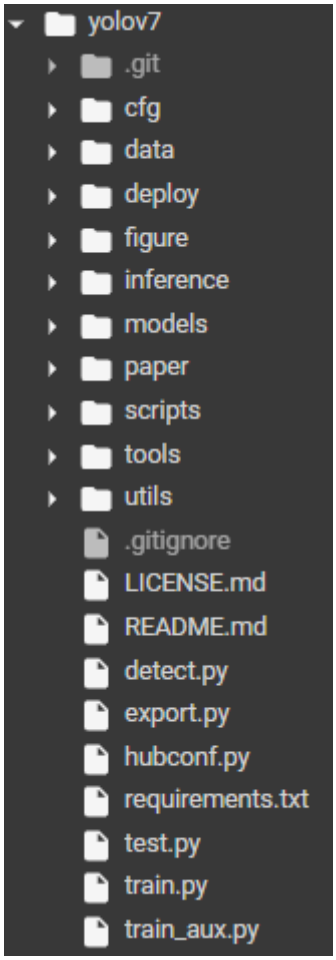


Figura 7. Estructura del fichero creado al clonar el repositorio YOLOv7

Antes de entrenar el modelo es necesario instalar las bibliotecas de las que hace uso, que se encuentran en el fichero requeriments.txt ubicado en el directorio creado al clonar el repositorio. Para ello se cambia la ruta actual a /content/yolov7 con el comando cd y se instalan las bibliotecas del archivo antes mencionado con la utilidad pip, pero en este caso se aumenta la biblioteca easyocr utilizada en los siguientes pasos para obtener los caracteres de placa.

```
!pip install -r requirements.txt easyocr
```

Figura 8. Instalación con la herramienta pip de las bibliotecas a utilizar por el modelo YOLOv7 y el reconocedor óptico de caracteres

3.2. Entrenamiento modelo YOLOv7

3.2.1. Entrenamiento

Para el entrenamiento existe la necesidad de crear dos archivos en formato yaml, el primero contendrá cinco variables, tres donde se establecerán las rutas a los directorios creados

anteriormente en la estructura mencionada, una para el número de clases y una para los nombres de las clases, en este caso, solo se tiene una clase que es la placa del vehículo, llamada license-plate. El segundo corresponde a la configuración del modelo, por lo que se ajustó la configuración del modelo YOLOv7 al presente proyecto, en la cual solo le actualiza el número de clases a uno, pero se mantienen las demás variables que son: depth_multiple para agregar capas al modelo; width_multiple que aumenta el número de filtros de la capa; anchors o los cuadros delimitadores con una altura y ancho predefinidos de los que el modelo hará uso y de ser necesario ajustará; backbone la red neuronal previamente entrenada que extrae las características principales; head la capa donde se realizara la predicción.

```
%%writefile data/alpr.yaml
train: '../dataset/images/train'
val: '../dataset/images/val'
test: '../dataset/images/test'

# Classes
nc: 1  # number of classes
names: ['license-plate']  # class names
```

Figura 9. Ejemplo fichero de datos para el modelo YOLOv7

Se ejecuta el comando !nvidia-smi para verificar que se esta haciendo uso de la GPU y de la tecnología CUDA para un mejor funcionamiento. En caso de que el entorno de ejecución no esté siendo acelerado por GPU, se lo puede activar desde el apartado recursos, cambiar tipo de entorno de ejecución y en acelerador de hardware se debe seleccionar GPU.

```
!nvidia-smi

Tue Oct 18 21:56:29 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  A100-SXM4-40GB      Off  | 00000000:00:04.0 Off |                    0 |
| N/A   28C    P0    45W / 400W |      0MiB / 40536MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

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

Ilustración 10. Información de la GPU

Para entrenar el modelo se hace uso del comando presentando en la figura 11, donde train.py es el fichero que entrena al modelo; workers, el número de núcleos del CPU usados; batch-size, el número de imágenes procesadas antes de actualizar el modelo; data y cfg, los ficheros creados anteriormente, weights, pesos del modelo YOLOv7 original; name, el nombre del modelo que se está entrenando.

```
!python train.py --workers 4 --device 0 --batch-size 8 --data data/alpr.yaml \
--cfg cfg/training/yolov7_alpr.yaml --weights yolov7.pt \
--name yolov7_alpr --hyp data/hyp.scratch.custom.yaml
```

```
     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
   299/299     7.41G   0.01589   0.00239         0   0.01828         8       640: 100% 23/23 [00:03<00:00,  6.63it/s]
              Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:00<00:00,  8.02it/s]
                all          27          28       0.964       0.964       0.982       0.567
300 epochs completed in 0.439 hours.
```

Figura 11. Comando para entrenar el modelo YOLOv7

Al ejecutar el comando se realizan un total de 300 iteraciones para entrenar el modelo, las se completaron en 30 minutos, se generan los pesos del modelo que serán guardando en la ruta runs/train/yolov7_alpr/weights, de los que se recomienda hacer del uso del fichero best.pt, y se presentara métricas da cada iteración que serán tratadas más adelante.

3.2.2. Pruebas modelo YOLOv7

Para verificar el comportamiento del modelo y obtener las métricas de evaluación que se analizaran posteriormente, se utiliza el comando de la figura 12, en el que test.py es el archivo que realiza las pruebas y genera las métricas del modelo; weights son los pesos del modelo creados en su entrenamiento; tast es la tarea por realizar; data el fichero con las rutas. Y en este caso la ruta donde se almacenan los archivos es runs/test/exp

```
!python test.py --weights runs/train/yolov7_alpr/weights/best.pt --task test --data data/alpr.yaml
```

Figura 12. Comando para probar el modelo YOLOv7

Del mismo modo se puede hacer uso del comando presentado en la figura 13 que usa el fichero detect.py para detectar la placa vehicular de la imagen especificada en el parámetro source, haciendo uso de los pesos del modelo pasados al parámetro weights. Almacenando los resultados en runs/detect/exp

```
!python detect.py --weights runs/train/yolov7_alpr/weights/best.pt --source /content/dataset/testPrediction.jpg
```

Figura 13. Comando para detectar una imagen con el modelo YOLOv7

Figura 14. Detección de la placa vehicular del comando presentado en la figura 13

3.2.3. Evaluación

Como se mencionó anteriormente al ejecutar el fichero test.py se generan ciertas métricas, las cuales son: matriz de confusión, precisión, sensibilidad o exhaustividad y mAP; en el directorio runs/test/exp, no obstante, es necesario definir los siguientes términos: TP, true positive o verdaderos positivos, predicciones correctas del modelo; FP, false positive o falsos positivos, detecciones incorrectas del modelo; FN, false negative o falsos negativos, detecciones que el modelo no realizo pero que eran correctas; TN, true negative o verdaderos negativos, predicciones que el modelo no realizo porque eran incorrectas. Dado que, son necesarios para entender tanto la métrica matriz de confusión, presentada en la figura 15, como las siguientes métricas a tratar.

En la matriz de confusión se puede observar que un 90% de las predicciones fueron correctas, un 10% de las placas vehiculares no se predijeron y un 100% de las predicciones incorrectas fueran clasificadas como placas vehiculares, lo cual concuerda con el número de clases que se tiene para el conjunto de datos. En el caso de los detectores de objeto, las predicciones falsas positivas no son tomadas en cuenta, pues, son predicciones que no se realizan porque eran incorrectas.

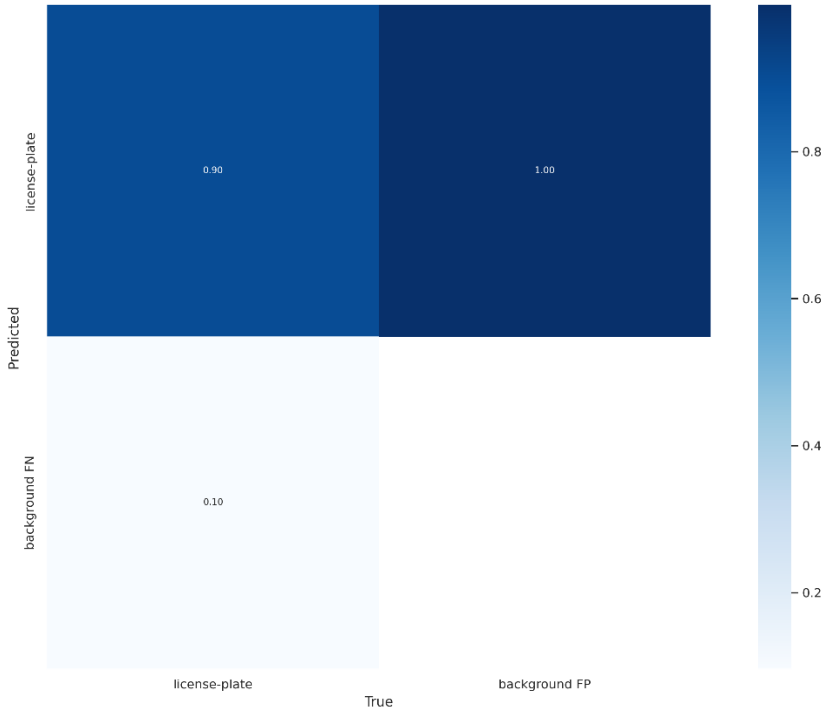


Ilustración 15. Matriz de confusión modelo YOLOv7 entrenado

Del mismo modo, se necesita entender el concepto de IOU, interesection over union o intersección sobre unión, que es un valor el cual cuantifica el grado de superposición entre dos cuadros delimitadores, el real y el predicho por el modelo. Y se calcula con la siguiente formula:

$$IOU = \frac{\acute{a}rea\ de\ superposición}{\acute{a}rea\ de\ union} = \frac{\acute{a}rea(B_p \cap B_{gt})}{\acute{a}rea(B_p \cup B_{gt})}$$

Donde, B_p representa el cuadro delimitador predicho y B_{gt} el cuadro delimitador real, también conocido en inglés como ground truth. El resultando puede estar entre el rango de 0 a 1, es decir, 0 representa que no existe superposición entre los cuadros delimitadores y 1 superposición total. Sin embargo, es necesario establecer un valor que valide si la predicción es correcta, por lo que, se usa el valor conocido como confianza. En otras palabras, si se establece un valor de confianza de 0.5 $IOU > 0.5$ se consideran predicción correctas o TP y $IOU \leq 0.5$ son predicción incorrectas o FP.

Así también, resulta importante definir las métricas conocidas como precisión y sensibilidad, por un lado, la precisión representa la capacidad del modelo para identificar solo los objetos relevantes, así como, en cuanto se puede confiar en las predicciones verdaderas (TP, TN). En otras palabras, ayuda a conocer cuántos de los valores predichos como verdaderos positivos y verdaderos negativos son correctos, por lo que, se calcula mediante la formula:

$$\frac{TP}{TP + FP}$$

Por lo que, esta métrica se grafica junto a diferentes valores de confianza como se puede observar en la figura 16, en la cual se evidencia que el valor de precisión es directamente proporcional al valor de confianza, es decir, al aumentar la confianza la precisión también aumentara y viceversa. Lo cual resulta lógico, ya que, al tener un valor de confianza alto se puede tener mayor seguridad de que las predicciones catalogadas como TP y TN sean correctas.

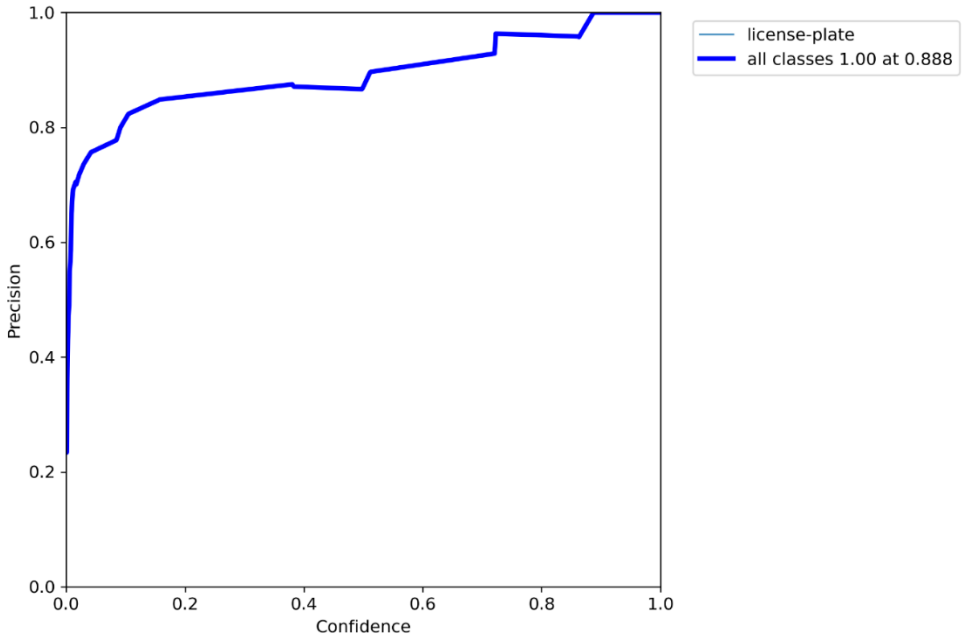


Figura 16. Grafica del valor de precisión respecto al valor de confianza

Por el otro lado, la exhaustividad o sensibilidad mide la habilidad del modelo para detectar todos los cuadros delimitadores reales, dicho de otra manera, permite conocer cuántos de los cuadros delimitadores reales se predijeron por el modelo. Siendo calculada mediante la formula:

$$\frac{TP}{TP + FN}$$

Al igual que la métrica de precisión, la métrica de sensibilidad se grafica con relación a diferentes valores de confianza, como se puede observar en la figura 17. Sin embargo, en este caso, los valores son inversamente proporcionales, dado que, al aumentar el valor de confianza el valor de sensibilidad decaerá y viceversa. Lo cual resulta inequívoco, puesto que, un aumento en el valor de confianza representa que las predicciones del modelo deberán tener un margen de error muy bajo, generando menos detecciones de cuadros delimitadores reales.

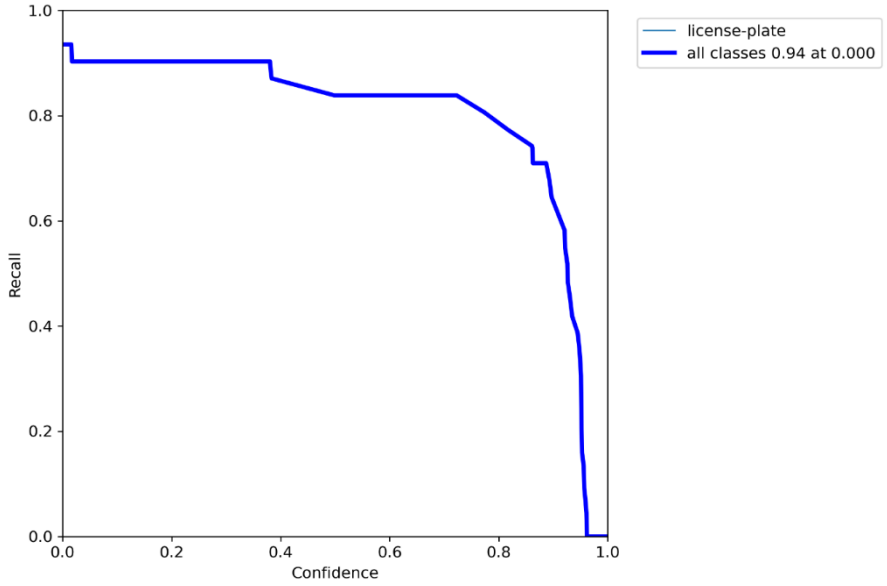


Figura 17. Grafica del valor sensibilidad respecto al valor de confianza

La curva PR es una métrica común en los detectores de objetos, pues, es una gráfica 60que representa los cambios en los valores de precisión y sensibilidad cuando se tienen diferentes valores de confianza. En este caso se puedo observar en la figura 18 como el valor de precisión decrece al aumentar el valor de sensibilidad, dicho de otra manera, el modelo necesita aumentar el número de detecciones, menor precisión, para detectar los cuadros delimitadores reales, mayor sensibilidad.

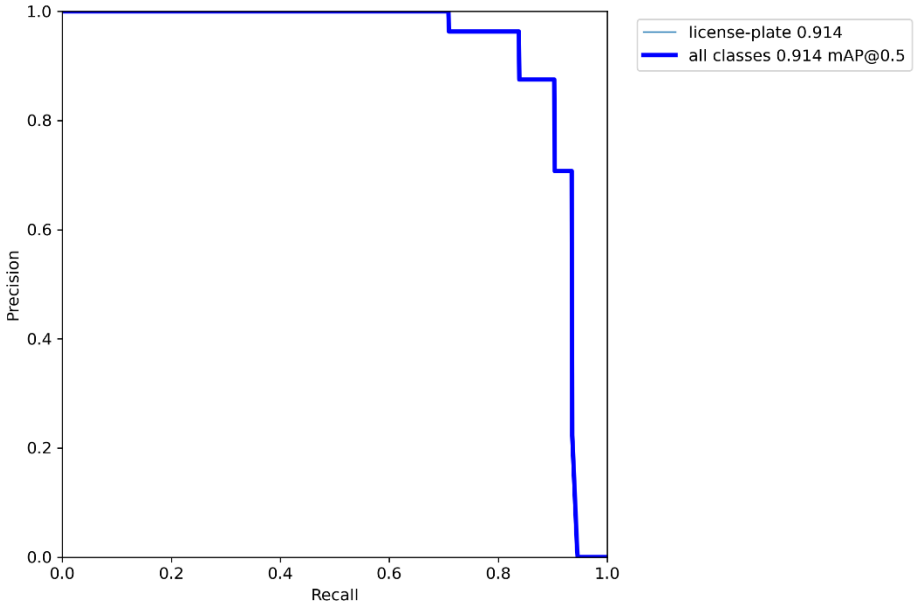


Figura 18. Grafica del valor de precisión respecto al valor de sensibilidad

Al calcular el área bajo la curva PR se obtiene la métrica AP, precisión promedio o avarage precision, que dependiendo el conjunto de datos también se la denomina mAP, usada para comparar el funcionamiento del modelo con diferentes clases e incluso para comparar diferentes modelos. Se puede definir como la precisión promedio para el rango de valores entre 0 y 1 de la sensibilidad, al igual que su nombre se calculada de diferentes maneras dependiendo el conjunto de datos de prueba. En el caso de YOLO se hace uso del conjunto de datos COCO, por lo que, se calcula un promedio para el AP interpolado de 101 puntos. Es decir, se recrea la forma de la curva PR usando 101 puntos igualmente espaciados [0, 0.01, …, 0.1, 0.11, …, 0.2, …, 0.9, 1] y se aplica la formula:

$$AP = \frac{1}{101}\sum_{r\in R} AP_r = \frac{1}{101}\sum_{r\in R} P_{interp}(r)$$

$$P_{interp}(r) = \max_{r':r'\geq r} p(r')$$

Donde, $p(r')$ es el valor de la precisión en el valor r' de la sensibilidad. Dicho de otra manera, en lugar de utilizar la precisión en todos los puntos de la curva, se interpola la precisión a los 101 puntos tomando la precisión máxima para de los valores con sensibilidad mayor a r.

Como se evidencia en la parte superior derecha de la figura 18, el resultado para el modelo entrenado fue de 0.914 usando un valor de confianza de 0.5. Del mismo modo, en la figura 11 se puede observar que la última iteración realizada por el modelo tuvo un mAP de 0.567 para valores de confianza entre 0.5 y 0.95.

3.3. Reconocimiento óptico de caracteres

Para obtener los caracteres de placa vehicular previamente detectada por el modelo se hace uso del lenguaje de programación Python, para la carga del modelo y la implementación de la librería easyOCR.

El programa comienza importando las librerías necesarias, inicializando la herramienta easyOCR en lenguaje español y estableciendo un valor de confianza en 0.2 para filtrar las secuencias de texto detectadas, la variable number_plate es inicializada como None, pues, será usada más adelante de manera global.

```python
from google.colab.patches import cv2_imshow
from utils.torch_utils import select_device
from models.yolo import Model
import torch
import cv2
import time
import re
import numpy as np
import easyocr
import matplotlib.pyplot as plt

EASY_OCR = easyocr.Reader(['es'])
OCR_TH = 0.2
number_plate=None
```

Figura 19. Importación de librerías e inicialización de variables globales.

La función main carga el modelo, obtiene el número de clases, en este caso license-plate, y verifica si el fichero ingresado es una imagen o un video, recibiendo como parámetros: model_path, ruta del modelo, file_path, ruta del archivo; isVideo, valor booleano que determina si el fichero de entrada es un video; results_path, ruta donde se almacenaran los resultados.

```python
def main(model_path=None , file_path=None, output_path = None, isVideo=False):
    print(f"[!] Cargando modelo... ")
    model =  torch.hub.load('.', 'custom', source ='local', path_or_model=model_path,force_reload=True)
    classes = model.names
    if isVideo:
      detect_video(model, file_path, output_path, classes)
    else:
      detect_image(model, file_path, output_path, classes)
```

Figura 20. Función main

En caso de que el fichero de entrada sea una imagen se ejecuta la función detect_image que recibe como parámetros el modelo cargado, la ruta del fichero de entrada, la ruta del fichero de salida y las clases, en este caso, license-plate. Es utilizada para leer la imagen que se carga en formato BGR, por lo que, se convierte el formato de color de BGR a RGB y se llama a la función detect_license_plate para detectar la placa vehicular. Una vez se tiene los resultados de la detección del modelo se regresa la imagen al formato BGR y se imprimen los resultados en la imagen de entrada, que a su vez serán guardados en el fichero de salida ingresado como parámetro.

```
def detect_image(model, img_path, output_path, classes):
    print(f"[!] Detectando placas vehiculares en imagen: {img_path}")
    frame = cv2.imread(img_path)
    frame = cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
    results = detect_license_plate(frame, model = model)
    frame = cv2.cvtColor(frame,cv2.COLOR_RGB2BGR)
    frame = plot_boxes(results, frame,classes = classes)
    cv2_imshow(frame)
    cv2.imwrite(f"{output_path}",frame)
```

Figura 21. Función detect_image

Por un lado, al ejecutarse la función detect_license_plate se envía como parámetros la imagen y el modelo con el fin de detectar la placa vehicular, para llamar al modelo es necesario convertir frame de tipo numpy array a lista. De las detecciones realizadas por el modelo se obtienen y devuelven las etiquetas y coordenadas detectadas.

```
def detect_license_plate(frame, model):
    frame = [frame]
    results = model(frame)
    labels, cordinates = results.xyxyn[0][:, -1], results.xyxyn[0][:, :-1]
    return labels, cordinates
```

Figura 22. Función detect_license_plate

Mientras que, por el otro, al llamar a la función plot_boxes se procesan las detecciones totales, es decir, recibe como parámetros los resultados de la detección de los cuales se obtienen las etiquetas y las coordenadas usadas para verifica que la detección tenga un valor de confianza mayor o igual a 0.5. Del mismo modo, recibe el parámetro use_last_number_plate que condiciona la llamada a la función get_characters, encargada de obtener los caracteres de la placa, pues, en caso de que el fichero sea un video se detectan y almacenan los caracteres en la variable global number_plate cada 5 frames. Finalizando con el retorno de la imagen con los caracteres escritos en la esquina superior izquierda del cuadro delimitador de la detección.

```python
def plot_boxes(results, frame, classes, use_last_number_plate=False):
    global number_plate
    labels, cord = results
    n = len(labels)
    x_shape, y_shape = frame.shape[1], frame.shape[0]
    for i in range(n):
        plate = cord[i]
        if plate[4] >= 0.50:
            x1, y1, x2, y2 = int(plate[0]*x_shape), int(plate[1]*y_shape), int(plate[2]*x_shape), int(plate[3]*y_shape)
            text_d = classes[int(labels[i])]
            coords = [x1,y1,x2,y2]
            if not use_last_number_plate:
              print(f"[!] Obteniendo caracteres placa vehicular Nº {i+1} . . . ")
              number_plate = get_characters(img = frame, coords= coords, region_threshold= 0.2)
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.rectangle(frame, (x1, y1-20), (x2, y1), (0, 255,0), -1)
            cv2.putText(frame, f"{number_plate}", (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,255,255), 2)
    return frame
```

Figura 23. Función plot_boxes

La función get_characteres recibe como parámetros la imagen, las coordenadas de la placa y un valor de confianza relacionado al texto que será detectado. Como primer paso se recorta la placa vehicular de la imagen, con las coordenadas recibidas, y se hace uso de la librería easyOCR para reconocer el texto, sin embargo, es necesario llamar a la función filter_text para validar los caracteres. Para finalizar se retorna los caracteres de la placa vehicular.

```python
def get_characters(img, coords,region_threshold):
    xmin, ymin, xmax, ymax = coords
    number_plate = img[int(ymin):int(ymax), int(xmin):int(xmax)]
    ocr_result = EASY_OCR.readtext(number_plate)
    text = filter_text(region=number_plate, ocr_result=ocr_result, region_threshold= region_threshold)
    if len(text) == 1:
        text = text[0]
    return text
```

Figura 24. Función get_characters

Como se mencionó anteriormente la función filter_text validara si el área del cuadro delimitador de los caracteres detectados dividido para el área total del cuadro delimitador de la placa vehicular es mayor al valor de confianza establecido, en este caso, 0.2. Por lo que recibe como parámetros, la región de la placa vehicular en la imagen, los textos detectados por la librería easyOCR y el valor de confianza. En primer lugar, se obtiene el área de la placa vehicular que será divida con el área del cuadro delimitador de cada texto reconocido, con el fin de validar que solo se tomen en cuenta los textos con un área mayor al 20% del área total de la placa vehicular, ya que, estos serán los retornados. Este proceso se realiza, ya que, por lo general si un texto en la placa vehicular es menor al 20% de su área puede no ser la secuencia de caracteres buscada.

```python
def filter_text(region, ocr_result, region_threshold):
    rectangle_size = region.shape[0]*region.shape[1]
    plate = []
    print(ocr_result)
    for result in ocr_result:
        length = np.sum(np.subtract(result[0][1], result[0][0]))
        height = np.sum(np.subtract(result[0][2], result[0][1]))
        if length*height / rectangle_size > region_threshold:
            plate.append(result[1])
    return plate
```

Figura 25. Función filter_text

En caso de que el fichero sea un video, la función main llamara la función detect_video enviando como parámetros el modelo cargado, el fichero de enterada, la ruta del fichero de salida, o video con las detecciones, y las clases. En la cual se extraen característica del video de entrada como lo son la altura, el ancho, los frames por segundo, el número total de frames, entre otras, usadas para recrear el video con las detecciones. Puesto que, un video es un conjunto de imágenes, o frames, se itera el video por cada frame haciendo uso de la librería openCV, y si bien se detecta la placa vehicular en cada iteración, los caracteres se reconocen cada 5 frames, a razón de mejorar la lectura para el ser humano en los resultados.

```python
def detect_video(model, vid_path, output_path, classes):
        print(f"[!] Detectando placas vehiculares en video: {vid_path}")
        cap = cv2.VideoCapture(vid_path)
        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = int(cap.get(cv2.CAP_PROP_FPS))
        totalFrames = int(cap.get(cv2. CAP_PROP_FRAME_COUNT))
        codec = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(output_path, codec, fps, (width, height))
        for frame_no in range(totalFrames):
          ret, frame = cap.read()
          frame = cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
          results = detect_license_plate(frame, model = model)
          frame = cv2.cvtColor(frame,cv2.COLOR_RGB2BGR)
          if ret and frame_no % 5 == 0:
            print(f"[!] Detectando frame № {frame_no} de {totalFrames} ")
            frame = plot_boxes(results, frame, classes = classes)
            out.write(frame)
          elif ret and frame_no % 1 == 0:
            frame = plot_boxes(results, frame, classes = classes, use_last_number_plate=True)
            out.write(frame)
        out.release()
        print(f"[!] Video guardado en {output_path}")
```

Figura 26. Función detect_video

3.4. Pruebas

Para probar en conjunto tanto la detección de la placa vehicular realizada por el modelo YOLOv7, como el reconocimiento de los caracteres por parte de la librería easyOCR se ejecutará el programa haciendo un llamado a la función main para un total de quince imágenes y cinco videos. Sin embargo, cabe aclarar que el llamado a la función main dependerá del tipo de fichero como se puede observar en la figura 27

```
main(model_path='runs/train/yolov7_alpr/weights/best.pt', file_path="/content/video.mp4", output_path="/content/video_resultante.mp4", isVideo=True)  #Video
main(model_path='runs/train/yolov7_alpr/weights/best.pt', file_path="/content/imagen.jpeg", output_path="/content/imagen_resultante.jpg") #Imagen
```

Figura 27. Llamada a la función main para detectar ficheros tipo imagen y video.

Algunos de los resultados obtenidos se presentan a continuación en la tabla 1.

Tabla 1. Detecciones en ficheros tipo imagen

Entrada	Salida
	
	


PAC-1795


PAC-1795


PPF-040


PPF-040

4. Resultados

Una vez realizadas las pruebas del modelo junto a la librería de OCR se pueden analizar los resultados obtenidos, sin embargo, es necesario establecer cuando un resultado se considera correcto. Por lo que, para el presente estudio se toman como resultados correctos aquellos que detecten los caracteres correspondientes al número de la placa vehicular, dicho de otra manera, la detección debe ser capaz de retornar los tres primeros y tres últimos caracteres de la placa del automóvil. Del mismo modo, se debe mencionar que para las imágenes con múltiples vehículos bastará con la detección de los caracteres de una placa vehicular para clasificarla como correcta.

Así pues, de las quince imágenes ingresadas se tiene un total de doce detecciones correctas y tres detecciones incorrectas, es decir, se tiene un 80% de detecciones correctas y un 20% de detecciones incorrectas, como se puede observar en la figura 28.

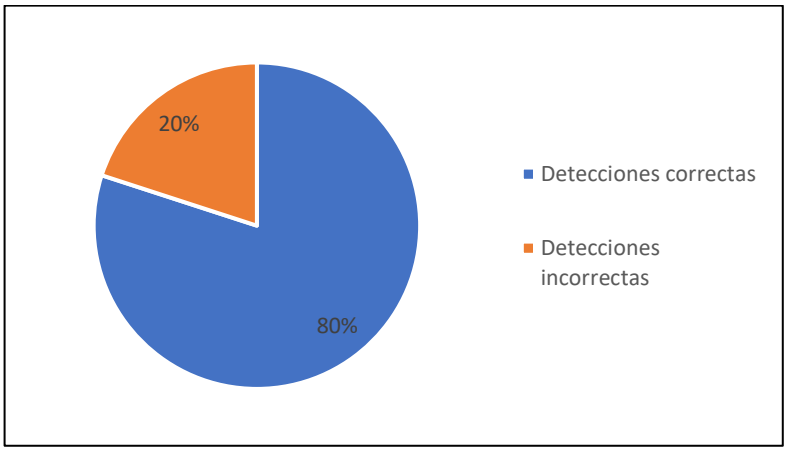


Figura 28. Porcentaje de detecciones correctas e incorrectas en imágenes.

No obstante, las detecciones incorrectas fallaron al reconocer los caracteres de la placa vehicular, en otras palabras, el modelo YOLOv7 entrenado fue capaz de detectar la placa, pero el inconveniente se presentó con la librería easyOCR. Por ejemplo, en la figura 29 el fallo se presentó en el primer carácter, dado que, se esperaba los caracteres “PBD 7342” pero la detección retorno “[PBD 7342”. Del mismo modo, en la figura 30 se puede observar que la cadena de texto devuelta fue “[XBY-907]”, mientras que la respuesta era “XBY 907”, sin embargo, en esta figura se puede evidenciar a su vez un fallo en la detección, pues, se detecta un cartel como una placa vehicular.

Figura 29. Detección del segundo fichero tipo imagen ingresado

Figura 30. Detección del quinto fichero tipo imagen ingresado

Respecto el método de clasificación para detecciones correctas en las pruebas con videos se continuo con el anterior descrito, es decir, si el modelo es capaz de detectar como minino una placa vehicular la salida será catalogada como correcta. En este caso, de los cinco videos ingresados, se obtuvo un total de tres salidas correctas y dos incorrectas.

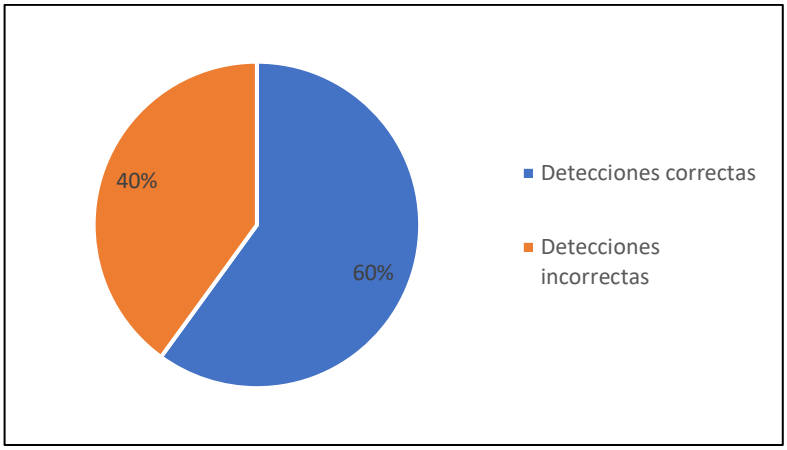
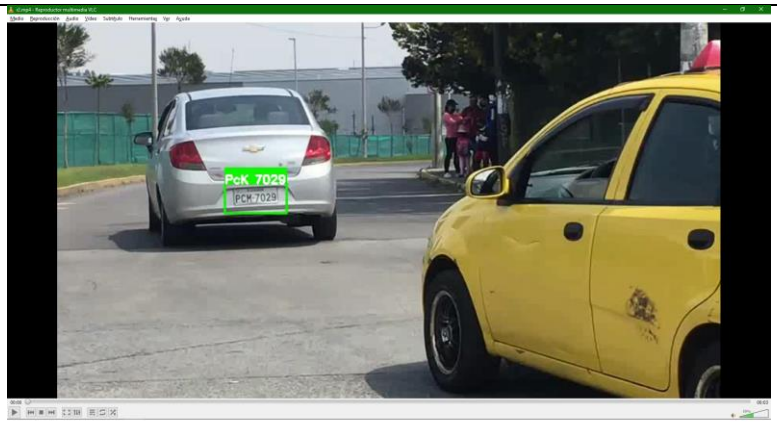
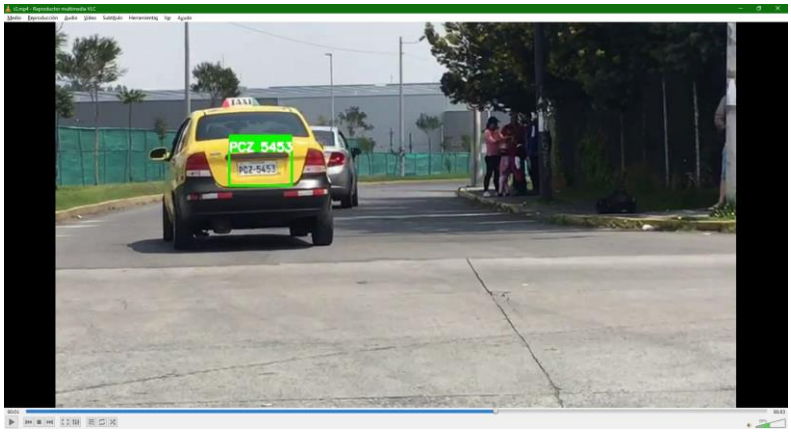


Figura 31. Porcentaje de detecciones correctas e incorrectas en videos

Tomando como ejemplo el primer video ingresado se puede observar dos vehículos, de los cuales el modelo logra detectar los caracteres de la placa vehicular del segundo automóvil, como se evidencia en la tabla 2.

Tabla 2. Detecciones en ficheros tipo video.

Número de Detección	Deteccion
Detección 1	
Detección 2	

5. Conclusiones y Recomendaciones

5.1. Conclusiones

El modelo YOLO en su séptima versión es capaz de ser entrenado para la detección de placas vehiculares, en este caso, con el conjunto de datos se obtuvo un mAP de 0.91 para un valor de confianza de 0.5.

La librería easyOCR se puedo implementar de forma adecuada para la extracción de caracteres de las placas vehiculares previamente detectadas por el modelo YOLO, tanto para ficheros de tipo imagen o video.

Los modelos de detección de objetos que usan métodos de aprendizaje profundo son una alternativa a tomar en consideración respecto a los métodos de detección de objetos tradicionales por visión artificial.

5.2. Recomendaciones

Es recomendable entrenar uno o más modelos que se encuentren en la categoría two-stage para comparar los resultados en relación a la categoría one-stage, a la que pertenece el modelo YOLO. Así también, implementar otra librería de reconocimiento óptico de caracteres con el fin de contrastar el rendimiento y los resultados de la librería easyOCR.

En caso de buscar implementar el modelo en producción se recomienda recopilar un mayor número de imágenes para el conjunto de datos tanto de entrenamiento como de prueba.

6. BIBLIOGRAFÍA

Amazon Web Services [AWS]. (s.f.). *¿Qué es el reconocimiento óptico de caracteres (OCR)?* Obtenido de AWS: https://aws.amazon.com/es/what-is/ocr/#:~:text=Optical%20Character%20Recognition%20(OCR)%20is,scan%20as%20an%20image%20file.

Ambika, P. (2020). Machine learning and deep learning algorithms on the Industrial Internet of Things (IIoT). En P. E. Pethuru Raj, *Advances in Computers* (Vol. 117, págs. 321-338). Elsevier,. doi:10.1016/bs.adcom.2019.10.007

Bermúdez, C. A. (2002). *Un sistema mediador para la integración de datos estructurados y semi-estructurados.* Obtenido de RUC: http://hdl.handle.net/2183/1129

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (23 de Abril de 2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. *arxiv*.

Chollet, F. (2017). *Deep learning with python.* New York, NY: Manning Publications.

Durán, M. A. (2021). *análisis, diseño e implementación de un sistema de control de ingreso de vehículos basado en visión artificial y reconocimiento de placas en el parqueadero de la universidad politécnica salesiana – sede cuenca.* Cuenca: Universidad Politécnica Salesiana.

Google. (s.f.). *Frequently Asked Questions: Colaboratory*. Obtenido de Google: https://research.google.com/colaboratory/faq.html

Gutiérrez, A. (s.f.). *Cursos: Bases de Datos.* Obtenido de Atlantic International University: https://www.aiu.edu/cursos/base%20de%20datos/pdf%20leccion%201/lecci%C3%B3n%201.pdf

IBM. (5 de Enero de 2022). *What Is Optical Character Recognition (OCR)?* Obtenido de IBM: https://www.ibm.com/cloud/blog/optical-character-recognition

International Business Machines [IMB]. (1 de Mayo de 2020). *Deep Learning*. Obtenido de IBM: https://www.ibm.com/cloud/learn/deep-learning

IPython. (s.f.). *Home*. Obtenido de IPython: https://ipython.org/

JaidedAI. (s.f.). *EasyOCR*. Obtenido de GitHub: https://github.com/JaidedAI/EasyOCR

Jupyter. (s.f.). *Projects*. Obtenido de Jupyter: https://docs.jupyter.org/en/latest/projects/architecture/content-architecture.html

O' Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Velasco, G., Krpalkova, L., . . . Walsh, J. (Abril de 2019). Deep learning vs. traditional computer vision. *In Science and information conference*, 128-144.

OpenCV. (s.f.). *Introduction*. Obtenido de OpenCV Documentation: https://docs.opencv.org/5.x/d1/dfb/intro.html

Poornima M. Chanal, M. S. (2021). Chapter 7 - Security and privacy in the internet of things: computational intelligent techniques-based approaches. En P. D. Siddhartha Bhattacharyya, *Recent Trends in Computational Intelligence Enabled Research* (págs. 111-127). Academic Press. doi:10.1016/B978-0-12-822844-9.00009-8

Prada Madrir, E. (2008). Los insumos invisibles de decisión: datos, información y conocimiento. *Anales de Documentación*(11), 183-196. Obtenido de https://www.redalyc.org/articulo.oa?id=63501110

PyTorch. (s.f.). *Home*. Obtenido de PyTorch: https://pytorch.org/

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. *arxiv*.

Shuai Mao, B. W. (2019). Opportunities and Challenges of Artificial Intelligence for Green Manufacturing in the Process Industry. *Engineering,, 5*(6), 995-1002. doi:10.1016/j.eng.2019.08.013

Snedaker, S., & Rima, C. (2014). Business Continuity and Disaster Recovery in Healthcare. En S. Snedaker, & C. Rima, *Business Continuity and Disaster Recovery Planning for IT Professionals (Second Edition)* (págs. 275-336). Syngress,. doi:10.1016/B978-0-12-410526-3.09976-1

Subasi, A. (2020). Chapter 3 - Machine learning techniques. En A. Subasi, *Practical Machine Learning for Data Analysis Using Python* (págs. 91-202). Academic Press. doi:10.1016/B978-0-12-821379-7.00003-5

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (6 de Julio de 2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arxiv*. doi:arXiv:2207.02696

Woolf, B. P. (2009). Chapter 7 - Machine Learning. En B. P. Woolf, *Building Intelligent Interactive Tutors* (págs. 221-297). Morgan Kaufmann. doi:10.1016/B978-0-12-373594-2.00007-1

Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object Detection With Deep Learning: A Review. *IEEE, 30*(11), 3212-3232. doi:10.1109/TNNLS.2018.2876865

Zou, Z., Shi, Z., Guo, Y., & Ye, J. (2019). Object Detection in 20 Years: A Survey. *arXiv*. doi:10.48550/arXiv.1905.05055