Universidad Nacional de Ingeniería

Facultad de Ingeniería Eléctrica y Electrónica

TESIS

Aplicación de algoritmos de aprendizaje profundo con análisis de metadata para identificación vehicular

Para obtener el Título Profesional de Ingeniero Electrónico

Elaborado por

Nicolás Francisco Silva Matute

0000-0002-2071-476X

Asesor

Dr. Ing. Carlos Celestino Medina Ramos

0000-0001-9747-3928

LIMA – PERÚ

2024

Citar/How to cite	Silva Matute [1]
Referencia/Reference Estilo/Style: IEEE (2020)	[1] N. Silva Matute, "*Aplicación de algoritmos de aprendizaje profundo con análisis de metadata para identificación vehicular*" [Tesis]. Lima (Perú): Universidad Nacional de Ingeniería, 2024.

Citar/How to cite	(Silva Matute, 2024)
Referencia/Reference Estilo/Style: APA (7ma ed.)	Silva Matute, N. (2024). *Aplicación de algoritmos de aprendizaje profundo con análisis de metadata para identificación vehicular*. [Tesis. Universidad Nacional de Ingeniería] Repositorio institucional Cybertesis UNI.

Dedicatoria

A mis padres y hermano por brindarme fortaleza y apoyo para conseguir mis objetivos, al formarme sabiendo que con esfuerzo y sacrificio se obtendrán grandes resultados.

Agradecimiento

Agradecimiento a toda la comunidad universitaria. en especial a mi asesor de tesis Dr. Ing. Carlos Medina Ramos por brindarme los conocimientos necesarios para encaminar esta tesis. Al Ing. Michael Vera por brindarme los implementos y el espacio para desarrollar la investigación. A mi compañero Luis Huingo por brindarme apoyo con sus conocimientos para el desarrollo del sistema.

Resumen

La presente investigación está orientada al desarrollo de un sistema de apoyo para la seguridad ciudadana, debido al elevado índice de criminalidad que se refleja en los incrementos de la violencia y la tasa de ocurrencia. En tal sentido, actualmente es necesario disponer de sistemas automatizados para resolver el problema, en particular la identificación vehicular mediante la extracción de *metadata* como son el color y los caracteres de la matrícula, para identificar presuntos delitos por sustracción y otros.

Para la obtención del sistema de detección se utilizan técnicas de aprendizaje profundo como el aprendizaje por transferencia para la localización del área de la placa y algoritmos de OCR (Reconocimiento óptico de caracteres) para detectar las letras y números de la matrícula. Para el color se aplica la técnica de procesamiento de imágenes con el objetivo de analizar el espacio de color HSV.

Como respuesta a la aplicación de los algoritmos y la metodología se obtuvieron resultados satisfactorios como se indica: para el algoritmo que permite la detección del área de la placa vehicular se obtuvo una precisión de 97.7%. Adicionalmente, con el algoritmo de OCR se realizó la identificación de caracteres con una precisión de 96.6%. Finalmente, para el análisis del espacio HSV y procesamiento de la imagen, se obtuvieron resultados significativos al demostrar ser un sistema que orienta correctamente la percepción del color del vehículo.

Palabras Clave: Aprendizaje profundo, Aprendizaje por transferencia, Placas vehiculares, Reconocimiento óptico de caracteres, Metadata.

Abstract

This research is oriented towards the development of a support system for citizen security, due to the high crime rate that is reflected in increases in violence and the rate of occurrence. In this sense, it is currently necessary to have automated systems to solve the problem, particularly vehicle identification through the extraction of *metadata* such as the color and characters of the number plate, to identify suspected crimes such as theft and others.

To obtain the detection system, deep learning techniques such as transfer learning are used to locate the area of the plate, and Optical Character Recognition (OCR) algorithms are employed to detect the letters and numbers of the number plate. For color, image processing techniques are applied to analyze the HSV color space.

In response to the application of the algorithms and methodology, satisfactory results were obtained as follows: for the algorithm that allows the detection of the area of the number plate, an accuracy of 97.7% was achieved. Additionally, with the OCR algorithm, character identification was performed with an accuracy of 96.6%. Finally, for the HSV space analysis and image processing, significant results were obtained, demonstrating a system that correctly identifies the vehicle's color perception.

Keywords: Deep Learning, Transfer Learning, Vehicle Plates, Optical Character Recognition, Metadata.

Tabla de Contenido

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Introducción

En la presente tesis se abordó el problema de la inseguridad ciudadana, para lo cual se propuso la aplicación de algoritmos de aprendizaje profundo. La inseguridad es un tema que afecta a los ciudadanos del mundo y genera gran preocupación, ya que al salir a las calles existe el constante peligro de ser víctima de la delincuencia. En particular, los vehículos son el medio más utilizado en la acción de estos delitos. Por esta razón, es de preocupación la casi nula existencia de sistemas de seguridad basados en inteligencia artificial capaces de realizar de forma efectiva tareas de identificación vehicular.

En los reportes que realiza el Instituto Nacional de Estadística e Informática (INEI) relacionados a los datos estadísticos de diversos crímenes en el Perú de forma anual, muestran como la cantidad de denuncias y delitos que se cometen aumentan cada año, este constante incremento revela la preocupante situación de inseguridad a la que los ciudadanos están expuestos (Bonett & Altamirano, 2023)(Carhuavilca Bonett & Peña Aldazabal, 2022). La cantidad de denuncias por robo de vehículo no ha sido ajena a este incremento anual, debido a que pueden ser sustraídos para luego cambiar o modificar la placa, con el objetivo de cometer actos delictivos. A esto se suma que la mayoría de los sistemas de control vehicular que se usan actualmente en el Perú se basan principalmente en una simple inspección visual realizada por el personal de seguridad, esto genera la posibilidad de incurrir en errores propios del ser humano y gran dificultad para reconocer placas clonadas o alteradas. En consecuencia, se genera la clara necesidad de plantear el uso de nuevas tecnologías que permitan brindar un sistema más eficiente y seguro para detectar estos vehículos.

Con el fin de abordar esta problemática, esta tesis se orientó al diseño de un sistema de seguridad capaz de obtener *metadata* de vehículos para que puedan ser identificados. Como parte de la obtención de *metadata* el sistema fue capaz de detectar, reconocer y almacenar los caracteres de las placas de vehículos en tiempo real, así como

el color. El sistema desarrollado es capaz de realizar diferentes tareas enfocadas en la seguridad.

Para el logro del objetivo de esta investigación se utilizó el lenguaje de programación Python, con el cual se consiguió utilizar diferentes técnicas de aprendizaje profundo, por ejemplo, el aprendizaje por transferencia o la creación de redes neuronales convolucionales (CNN) para el entrenamiento de modelos de visión por computador. Además, se desarrolló un algoritmo para la obtención de imágenes de placas vehiculares de Perú para la creación del conjunto de imágenes denominado UNI-LimaLP, con el que se entrenaron los modelos. Luego de analizar la eficiencia del modelo para cumplir con la tarea de detección se implementó el algoritmo que permite procesar las imágenes y obtener el área de la placa. Adicionalmente, el estudio se enfocó en la implementación de un sistema que optimice los resultados del algoritmo de reconocimiento óptico de caracteres (OCR) y un algoritmo que detecte el color de los vehiculos.

Finalmente, para realizar la comprobación del correcto funcionamiento del sistema se utilizaron videos, en tiempo real, de vehículos ingresando a la Universidad Nacional de Ingeniería tomados con una cámara ip, y la tarea de detección se realizó de modo remoto.

Con la intención de ordenar los puntos importantes de la investigación, se dividió esta tesis en cuatro capítulos. El capitulo I se enfoca en el detallar con amplitud la situación problemática, así como los antecedentes que expongan el estado del arte de la tesis. El capítulo II expone los aspectos teóricos necesarios para la comprensión de la tesis, sus variables y conceptos importantes. En el capitulo III se aborda el diseño del algoritmo, el estudio y entrenamiento de los modelos de detección de placas vehiculares, caracteres y color; la evaluación de estos, el esquema del funcionamiento del sistema, y la elección del dispositivo capaz de obtener imágenes en tiempo real. En el capitulo IV se muestran los resultados obtenidos a partir de las pruebas del sistema.

Capítulo I. Parte introductoria del trabajo

1.1 Generalidades

Dada la creciente inseguridad que aqueja a diversos países del mundo, la necesidad de desarrollar sistemas enfocados en la seguridad ciudadana se ha convertido en una necesidad prioritaria en las ciudades. Entre estos sistemas, la identificación de vehículos se ha convertido en una herramienta fundamental para enfrentar la delincuencia. La identificación de vehículos tiene muchas aplicaciones tales como: la búsqueda de coches robados, sistemas de peaje automático y la gestión de los coches que acceden a una zona o institución. Existen dos métodos para identificar un coche. Uno es el sistema de identificación por microondas en el que cada coche tiene su propia etiqueta de identificación (distinta de la matrícula) y un emisor envía una señal de microondas a la etiqueta de identificación para leer la información. El otro es un sistema de identificación enfocado en la matrícula, en el que se captura la imagen de un coche y se reconoce el número de matrícula mediante diversas técnicas de procesamiento de imágenes (Soh et al., 1994). Un ejemplo de aplicación se observa en el estudio "Prototipo de un sistema para controlar el acceso de vehículos y sus ocupantes al parqueadero de un conjunto residencial, implementando RFID y detección de huella digital ". En este los autores aplican la identificación por radio frecuencia (RFID) para desarrollar un prototipo de un sistema de control de acceso vehicular en el parqueadero de un conjunto residencial. Los autores destacan que la implementación del sistema reduce el tiempo de ingreso y salida del parqueadero, al tiempo que mejora la seguridad y control en la gestión del acceso vehicular (Campo Romero & Cruz Camelo, 2016). Por otro lado, la solución enfocada en el uso de la placa como medio de identificación sugiere el hecho de que existen datos propios de los vehículos que se obtienen de forma visual. De forma que el uso de procesamiento de imágenes para obtener estos datos es una solución viable. En la actualidad, nuevos

algoritmos se han desarrollado para dar paso a la visión por computador haciendo más confiables este tipo de sistemas.

1.2 Descripción del Problema de Investigación

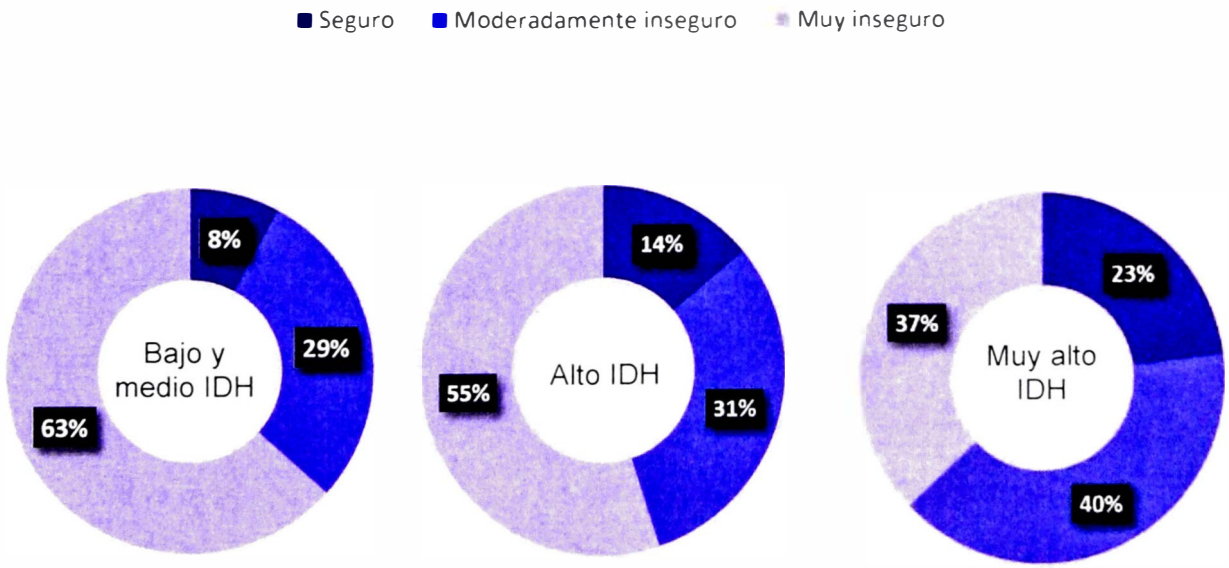
En esta sección se detalla la realidad problemática en lo concerniente a la detección de matrículas, además se expone el problema, los objetivos, hipótesis y variables.

1.2.1 Planteamiento del Problema

La inseguridad es un problema que a nivel mundial va en aumento, como revela el Programa de las Naciones Unidas para el Desarrollo (PNUD) en el reporte publicado en el año 2022. En este se introduce el concepto de Índice de Inseguridad Humana Percibida (I-IHP), se menciona que este índice se encuentra en constante aumento y se analiza su relación con el Índice de Desarrollo Humano (IDH). En la Figura 1 se observa que la inseguridad humana percibida es alta en todos los grupos del IDH, más de tres cuartas partes de la población se siente insegura, incluso en países con un IDH muy alto. Esta generalización de la inseguridad en todos los estratos sociales se traduce en que 6 de cada 7 personas han percibido esta sensación de inseguridad (Tapia et al., 2022).

Figura 1

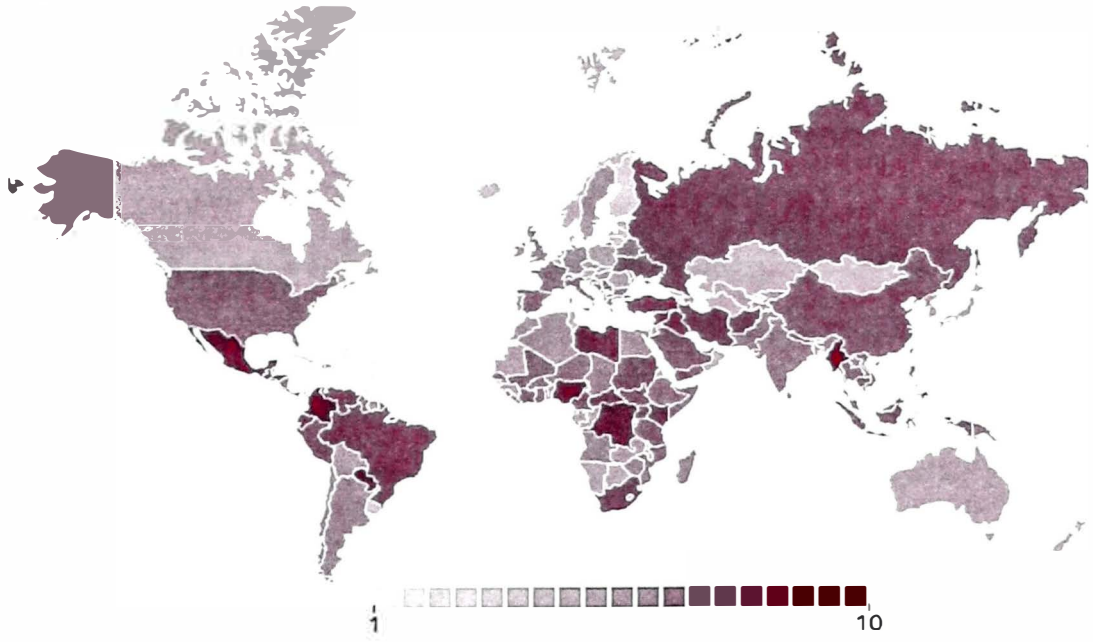
Gráficas de sensación de seguridad para diferentes grupos de Índice de Desarrollo Humano



Dado que la sensación de inseguridad proviene de diversos factores se debe analizar otro concepto, el índice de criminalidad. Este es un indicador estadístico empleado para evaluar y contrastar el grado de delincuencia y acciones delictivas en una determinada región geográfica durante un lapso temporal específico. La organización "The Global Initiative" se dedica a investigar la delincuencia organizada, de manera que se pueda plantear métodos de reducción de esta. Con este objetivo, cada año elabora un reporte relacionado al índice de criminalidad. Este índice permite clasificar cada país por el nivel de criminalidad, cabe resaltar que se basa principalmente en crímenes que se denuncian y por esta razón puede que en ciertos países no se vea reflejada la realidad.

Figura 2

Mapa del índice de criminalidad en el mundo, en el cual a más oscuro implica mayor índice



En la Figura 2 se presenta de forma gráfica los diferentes valores de índice de criminalidad en el mundo, se comprueba que la mayoría de los países tienen un tono por encima del centro de la escala, lo cual concuerda con la creciente sensación de inseguridad. Además, se observa que los tonos que pertenecen al rango de 5 a 10 se encuentran en Sudamérica, África y Asia (The Global Initiative, 2023b). Si se analiza la Tabla 1, la cual presenta el promedio del índice por regiones, resalta que América del Sur

se encuentra en tercer lugar de los índices más altos. Al considerar que en esta región muchos delitos no son denunciados, por lo que no forman parte de este índice, se comprende la real necesidad de desarrollar soluciones que enfrenten el problema de la inseguridad. En esta región se encuentra Perú, ocupando el puesto 32 del mundo y el sexto en América del Sur con un índice de 6.40.

Tabla 1

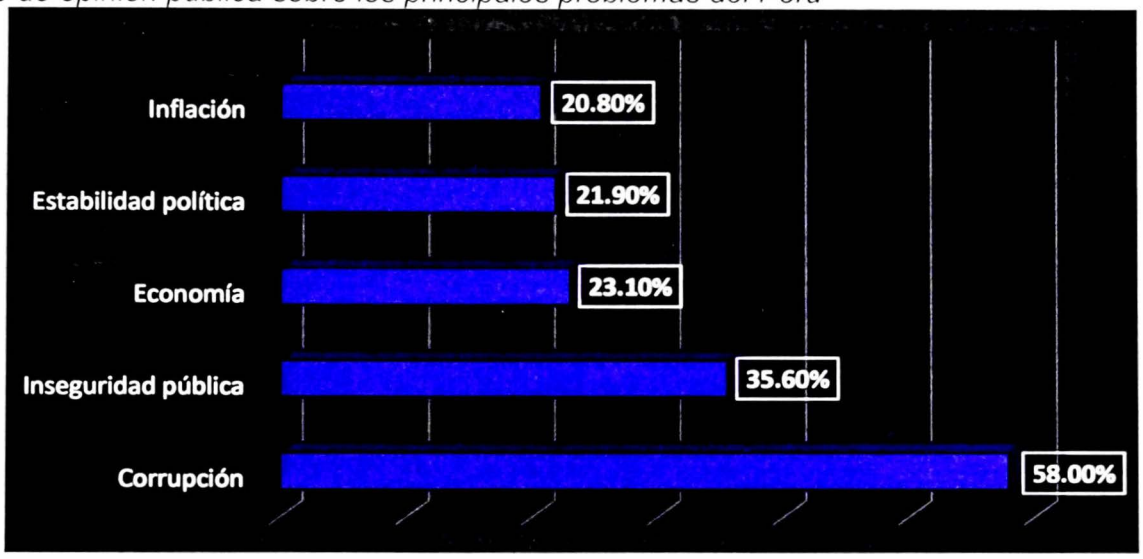
Regiones con el mayor índice de criminalidad promedio

Región	Índice de criminalidad
Centro América	6 28
Asia Occidental	6 02
América del Sur	5 94

Según el reporte enfocado en Perú, la presencia de redes criminales es un problema importante, particularmente en Lima y la Provincia Constitucional del Callao, donde se dedican a la micro comercialización de drogas, robos, extorsión, tráfico de personas y delitos contra la flora (The Global Initiative, 2023a). Por esta razón, la inseguridad es un serio problema que se debe abordar en el país y se sustenta aún más con la opinión de los ciudadanos sobre los principales problemas del Perú, presentada en la encuesta realizada por *Activa Research* a 800 ciudadanos peruanos en el año 2023. Los datos recopilados en esta encuesta se observan en la Figura 3, donde se exponen cinco problemas con su respectivo porcentaje según la cantidad de encuestados que mencionaron dicho problema como principal en el país. La inseguridad pública ocupa el segundo lugar dentro de estas problemáticas, solo por debajo de la corrupción y por encima de la economía por más de un 10 por ciento. Esto evidencia la necesidad de brindar mejores servicios de seguridad ciudadana, al ser considerada de vital importancia (ActivaResearch, 2023).

Figura 3

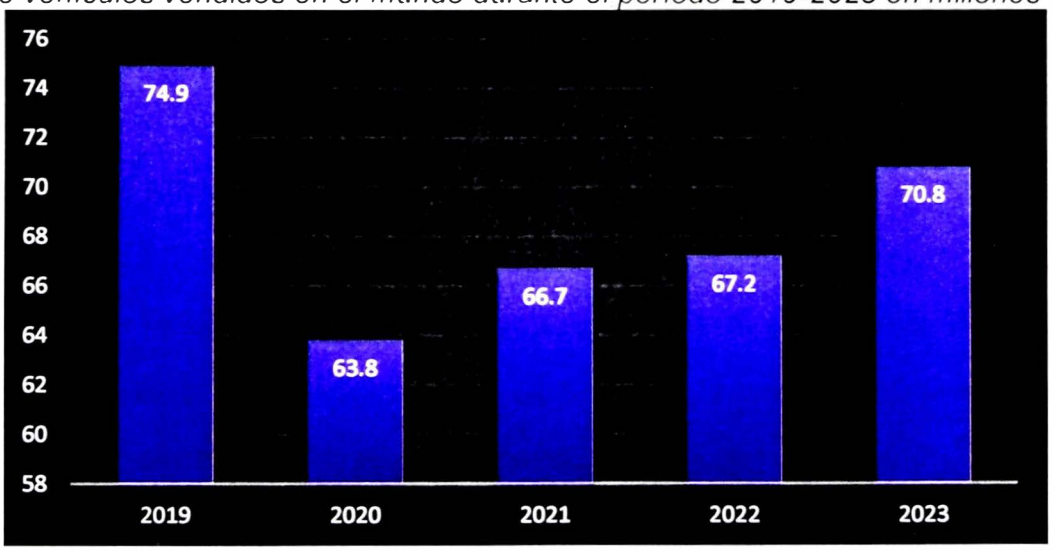
Gráfico de opinión pública sobre los principales problemas del Perú



En la Figura 4 se muestran los datos recopilados por Statista, con relación a la cantidad de vehículos vendidos del 2019 al 2022 con una proyección al 2023. En dicha gráfica se aprecia la existencia de un crecimiento en las ventas por año desde el año 2020.

Figura 4

Número de vehículos vendidos en el mundo durante el período 2019-2023 en millones



Las cifras de ventas globales de vehículos alcanzaron aproximadamente los 67.2 millones de unidades en el año 2022, en comparación con los 66.7 millones de unidades registrados en 2021. Durante los años 2020 y 2021, la industria experimentó una tendencia a la baja debido a la desaceleración económica mundial. Además, factores como la

pandemia de COVID-19 y el conflicto de Rusia con Ucrania contribuyeron a la escasez de semiconductores automotrices y a nuevas interrupciones en la cadena de suministro en el año 2022. A pesar de estos desafíos, se pronostica que las ventas continuarán aumentando en el año 2023, hasta alcanzar los 70,8 millones. Por lo tanto, incluso ante dificultades globales, la industria automotriz no deja de incrementar en sus ventas, esta tendencia ascendente plantea preocupaciones en cuanto a los impactos ambientales, el congestionamiento vial y; sobre todo la inseguridad vehicular (Statista Research Department, 2023). Además, según la Organización Internacional de Constructores de Automóviles (OICA), la producción de vehículos en el mundo; esto incluye autos, motos y otros, superó los 85 millones de unidades en 2022. Esto simboliza un incremento del 6% en comparación con el año anterior y se acerca a los niveles anteriores a la pandemia. Reforzando así la idea del constante aumento en la adquisición de automóviles nuevos en todo el mundo (Mena Roa & Statista Research Department, 2023).

La comercialización de vehículos nuevos en el Perú experimenta esta misma tendencia de acuerdo con cifras de la Superintendencia Nacional de los Registros Públicos (SUNARP), la cual menciona que durante la primera mitad del 2023 la venta de nuevos vehículos livianos fue de 86.763 unidades. Si se compara este valor con las ventas del mismo periodo del año anterior, se observa que es un 8,3% mayor (ASOCIACIÓN AUTOMOTRIZ DEL PERÚ, 2023). Por otro lado, para analizar la cantidad de vehículos que transitan por el país se tendrá que revisar el Índice Nacional de Flujo Vehicular. El Instituto Nacional de Estadística e Informática (INEI) se encarga de realizar el informe "Flujo Vehicular por Unidades de Peaje", en este se expone el registro de entrada y salida de vehículos que transitan por los peajes a nivel nacional, para vehículos ligeros y pesados. En el informe presentado el mes de Julio del 2023, el INEI menciona que existe un aumento del 2.8% con respecto al mes de Julio del 2022. Además, se registró un aumento del 10.6% en este indicador durante la primera mitad del año 2023 al compararlo con los datos del 2022 entre enero y julio (Carhuavilca Bonett & Escribano Pinaud, 2023). Estos incrementos

demuestran que el aumento en la cantidad de vehículos nuevos que se venden se traduce en mayor cantidad de vehículos transitando en las ciudades y alrededor del país.

Para el año 2023, la Asociación Automotriz del Perú (AAP) ha publicado informes mensuales que exponen la verdadera problemática en el campo automotriz del Perú. En la Tabla 2 se muestran los datos que corresponden al periodo de enero-abril (Asociación Automotriz del Perú, 2023).

Tabla 2

Cantidad de vehículos por tipo y su variación en función al período enero-abril del 2023

	Tipo de vehículo	Unidades	Variación con respecto a 2022
Vehículos livianos	Automóvil	12 756	-1 2%
	Camionetas	7 964	24 9%
	Todoterrenos	26 008	14 5%
	Furgonetas	11 211	2 9%
Vehículos pesados	Camiones	4 576	-11 1%
	Ómnibus	639	-28 1%
Vehículos menores	Motos	77 778	-13 0%
	Trimotos	34 262	-11 7%

Se definen tres categorías de vehículos: livianos, pesados y menores. En el caso de los vehículos livianos, se observa un incremento generalizado, a excepción de los automóviles, los cuales muestran una variación negativa indicativa de una disminución en su demanda. Destaca que las camionetas presentan el mayor crecimiento en este segmento, estas cifras pueden ser indicativas de un cambio en las preferencias de transporte y un posible impacto en la movilidad y seguridad vial. Por otro lado, los ómnibus también muestran una variación negativa significativa del -28.1%, lo cual indica una disminución en la adquisición de este tipo de vehículos destinados al transporte de pasajeros. Precisamente este dato es muy preocupante, porque si se reduce la cantidad de vehículos para el transporte masivo y continúa aumentando los vehículos menores, puede llevar a una mayor congestión, lo que dificulta el control vehicular. Por lo tanto, resulta claro que los ciudadanos peruanos actualmente hacen gran uso de vehículos

livianos para realizar diferentes actividades diarias y, por ende, brindar seguridad a estos vehículos es de vital importancia para que los peruanos gocen de buena calidad de vida.

La proliferación de vehículos en Perú puede generar una serie de desafíos en términos de seguridad, dado que la gran cantidad de automóviles y otros vehículos en circulación brinda mayores oportunidades para robar los vehículos o realizar delitos en estos, como robos, asaltos y secuestros. Esta idea se sustenta en los datos presentados por el INEI en el informe técnico "Estadística de la Criminalidad, Seguridad Ciudadana y Violencia", en el cual son recopilados datos estadísticos de diversos crímenes en el país. En el caso de robo de vehículos, la cantidad de vehículos denunciados asciende a 14,315 durante la primera mitad del año 2023 (Bonett & Altamirano, 2023), para este mismo periodo se tenía 11,467 denuncias en el año 2022, siendo un aumento de un 5% (Carhuavilca Bonett & Peña Aldazabal, 2022). Además, la cantidad de delitos contra el patrimonio representa un 68.8% del total de denuncias durante el periodo antes mencionado, de los cuales muchos delitos son cometidos utilizando vehículos previamente robados. En consecuencia, se concluye que los sistemas que actualmente se utilizan para la seguridad vehicular, tanto para evitar robos de vehículos como para alertar cuando estos vehículos robados transitan por las calles, no son eficientes ni responden a los incrementos descritos. Si bien en muchos lugares se está aplicando nuevas tecnologías para realizar estas tareas, la mayoría de los sistemas de seguridad cuentan únicamente con una persona encargada de realizar inspecciones visuales sobre las placas de los vehículos, lo cual puede incurrir en fraudes o errores. Por lo tanto, la necesidad de explorar nuevos métodos que mejoren estos sistemas debe ser una prioridad.

La identificación vehicular automática (IVA) en tiempo real es una importante herramienta tecnológica en el ámbito de la seguridad vehicular, ya que puede ser utilizado para la identificación de vehículos robados, la investigación de crímenes o la emisión de multas. Los sistemas IVA son diseñados para identificar las características únicas de cada vehículo en un tiempo y lugar específico (Bernstein & Kanaan, 1993). Sin embargo, existen

limitaciones en la aplicación de esta tecnología, debido a que uno de los identificadores principales son las placas vehiculares; y en el Perú presentan ciertas particularidades que dificultan su detección, tales como el uso de placas con diseños variados y la presencia de suciedad o daños en las placas. En este contexto, surge la necesidad de diseñar un sistema de seguridad vehicular capaz de obtener estas características o *metadata*, utilizando técnicas de aprendizaje profundo y procesamiento de imágenes para mejorar la eficiencia y precisión en la detección.

1.2.2 Problema General

1. ¿De qué manera se relacionan la aplicación de algoritmos de aprendizaje profundo con análisis de *metadata* y la identificación vehicular?

1.2.3 Problemas Específicos

1. ¿Qué relación existe entre la identificación vehicular y las arquitecturas de aprendizaje profundo basadas en redes neuronales convolucionales?
2. ¿De qué manera la implementación de algoritmos de optimización aplicados a redes neuronales convolucionales es eficaz para la identificación vehicular?
3. ¿De qué modo influye el reconocimiento de caracteres en la identificación vehicular?
4. ¿En qué medida se puede relacionar la evaluación de la *metadata* con la identificación vehicular?

1.2.4 Variables

1. Variable Dependiente: Identificación vehicular
2. Variable Independiente: Algoritmos de aprendizaje profundo con análisis de *metadata*

1.3 Objetivos del Estudio

1.3.1 Objetivo General

1. Implementar algoritmos de aprendizaje profundo con análisis de *metadata* capaz de permitir la identificación vehicular.

1.3.2 Objetivos Específicos

1. Implementar arquitecturas de aprendizaje profundo basadas en redes neuronales convolucionales para identificación vehicular.
2. Implementar algoritmos de optimización aplicados a redes neuronales convolucionales para identificación vehicular.
3. Implementar algoritmos de reconocimiento de caracteres para identificación vehicular.
4. Evaluar la influencia de la *metadata* y algoritmos para la identificación vehicular.

1.4 Hipótesis del Estudio

1.4.1 Hipótesis General

1. Con la aplicación de algoritmos de aprendizaje profundo con análisis de *metadata* se realizará la identificación vehicular.

1.4.2 Hipótesis Específicas

1. La implementación de arquitecturas de aprendizaje profundo basadas en redes neuronales convolucionales es eficaz en la identificación vehicular.
2. La implementación de algoritmos de optimización aplicados a redes neuronales convolucionales es relevante para la identificación vehicular.
3. La aplicación de un correcto algoritmo de reconocimiento de caracteres es significativa para la identificación vehicular.
4. La evaluación de la *metadata* y algoritmos resulta eficaz para la identificación vehicular.

1.5 Matriz de operacionalización de variables de la investigación: "Aplicación de algoritmos de aprendizaje profundo con análisis de *metadata* para identificación vehicular"

Tabla 3

Dimensiones e indicadores de las variables estudiadas

Variable	Dimensiones	Indicadores
Algoritmos de aprendizaje profundo con análisis de metadata	D1 Arquitectura de aprendizaje profundo basada en redes neuronales convolucionales	Precisión
		Rendimiento computacional
	D2 Algoritmos de optimización	Tiempo de respuesta
	D3 Reconocimiento de caracteres	Precisión
	D4 Metadata	Color y matricula
Identificación vehicular	D1 Formato de numeración	Cantidad de caracteres en la placa y distribución
	D2 Matricula	Formatos
	D3 Color	Valor de pixeles en formato RGB
	D4 Metadata Sunarp	Color
		Marca
		Datos del propietario
		Otros

1.6 Antecedentes Investigativos

En esta sección se consideran tesis e investigaciones que presentan cierta similitud con los objetivos y algoritmos del presente trabajo.

1.6.1 Antecedentes Nacionales

"Reconocimiento automático de placas de rodaje utilizando una red neuronal convolucional para el ingreso de vehículos en la universidad Ricardo Palma" (Ramirez Mejia & Tito Apaza, 2020)

En esta tesis los autores proponen el diseño de un sistema basado en redes neuronales convolucionales, así como una aplicación móvil para el reconocimiento de placas vehiculares en la entrada de la universidad Ricardo Palma. Para este sistema utilizaron el lenguaje de programación Matlab con el cual se realizó el entrenamiento

de diferentes modelos y fotografiaron placas vehiculares de vehículos en la universidad para crear sus imágenes de entrenamiento.

En esta investigación los autores concluyen que mediante el uso del lenguaje de programación Matlab y su *toolbox* de *Deep Learning* se pueden entrenar modelos de detección, consiguiendo un porcentaje de efectividad del 40%, 75% y 90%. Además, lograron integrar una aplicación diseñada en Matlab para la visualización de las imágenes de placas y su detección.

"Desarrollo de un algoritmo para la localización automática de placas vehiculares peruanas usando técnicas de procesamiento de imágenes" (Salazar Márquez, 2014)

En esta investigación se hace uso de técnicas de procesamiento por imágenes para enfrentar los desafíos asociados al monitoreo delitos de tránsito, por ejemplo, hurto de automóviles y accidentes. Mencionan que uno de los principales obstáculos para implementar nuevas tecnologías en este campo, es la gran cantidad de vehículos en circulación, lo que dificulta el monitoreo de todos los vehículos infractores. Por lo cual, proponen el desarrollo de un algoritmo de localización de placas en una imagen, es decir, ser capaz de detectar el área correspondiente a la matrícula. Utilizan únicamente métodos de procesamiento de imágenes.

Obtenidos los resultados, se encuentra factible implementar un sistema de detección de placas de vehículos con mayor flexibilidad utilizando técnicas de localización. La efectividad del método desarrollado se ve influenciada por las condiciones de la imagen, la antigüedad de la matrícula y su resolución. Asimismo, se propone mejorar la tasa de falsos positivos mediante la introducción de una nueva condición que verifique que el candidato a ser placa posea letras y números, esto se consigue al segmentar cada carácter en las regiones de la imagen. Esta mejora permitiría obtener con mayor precisión la placa vehicular al eliminar otros candidatos.

"Sistema de monitoreo para la detección automática de placas de vehículos en Lima Metropolitana utilizando redes neuronales" (Montero Calderon & Ponce Diaz, 2022)

Ante la problemática existente para la policía nacional de realizar la detección de vehículos robados de forma eficaz, esta tesis presenta como solución, un sistema de monitoreo que detecta automáticamente matrículas de autos que se reportan como robados o que hayan cometido infracciones. Por lo que se utiliza el modelo YOLO v4 para detectar las placas y como algoritmo de reconocimiento de caracteres, se usa Pytesseract.

Por lo expuesto se sugiere utilizar Yolov4 para obtener la placa vehicular y luego detectar la posición de los caracteres resulta eficaz para que el algoritmo Pytesseract reconozca con facilidad cada carácter con precisiones cercanas al 90%

1.6.2 Antecedentes Internacionales

"A Smart Access Control for Restricted Buildings Using Vehicle Number Plates Recognition System" (Kahie et al., 2021)

Los autores de esta investigación desarrollaron un prototipo de sistema automático de puertas para acceso a instituciones privadas. Este sistema de puertas permite el acceso de vehículos que se encuentran en una base de datos, para lograr esto, utilizan una cámara de *raspberry* para capturar las imágenes y usando preprocesamiento de imágenes se obtuvieron las placas vehiculares. Luego se plantea la segmentación y detección de caracteres con algoritmos de OCR.

En esta investigación concluyen que mediante el uso de este sistema realiza una correcta detección de los caracteres de las placas y al comparar con una base de datos se logra un control eficiente de los vehículos permitidos. Una limitación importante es que, al usar únicamente procesamiento de imágenes para detectar las placas, el sistema es propenso a errores ante variaciones de color, iluminación y formas de las placas.

"Parking System License Plate Detection based on Convolution Neural Networks GPU Optimization" (Elkhatib et al., 2021)

En esta investigación se hace uso de redes neuronales convolucionales con el objetivo de detectar las placas vehiculares y sus caracteres, además, se propone un método de optimización del sistema mediante el uso de GPU.

Analizando los resultados se muestra que, mediante el uso de redes neuronales convolucionales y una correcta optimización de la GPU, se consigue mejorar la eficiencia del sistema de detección, pues se reduce la cantidad de memoria utilizada y el tiempo en la que los resultados se obtienen. Cabe mencionar que esta reducción conlleva también una reducción en la precisión, por lo que se debe analizar si esta pérdida de precisión afecta de forma significativa a los resultados.

"Detección e identificación de características de vehículos utilizando algoritmos de machine learning" (Nicanor Mariotti et al., 2020)

En esta tesis los autores abordan la implementación de un sistema modular para realizar la tarea de reconocimiento automático de placas vehiculares, para lo cual desarrollaron módulos individuales para realizar ciertas etapas de la detección. Utilizaron transfer learning en redes neuronales convolucionales con un *dataset* de dominio público para realizar la detección de las placas, además utilizaron una red recurrente-convolucional y el software CRAFT para la obtención de los caracteres.

En esta investigación se describe que la escasez de imágenes de placas propias del país en el que se implementó esta tesis fue un problema importante para obtener mejores resultados. Concluyen que el sistema planteado incluye variaciones en la precisión para diferentes iluminaciones y climas, sin embargo, de manera general se obtienen buenos valores de *recall*.

"Toward End-to-End Car License Plate Detection and Recognition With Deep Neural Networks" (Li et al., 2019)

En este caso los autores buscaron la posibilidad de diseñar y entrenar una red neuronal capaz de realizar las tareas de extracción de características importantes de la imagen. y con estas realizar la detección de la placa y en esta misma red realizar la detección de los caracteres. Por lo tanto, diseñaron una arquitectura que incluyen redes neuronales convolucionales y otros métodos de extracción de características, que en conjunto realizan las mismas tareas que los clásicos diseños por etapas.

En esta investigación concluyen que al utilizar una arquitectura en la que se realiza toda la extracción de características y la detección tanto de placas como de caracteres, se ve reducido el tiempo y memoria que el sistema utiliza, manteniendo altas precisiones. Esta forma de diseñar el sistema evita el uso de un algoritmo de recorte para la matrícula, así como la etapa para segmentar caracteres.

1.7 Marco filosófico

En este trabajo se plantea una solución tecnológica ante una problemática que afecta de forma directa a los ciudadanos y su seguridad, por lo que se tomará como base filosófica el humanismo.

1.7.1 El humanismo

El concepto de "Humanismo" se refiere generalmente a la creencia que considera al ser humano como centro y fuente del sentido y valor. Además, considerando el planteamiento de Protágoras de Abdera, las necesidades del ser humano deber ser priorizadas (Mäkelä, 2017).

Por otro lado, la *American Humanist Association* plantea como definición:

El humanismo es una filosofía fundamentada en la razón científica, enraizada en la expresión artística y guiada por la compasión. Defiende la valía de todo individuo humano, respaldando la ampliación de la autonomía personal en sintonía con la responsabilidad social y medioambiental (American Humanist Association, 2023).

Se concluye que el humanismo promueve la participación democrática y la difusión de una sociedad inclusiva, que defiende la integridad humana y la equidad en la sociedad. Rechaza la creencia de lo sobrenatural, al identificar a las personas como una pieza dentro de la naturaleza y plantea que el origen de los principios son la experiencia y cultura humana, sin importar si son por religión, ética o política. Define que la abstracción teológica o ideológica es menos relevante que las necesidades e intereses del ser humano, con relación a como se definen los objetivos de vida, y afirma que la sociedad tiene que aceptar que cada uno es responsable de su destino.

1.7.2 Relación del uso de la tecnología y el humanismo

En este trabajo se plantea un sistema que brinde mayor seguridad y permita mejorar el índice de calidad de vida, en concordancia con este concepto es importante analizar el vínculo que existe entre el humanismo y la aplicación de la tecnología.

Según Mäkelä, gracias a diversos estudios, cada vez se es más consciente de que las tecnologías usadas por las redes sociales y medios de comunicación son instrumentos de alienación humana. Los seres humanos están siendo sistemáticamente desconectados de la experiencia inmediata del mundo (Mäkelä, 2017). Continuando con esta idea, Simondon plantea una perspectiva intrigante acerca de la liberación de los seres humanos, sosteniendo que es necesario emancipar también a las máquinas para lograr dicha liberación. Argumenta que las máquinas tienen el potencial de inducir una nueva forma de alienación y propone un enfoque renovado del Enciclopedismo como respuesta a esta problemática. Al hacer referencia al propósito histórico del Enciclopedismo, que buscaba liberar a los seres humanos de formas específicas de alienación, Simondon destaca la importancia de abordar la alienación inducida por las máquinas en la sociedad actual. Sugiere que es inherente al desarrollo humano que cualquier invento, ya sea ético, técnico o científico, diseñado inicialmente para liberar y redescubrir a la humanidad, tiene el potencial de volverse finalmente en contra de su propósito y esclavizar a los humanos (Simondon, 1989).

Dado el aparente contraste entre los valores humanistas tradicionales y la omnipresencia distractora de la tecnología en general, ¿cómo podrían conciliarse? Además de las preocupaciones que suscitan los medios de comunicación sociales y de masas, ¿cómo podría ponerse la tecnología al servicio de las necesidades humanas esenciales de refugio? Estas cuestiones se abordarán desde dos perspectivas: humanismo y tecnología (Mäkelä, 2017).

Al considerar la dimensión científica de la Ilustración, la noción de razón universal se fundamenta en la necesidad matemática desde la época de Descartes. En este contexto, la perspectiva humanista que se alinea con esta razón universal, percibiéndola como la esencia de la humanidad, tiende a equiparar la tecnología con una mera puesta en práctica del conocimiento científico (Barthélémy, 2010). Según este punto de vista, el fin último del conocimiento científico es afirmar el dominio y el control sobre la naturaleza, como afirmaba Descartes: "hacernos dueños y señores de la naturaleza" (Descartes, 1970). Este planteamiento humanista sobre las innovaciones tecnológicas tiene sus raíces en la obra del filósofo español José Ortega y Gasset, quien argumentó que la tecnología es la creación de nuevas posibilidades que no se encuentran de forma natural, de forma que estas faciliten realizar diversos planes de vida. Además, debe estar subordinada a los valores humanos para ser verdaderamente beneficiosa (Medina, 1995).

Al aplicar esta idea al contexto de la seguridad, se busca encontrar soluciones tecnológicas que sean coherentes con los valores y principios humanos que se tienen definidos en la sociedad actual, para que se genere un verdadero impacto positivo. Asimismo, en el trabajo de Morozov el autor señala que la tecnología no es capaz de solucionar los problemas sociales, por sí misma; sin embargo, resulta ser una herramienta muy útil para una sociedad consciente y humanista (Morozov, 2013). Esto indica que los nuevos conocimientos científicos o innovaciones tecnológicas no brindarán soluciones para la sociedad si se deja de lado la importancia que se le debe dar a la vida y a los valores.

Mediante este análisis, se concluye que la relación entre el humanismo y la tecnología en el contexto de la seguridad es fundamental para garantizar soluciones efectivas y beneficiosas para la sociedad. Es necesario entender que la tecnología no puede ser vista como un fin en sí misma, sino como una herramienta poderosa que debe estar al servicio de los valores y principios humanos. El enfoque humanista invita a la reflexión sobre la importancia de considerar el impacto social y ético de las innovaciones tecnológicas, así como asegurar que estén en armonía con las necesidades y aspiraciones de las personas. Solo a través de esta perspectiva consciente y humanista se puede encontrar soluciones tecnológicas que promuevan la seguridad, mejoren la calidad de vida y promuevan de felicidad en la sociedad.

Capítulo II. Marco teórico y conceptual

En esta parte de la tesis se describen los conceptos que se requieren para el correcto entendimiento de las variables de la investigación.

2.1 Marco teórico

2.1.1 Fundamentos teóricos para la identificación vehicular

En esta sección se presenta la base teórica para la comprensión de la principal tarea para la que se diseñó este sistema, por esta razón se analizan los elementos que permiten realizar la identificación de vehículos.

1. Metadatos. Los metadatos, comúnmente conocidos como "datos sobre datos", son una parte crucial de la gestión de información y la operación de sistemas informáticos. Es importante destacar que, aunque un metadato puede tomar una variedad de formas, su función principal es proporcionar descripciones detalladas de otros datos. En este sentido, cuando se utiliza para un propósito descriptivo específico, un dato se considera un metadato. Para comprender la naturaleza de los metadatos, entonces, es esencial utilizar este enfoque de descripción y contextualización (Bargmeyer & Gillman, 2000). La información descriptiva que proporcionan puede ser sobre el contenido, calidad, productor, propósito, organización, entre otros; sobre un objeto que podría ser desde un objeto físico a un reporte digital (Guptill, 1999). Estas características sobre los objetos se presentan como datos estructurados y son muy útiles para realizar tareas de búsqueda, organización, identificación, clasificación o interpretación de información de una manera eficiente (Lytras & Sicilia, 2007).

En el área de los sistemas informáticos, el rol de los metadatos es indispensable para almacenar, procesar e intercambiar datos digitales, debido a la necesidad de que sus propiedades requieren una descripción representada de manera explícita. Dicha descripción permite a los sistemas y usuarios definir la importancia de un dato en cierta situación o interpretar el contenido de los datos. Precisamente los metadatos tienen el rol

de brindar estas descripciones, aumentando el valor de los datos e incrementando las posibilidades de uso (Kogalovsky, 2013).

Actualmente, en el ámbito de los productos alimenticios, se sabe que los consumidores requieren conocer ciertas características antes de adquirir un producto. Razón por la cual los metadatos como el nombre, peso, ingredientes, contenido, color, alertas o imágenes son importantes herramientas para presentar una descripción del producto y facilita que el usuario decida comprarlo (Gundimeda et al., 2019). En la Figura 5 se observa un ejemplo de metadata, donde se muestran de forma clara diversos aspectos del producto; como el nombre, sabor, cantidad e información nutricional del producto.

Figura 5

Figura que ilustra la aplicación de la metadata para mostrar características de un producto.

Otro ejemplo de metadatos se encuentra en los artículos científicos, donde el título, autor, resumen y bibliografía serán metadatos simples; el autor a su vez presenta metadatos como su correo electrónico o su filiación, generando relaciones y estructuras entre metadatos (Tkaczyk et al., 2015). Se comprende entonces que los vehículos tienen sus propios metadatos, un ejemplo de estructura de metadatos para este caso se presenta en la Tabla 4. Esta estructura de metadatos contiene diversos atributos como la matrícula, la hora de ingreso, color, ubicación y tipo de vehículo. Además, se define el tipo de dato que es y su descripción (X. Zhao et al., 2014).

Tabla 4

Estructura de los metadatos para un vehículo

Nombre del atributo	Tipo	Descripción
location	String	Lugar en el que se encuentra la cámara
enter_time	Long	Marca de tiempo cuando ingresa el objeto
exit_time	Long	Marca de tiempo cuando sale el objeto
color	Short	Color del objeto, se definen los posibles colores
license_plate_number	String	Caracteres que conforman la placa de un vehículo
model	String	Clasificación del modelo del vehículo detectado
license_plate_image	Byte	Imagen de la matrícula

En esta investigación se hará uso de la placa vehicular y del color del vehículo como los metadatos que permitirán identificar un vehículo, así como otros metadatos para definir tiempo y lugar.

2. Placa única de rodaje. En el país el encargado de brindar el reglamento y las especificaciones que permitan mantener un formato entre las placas vehiculares es el Ministerio de Transportes y Comunicaciones (MTC). Esto queda estipulado en el artículo 32 de la Ley N° 27181, en el cual se indica también, "que los vehículos que circulen por cualquier vía pública tienen la obligación de exhibir la denominada placa única de rodaje, cuya clasificación, características y el procedimiento para su obtención, son establecidos por el MTC" (Ley N° 27181, Ley General de Transporte y Tránsito Terrestre, 2020).

El elemento que permite realizar la identificación única de cada vehículo es también llamado placa única nacional. La placa de rodaje se elabora a partir del material aluminio y está conformado por un par de placas. Estas placas se colocan en la parte delantera y trasera del vehículo, en esta se indican los caracteres que identifican a la placa. Está compuesta por letras y números que se dividen en grupos de tres por una raya. La pintura de fondo es de color blanca e incluirán medidas de seguridad que dificulten su falsificación (Aguilar Anaya, 2022). En esta definición se enfatiza el hecho de que, según el reglamento de placa única nacional de rodaje, los vehículos únicamente deben tener una placa en la

parte delantera y otra en la parte trasera. Esto no suele ser respetado y se observa que muchos vehículos poseen más de una placa, ya sea pintada o como un adhesivo.

3. Elementos de identificación de las placas. Según el reglamento de placa única de rodaje en el decreto supremo 017-2008, el artículo N°12 indica el contenido que deben tener las placas metálicas de los vehículos (Decreto Supremo N° 017-2008-MTC, Reglamento de Placa Única Nacional de Rodaje, 2008).

Los elementos que se podrán encontrar se visualizan en la Figura 6:

a. Palabra "PERU". La cual debe ser escrita en letras mayúsculas y se encontrará en la parte superior centrada de la placa, además se imprime en alto relieve.

b. Figura de la bandera peruana. Esta debe encontrarse en la parte de arriba y alineada a la izquierda de la placa. Además, debe estar completamente integrada a la lámina de la placa y debe ser imposible de remover mediante ningún medio químico o físico sin dañar el sistema retroreflectivo de la matrícula.

c. Holograma de seguridad. Se sitúa en la esquina superior derecha de la matrícula, portando impreso el número de registro de la matrícula. Su aplicación se realiza en condiciones de baja temperatura y debe ser inamovible sin provocar daños en la matrícula. Además, se utiliza un holograma similar como una tercera placa colocada en el parabrisas, este permite realizar la identificación del vehículo con la información que contiene.

d. Código de Seguridad. Se emplea un número de seguridad grabado mediante tecnología láser, el cual debe ser exclusivo para distintas matrículas. Únicamente se puede eliminar este código de la matrícula mediante métodos que generen daños irreparables, y debe mantenerse a la vista siempre.

e. *Marca de agua.* Se debe implementar una marca de agua incluida dentro de la lámina retroreflectante, la cual no debe estar impresa sobre la superficie de la misma. Este sello debe ser elaborado de manera que falsificarla sea una tarea difícil

o imposible, por lo tanto, no debe ser factible su eliminación mediante métodos que no generen daños parciales o totales al sistema retroreflectante de la matrícula.

f. ***Placa vehicular.*** El número que identifica a la matrícula está constituido por mayúsculas y números, escritos de forma que sobresalen y resalten. Debe ser colocada en el centro de toda la matrícula.

Precisamente este último elemento es en el que se enfocó esta tesis, pues es la que se identifica de forma más clara y a mayor distancia por su tamaño, además en el reglamento antes mencionado, en el artículo N°14, se estipula que:

"Se encuentra prohibido efectuar añadidos o alteraciones al contenido de la Placa Única Nacional de Rodaje, así como colocar cualquier elemento físico o luminoso que imposibilite o dificulte su visualización o su lectura a través de medios electrónicos, computarizados u otro tipo de mecanismos tecnológicos que permitan verificar la comisión de las infracciones de tránsito" (Decreto Supremo N° 017-2008-MTC, Reglamento de Placa Única Nacional de Rodaje, 2008).

En la Figura 6 se presenta una imagen de referencia de los objetos que se encuentran en una placa de rodaje.

Figura 6

Elementos que conforman la placa única de rodaje

4. **Tipos de placas.** Según el artículo N°8 se pueden definir diferentes tipos de Placa Única Nacional de Rodaje, estos son las placas ordinarias, placas de vehículos menores, placas especiales, placas de gracia, de exhibición y rotativa (Decreto Supremo N° 017-2008-MTC, Reglamento de Placa Única Nacional de Rodaje, 2008).

a. ***Placas Ordinarias.*** Estas son las placas utilizadas para reconocer de manera única a los vehículos que transitan por las carreteras y calles públicas.

b. ***Placas Especiales.*** Cumplen la función de reconocer los automóviles de menor o mayor tamaño, tanto ligeros como pesados, los cuales, durante su desplazamiento por las vías públicas, desempeñan labores o tareas en beneficio de la comunidad para mantener el orden y seguridad, también en situaciones excepcionales que imposibiliten la posibilidad de utilizar la matrícula ordinaria. Por ejemplo, en el caso de vehículos destinados a servicios de emergencia, como bomberos, ambulancias o patrullas municipales.

c. ***Placa de gracia.*** Se brinda a los vehículos de funcionarios diplomáticos, representantes residentes, altos funcionarios y personal internacional, de forma que se cumpla con la norma de cortesía. Esta placa permite reconocer a dichos vehículos como propiedad de los mencionados funcionarios.

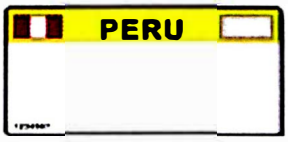
d. ***Placa de exhibición.*** Los vehículos recién fabricados que necesiten transitar por carreteras o calles previo a ser registrados oficialmente en el Registro de Propiedad Vehicular serán identificados con este tipo de matrículas.

En esta clasificación se aprecia la variedad de placas que existen, sin embargo, para el estudio de esta tesis se consideró el análisis de las placas ordinarias y especiales. En particular, dentro de las del tipo ordinarias se presentan las diferentes categorías que existen para los vehículos menores, livianos y pesados.

Según el reglamento de clasificación de vehículos brindado por el gobierno peruano (Ministerio de Transporte y Comunicaciones, 2003), los vehículos pertenecerán a cierta categoría dependiendo de sus características y funciones. Además, a cada categoría le corresponde ciertas características para el color de la placa, como presenta la AAP en la Tabla 5 (Asociación Automotriz del Perú, 2016b).

Tabla 5

Características de placas vehiculares según su categoría

Categoría	Descripción	Tipo de Placas de Rodaje
Placas para Vehiculos Menores – L	Categorias L1, L2 y L3 Vehiculos que poseen una cantidad de llantas inferior a cuatro, con medidas máximas de 50 cm^3 y con una rapidez máxima de 50 $\frac{km}{hora}$	Placa Azul en su totalidad 
	Categoria L5 Vehiculos que poseen tres llantas con una delante y dos detrás, con medidas mayores a 50 cm^3, con una rapidez mayor a 50 $\frac{km}{hora}$ y valor de carga inferior a 1 tonelada.	Placa Azul con una linea superior en Amarillo 
Placas para Vehiculos Livianos y Pesados – M	Categoria M1 Vehiculos particulares con espacio para un máximo de 8 personas, sin incluir al conductor.	Placa Blanca en su totalidad 
	Categoria M1 Vehiculo destinado a transportar personas, con una capacidad máxima de 8 personas, sin contar con el conductor, como un taxi	Placa Blanca con una linea superior en color Amarillo 
	Categoria M2 y M3 Que estén destinados a transportar personas en áreas urbanas e interurbanas.	Placa Blanca con una linea superior en color Verde 
	Categoria M2 y M3 Que estén destinados a transportar personas entre provincias del pais	Placa Blanca con una linea superior en color Naranja 
Placas para Vehiculos Livianos y Pesados – N	Categoria N1, N2, N3 Vehiculos Motorizados para el transporte de mercaderias	Placa Amarilla en su totalidad 

Placas para Vehiculos Livianos y Pesados – O	Categoria O1, O2, O3, O4 Vehiculos no motorizados para el transporte de mercancias. Remolques (incluidos Semi-Remolques)	Placas Amarillas con una linea superior en color Blanco 

5. Formato de caracteres de placas. La Asociación automotriz de Perú también define el primer carácter de la placa, esta dependerá de la región en al que se registra el vehiculo para obtener la placa (Asociación Automotriz del Perú, 2016a).

La existencia de este formato, definido por ley, permite extraer el dato de la procedencia del vehículo únicamente con un carácter. Si la placa de un vehiculo inicia con la letra P, al utilizar la Tabla 6 se puede definir que pertenece a Tumbes o Piura, en el caso de Lima se definen como primeros caracteres letras de la A hasta F. Cabe mencionar que el carácter "E" será primero en placas especiales.

Tabla 6

Primer carácter para vehiculos menores, livianos y pesados según su número de región.

Región por número	Departamentos o Regiones	Primer Carácter
1	Tumbes y Piura	P
2	Lambayeque, Cajamarca y Amazonas	M, K
3	San Martin	S
4	Loreto	L
5	La Libertad	T
6	Ucayali	U
7	Ancash	H
8	Junin, Huánuco y Pasco	W
9	Lima	A, B, C, D, F
10	Pisco, Apurimac y Madre de Dios	X
11	Ica y Huancavelica	Y
12	Arequipa	V

6. Reconocimiento Automático de Placas Vehiculares. El reconocimiento automático de placas vehiculares o "Automatic License Plate Recognition" (ALPR); cumple un rol de gran importancia para diversos sistemas enfocados en vehículos, como la imposición de multas o cobro de peaje de forma automática, para controlar el ingreso a estacionamientos y vigilancia en carretera (Du et al., 2013).

Un sistema ALPR tiene como principal tarea, extraer los caracteres de la placa de un vehículo en una imagen o secuencia de estas en un video; las imágenes suelen ser capturadas con cámaras de gran resolución que se instalan en postes, peajes de autopistas o puertas de entrada vehicular (Al-Shemarry et al., 2018). En particular, la detección de placas resulta ser una etapa crítica para los sistemas ALPR; dado que disminuye la dificultad de obtener la placa, al limitar la zona de estudio para las siguientes etapas. Esta tarea se complica debido a que se requiere un algoritmo capaz de detectar la placa en diferentes condiciones ambientales y considerando las variaciones en las placas peruanas. Para superar estas dificultades, el sistema requiere de una gran cantidad de imágenes que le permitan definir las características de una placa vehicular de Perú, a este grupo de imágenes se le denomina conjunto de datos o *dataset* (Silvano et al., 2021).

7. Conjunto de datos de placas vehiculares. La evolución de los sistemas ALPR ha experimentado un gran impulso gracias al uso de algoritmos especializados. Por otro lado, la cantidad de datos que se tengan para el entrenamiento determina la eficiencia del sistema, conjuntos de imágenes más grandes brindan la opción de utilizar arquitecturas más complejas y por ende robustas y precisas (Laroca et al., 2018).

Los *dataset* de uso público existentes, si bien incluyen gran cantidad de imágenes, han sido creados para aplicaciones en lugares específicos y las características de las placas vehiculares varían dependiendo del país, haciendo muy difícil su uso para obtener buenos resultados genéricos (González-Cepeda et al., 2022). Por ejemplo, se observa que algunos conjuntos de datos tienen placas con dos filas de caracteres, que pueden estar inclinados o tener baja resolución debido a la calidad de la cámara o a la distancia entre el

vehículo y la cámara. Precisamente estas diferencias han demostrado reducir la precisión de los modelos al intentar usar *dataset* extranjeros para detección de placas en un país distinto, incentivando a que se creen *dataset* con placas de múltiples países en un intento por obtener sistemas para uso general (Laroca et al., 2022).

Se concluye que los conjuntos de datos para ALPR requieren especificaciones avanzadas según el país o la región donde se implemente el sistema. Por lo tanto, un conjunto de datos creado para un estudio puede ser menos útil para otro y la creación de un *dataset* particular para placas peruanas resulta ser una solución. Sin embargo, existe la necesidad de un conjunto de datos de referencia que indique las condiciones y desafíos comunes del mundo real, de modo que pueda usarse como una referencia para la creación de un conjunto de datos para un sistema ALPR (Shashirangana et al., 2021).

En la Tabla 7 se presenta una comparativa entre diversos conjuntos de datos pertenecientes a múltiples países.

Tabla 7

Resumen de los dataset de placas vehiculares en el mundo y sus características

Dataset	Año	País	Imágenes	Vehículos
MTAVLP (Quoc et al., 2021)	2021	Vietnam	10773	Carros, camiones
GAP-LP (Kessentini et al., 2019)	2019	Túnez	9175	Carros
CCPD (Z. Xu et al., 2018)	2018	China	250000	Carros, motos
UFPR-ALPR (Laroca et al., 2018)	2018	Brasil	4500	Carros, motos, camiones, buses
PKU (Yuan et al., 2017)	2017	China	3977	Buses, camiones, carros
SSIG-SegPlate (Gonçalves et al., 2016)	2016	Brasil	2000	Carros
OpenALPR (OpenALPR, 2016)	2016	Europa, Brasil, US	510	Carros
AOLP (Hsu et al., 2013)	2013	Taiwán	2049	Carros
ChineseLP (Zhou et al., 2012)	2012	China	410	Carros

Dentro de los *dataset* más antiguos se tiene, AOLP (application-oriented license plate) y ChineseLP, estos *dataset* fueron desarrollados por países asiáticos con el objetivo

de implementar sistemas ALPR. Las imágenes se obtuvieron mediante cámaras colocadas en diferentes lugares, horas, climas, iluminaciones e inclinaciones; lo que permitió tener un *dataset* variado (Hsu et al., 2013). Estas variaciones hacen que sea un *dataset* muy complejo para la detección, generando sistemas más robustos ante cambios, sin embargo, tiene un tamaño limitado de imágenes. Otro *dataset* reducido es OpenALPR, las imágenes son relativamente simples debido a que fueron capturadas principalmente con cámaras de mano y, a menudo, solo existe un vehículo por fotograma. Además, llevan escenarios menos complicados del mundo real y, por lo general, las matrículas están bien centradas en la imagen (Laroca et al., 2021). Conjuntos de datos con mayor tamaño son SSIG-SegPlate y UFPR-ALPR, ambos fueron producidos en Brasil e incluyen imágenes de alta resolución, lo que permite el reconocimiento de placas a mayor distancia. En ambos conjuntos de datos, hay varios fotogramas de cada vehículo y, por lo tanto, se puede utilizar información redundante para mejorar los resultados del reconocimiento (Cabral et al., 2021). PKU tiene un tamaño similar a los antes mencionados, este fue diseñado para incluir diversos escenarios y condiciones, de forma que presenta una complejidad similar a los *dataset* asiáticos antiguos (Yuan et al., 2017). Si bien el conjunto de datos PKU se acerca a los 4000 datos y UFPR-ALPR los supera, el *dataset* GAP-LP alcanza las 9000 imágenes. Este fue desarrollado en Túnez, dentro de sus ventajas está que las imágenes fueron capturadas con diferentes cámaras, generando distintas calidades, iluminaciones y ángulos (Kessentini et al., 2019). Por otro lado, la cantidad de imágenes permite tener más datos para el entrenamiento y pruebas, al igual que el *dataset* MTAVLP que posee más de 10 000 imágenes etiquetadas. Este *dataset* continua con la línea de tener variedad de imágenes en cuanto a iluminación, sin embargo, únicamente se tomaron imágenes en dos posiciones definidas por lo que los escenarios no eran tan variados (Quoc et al., 2021). Es en el año 2018 que se presentó el que sería el *dataset* público con mayor cantidad de imágenes etiquetadas de placas vehiculares; como se observa en la Tabla 7 CCPD incluye 250 000 imágenes de vehículos, por lo que se estudiará la metodología tomada.

En la investigación que presenta el *dataset* "Chinese City Parking Dataset" (CCPD) se menciona que muchos estudios validan sus sistemas con *dataset* muy limitados y con condiciones ambientales controladas. Resaltan que los *dataset* públicos que se tenían hasta el año 2018 tenían dos falencias principales; la poca cantidad de imágenes, menor a 3000, y que las imágenes son adquiridas de puntos fijos, como cámaras de seguridad. CCPD recopila las fotos de vehículos en estacionamientos y calles en diversas partes de una provincia de China, donde los habitantes poseen miles de autos. En este caso se solicitó a los cobradores de los estacionamientos, quienes trabajan desde 7:30 am hasta las 10:00 pm todos los días independientemente de las condiciones climáticas, que tome foto a cada vehículo que aparque. Las fotos fueron tomadas con un dispositivo Android de mano, por lo que las imágenes presentan fuertes variaciones debido a la posición y el ángulo de disparo inciertos, así como a las distintas iluminaciones y fondos a distintas horas y en distintas calles (Z. Xu et al., 2018).

Figura 7

Imágenes de muestra del CCPD. Cada imagen está etiquetada con su recuadro (el borde amarillo) y la ubicación de los cuatro vértices (cuatro puntos rojos).

Como se observa en la Figura 7 existen distintos escenarios, los cuales son definidos como sub-*dataset* dependiendo de las características de la imagen.

Tabla 8

Descripción de los subgrupos de imágenes en el dataset CCPD

	Descripción
CCCP-Base	Imágenes que incluyen placas vehiculares con características estándar
CCPD-DB	La iluminación en la placa es muy oscura, irregular o muy brillante
CCPD-FN	La placa se encuentra muy cerca o muy lejos de la cámara
CCPD-Rotado	La placa esta rotada entre 20° y 50°, e inclinada entre -10° y 10°
CCPD-Inclinado	La placa esta rotada entre 15° y 45°, e inclinada entre 15° y 45°
CCPD-Difuminado	Placas difuminadas o borrosas por movimientos de la cámara
CCPD-Clima	Imágenes tomadas en dias lluviosos, nevados o nublados
CCPD-NP	Imágenes de carros sin placa

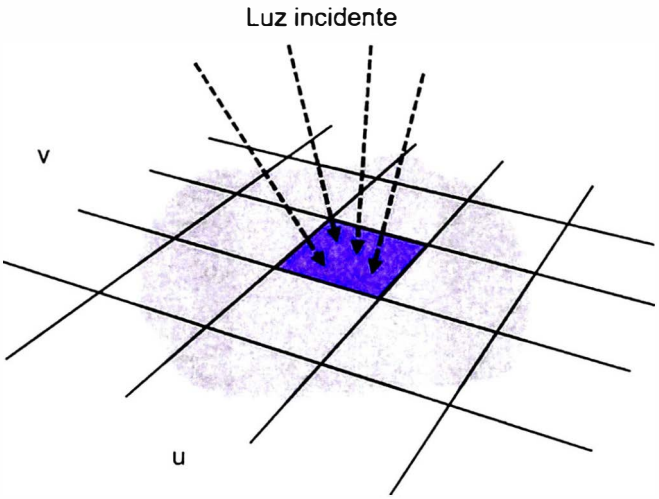
La existencia de los grupos definidos en la Tabla 8, demuestra la complejidad que hace que el *dataset* CCPD sea ampliamente utilizado para realizar el entrenamiento y validación de sistemas, por lo que la metodología utilizada y las características incluidas para las imágenes son una base correcta para la creación de un *dataset* propio para placas peruanas.

8. Digitalización de imágenes. Existen diversos dispositivos que generan imágenes, cámaras de video, escáneres, equipos médicos, radares, entre otros. La necesidad de utilizar estas imágenes en sistemas autónomos y computadoras ha incentivado el desarrollo de métodos de procesamiento digital de imágenes que permitan mejorar su calidad u obtener información a partir de ellas (Shih, 2017).

Con el objetivo de analizar las imágenes, estas se deben digitalizar antes de ser procesadas por el sistema. La obtención de una imagen digitalizada involucra tres pasos esenciales que transforman una señal visual continua en una representación discreta que puede ser almacenada y procesada por una computadora. En primer lugar, el proceso comienza con el muestreo espacial de la luz, donde la imagen analógica se divide en elementos discretos mediante sensores, como los de una cámara digital. Estos sensores están dispuestos en filas ordenadas (como un plano cartesiano), usualmente formando ángulos rectos entre sí en el plano del sensor. La Figura 8 presenta de forma gráfica el plano del sensor.

Figura 8

Representación del plano del sensor, el recuadro azul representa el elemento de la imagen con coordenadas I(u,v)

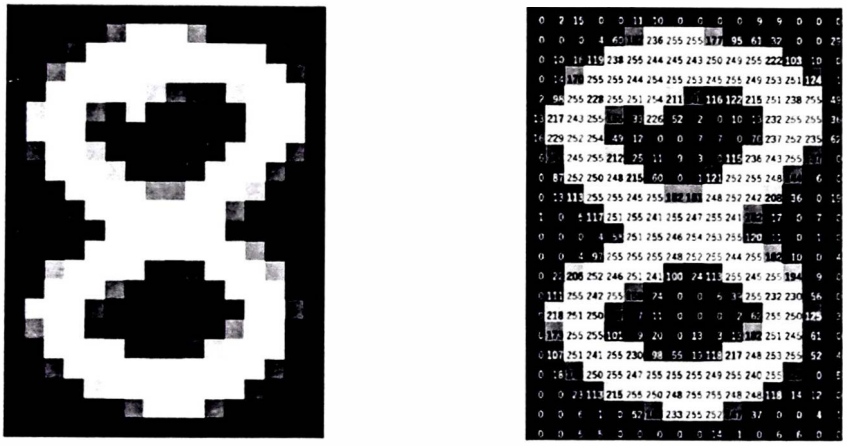


Luego, el muestreo temporal se encarga de medir la cantidad de luz incidente en cada sensor en intervalos regulares a lo largo del tiempo. Para facilitar su manipulación en la computadora, los valores de la imagen se convierten comúnmente a una escala entera, como 256 o 4096, mediante un proceso de conversión analógico a digital. Este último paso se realiza mediante un convertidor integrado en la electrónica del sensor o a través de hardware de interfaz especializado (Burger & Burge, 2016).

De forma conceptual, una imagen en su forma más simple o de un único canal, por ejemplo; binaria, monocromática, escala de grises o blanco y negro, es una función de dos dimensiones que se puede expresar como $f(x, y)$. Esta función se encarga de asignar un valor entero a cada par de coordenadas, este valor está relacionado con la intensidad o color del punto, cada uno de estos puntos se denomina píxel. Se sabe que las imágenes puedes tener múltiples canales, como las RGB donde un color se representa usando tres canales; rojo, verde y azul. Para este tipo de imágenes, cada píxel en la coordenada (x, y) se representa por una tupla $(r_{x,y}, g_{x,y}, b_{x,y})$ (Dey, 2018). El valor de cada píxel correspondiente a un canal es representado por un entero entre 0 y 255, donde los números más pequeños (más cercanos a cero) representan el negro, y los números más grandes (más cercanos a 255) denotan el blanco.

Figura 9

Descripción gráfica del almacenamiento digital de una imagen

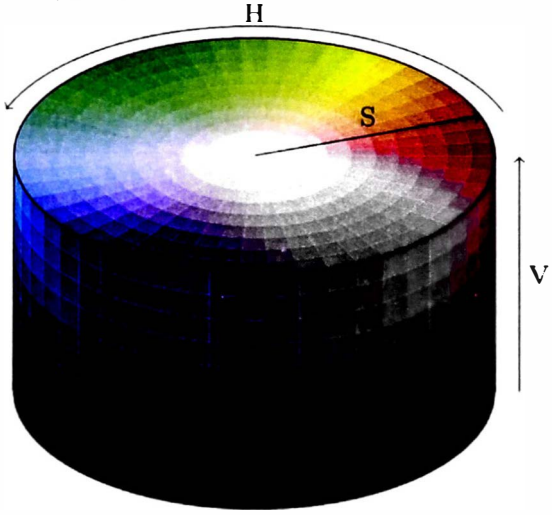


En la Figura 9 se observa un ejemplo en escala de grises, la imagen de la derecha representa la matriz de valores con la que el computador trabaja. Esto va en concordancia con el proceso de digitalización, que brinda como resultado un valor de intensidad propio de cada píxel.

9. Formato de color HSV. Dada la necesidad de realizar procesamiento del color en las imágenes aparecen formatos para definir el color de cada píxel. El espacio HSV es comúnmente utilizado para realizar esta tarea. Sus siglas provienen de la forma en la que se describe el color utilizando tres canales: el tono (Hue), la saturación (Saturation) y el valor (Value). En el espacio HSV el color es codificado como una tripleta $(\boldsymbol{H}, \boldsymbol{S}, \boldsymbol{V}) \in [\mathbf{0}, \mathbf{360}[\times [\mathbf{0}, \mathbf{100}] \times [\mathbf{0}, \mathbf{100}]$. Para la comprensión de estos canales se presenta la Figura 10, donde el tono (H) es representado como un ángulo en el cilindro, la saturación (S) y el valor (V) se definen como porcentajes. La saturación permite la obtención de colores derivados; el valor, al representar la iluminación permite obtener colores más oscuros o claros (Bredies et al., 2018).

Figura 10

Representación cilíndrica del espacio de color HSV.



2.1.2 Fundamentos teóricos para algoritmos de Deep Learning

En esta sección se presentan los fundamentos para la comprensión de los algoritmos que se implementaron para el desarrollo del sistema de detección.

1. **Detección de placas vehiculares.** La detección de la placa del vehículo representa una etapa importante en la identificación vehicular, por esta razón se han analizado diferentes métodos que logran esta tarea. En el artículo, "Automated License Plate Recognition: A Survey on Methods and Techniques" los autores realizan una revisión de los métodos que se han aplicado y muestran sus principales características. La clasificación que se muestra en el artículo es, por un lado, métodos que aplican técnicas tradicionales de visión por computador y por otro, métodos clasificadores (Shashirangana et al., 2021).

   a. ***Métodos basados en detección de bordes.*** En este caso se toma como base la idea de que las placas vehiculares son rectangulares y con ciertas medidas, por esta razón se utilizan algoritmos para extraer las características de los bordes. Esto funciona al existir una clara diferencia en los colores del vehículo y del interior de la placa, permitiendo obtener los bordes que conformen un rectángulo con las dimensiones de una placa.

b. ***Métodos basados en colores.*** Para estos algoritmos se toma de base el concepto de la diferencia de colores entre el fondo de la placa y el vehículo. Hacen uso de algoritmos genéticos, histogramas y matemática difusa para obtener sistemas capaces de afrontar la obvia presencia de ruido cuando se trabaja con colores.

c. ***Métodos basados en texturas.*** Al usar este método se considera la presencia de los caracteres de la placa vehicular en la imagen para realizar la detección. La diferencia entre el color del fondo de la placa y la de los caracteres generará una línea de transición, la cual en la escala de grises resalta de forma clara.

d. ***Métodos basados en caracteres.*** Para este método se busca la región que contenga caracteres en su interior, con el objetivo de localizar posibles candidatos de placas vehiculares. Con este objetivo se implementa la extracción de las posibles regiones con caracteres en la imagen, sobre las cuales usar una red neuronal de clasificación con características de placas vehiculares.

e. ***Métodos basados en Deep learning.*** Hace uso de los desarrollos realizados en el ámbito de la visión por computador. En particular de las redes neuronales convolucionales que permiten realizar la localización de la placa vehicular de manera precisa.

En la Tabla 9 se presenta un cuadro que resume las principales ventajas y limitaciones de cada técnica que los autores analizaron en el artículo. Los dos primeros métodos tienen como principal desventaja la sensibilidad ante variaciones en la imagen, esto limita la posibilidad de utilizar estos métodos en situaciones complejas. Son los métodos basados en aprendizaje profundo los más efectivos y robustos para realizar la tarea de detectar la placa en las imágenes sin importar las variaciones, sin embargo, requieren de mayor capacidad computacional.

Tabla 9

Resumen de técnicas para obtención de placas vehiculares en una imagen con sus ventajas y limitaciones

Técnica	Ventajas	Limitaciones
Métodos basados en bordes	Simple y rápido	Sensible ante bordes no deseados, no aplicable con imágenes borrosas y complejas
Métodos basados en color	Robusto ante deformaciones y rotaciones	Sensible a cambios de iluminación
Métodos basados en texturas	Robusto ante deformaciones de contornos en placas	Complejidad computacional, bajo desempeño con fondos complejos y cambios de iluminación
Métodos basados en caracteres	Puede ser usado para placas rotadas	Consume mucho tiempo y no aplicable para imágenes con textos extra
Métodos basados en aprendizaje profundo	Eficiente y robusto ante cambios ambientales	Consume muchos recursos, complejidad computacional

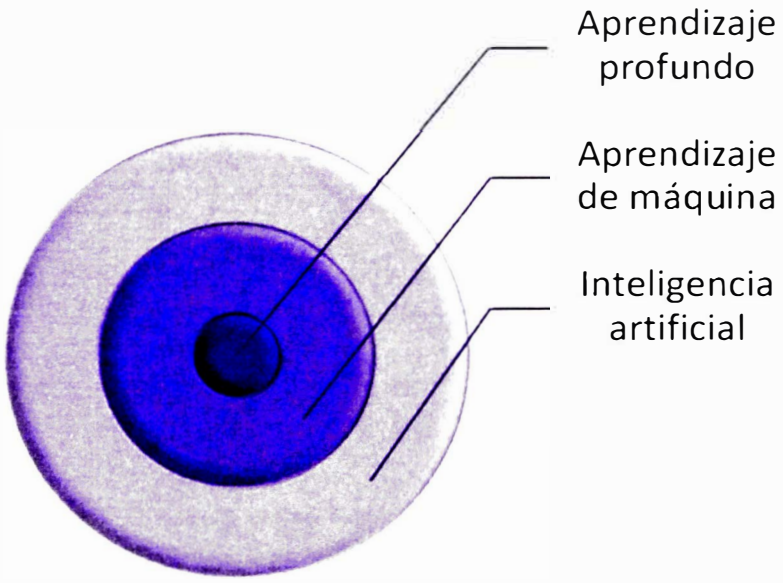
2. **Aprendizaje profundo.** Para comprender los métodos basados en aprendizaje profundo o *Deep learning*, se debe tener claro el concepto de Inteligencia Artificial (IA). Esta ha sido definida como la aplicación y utilización de la potencia computacional con el objetivo de emular las habilidades propias de los seres humanos, como podría ser pensar, sentir o realizar actividades físicas (Mariani et al., 2023). La IA también puede ser entendida al analizar los denominados agentes inteligentes, los cuales son todo tipo de dispositivo dotado con la capacidad de percibir su alrededor y capaz de actuar de forma óptima para el cumplimiento de su tarea con éxito. Informalmente, el concepto "inteligencia artificial" es usado si un dispositivo es capaz de realizar funciones que pueden ser asociadas a otra mente humana, como podría ser "aprender" o "resolver problemas" (Shinde & Shah, 2018). En el contexto de la innovación, se entiende a la IA como el desarrollo de sistemas que posean comportamientos de percepción, razonar y actuar en base a ello (Prem, 2019).

La inteligencia artificial cumple con brindar a las máquinas de la capacidad para aprender y percibir, por lo que se define que el aprendizaje de máquina es un campo dentro

de la IA. El aprendizaje de máquina o Machine Learning (ML) es un proceso que consiste en enseñar a un computador a aprender y predecir a partir de datos. También es un algoritmo o un modelo matemático que brinda a un computador la posibilidad de realizar acciones sin ordenes específicas (D. Zhao & Lorin, 2020). Con el avance de las tecnologías, el campo del aprendizaje de máquina enfrentó tareas que requerían de más esfuerzos para ser efectiva, esto generó la necesidad de plantear algoritmos que permitan reducir el error de forma óptima. Aparecen entonces una serie de técnicas, como un subconjunto del aprendizaje de máquina denominado "aprendizaje profundo". Estas técnicas proponen realizar un modelado abstracto de los datos en un nivel superior, mediante el uso de arquitecturas de múltiples capas que permiten implementar transformaciones de forma lineal y no lineal (Gonazales, 2018).

Figura 11

Representación de la jerarquía en el campo de la inteligencia artificial



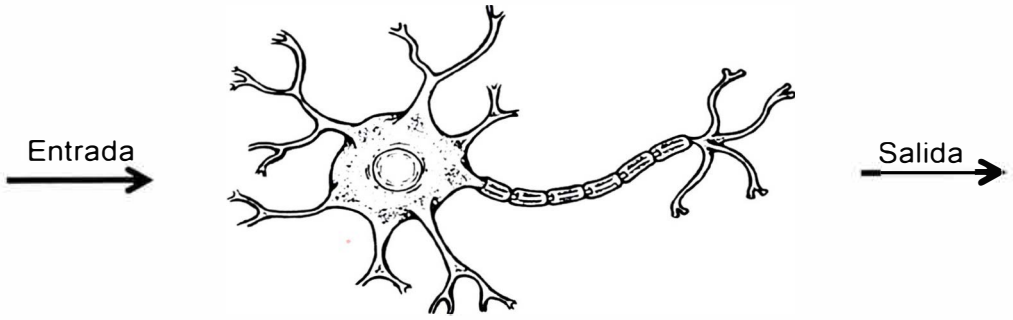
En el artículo "A Review of Machine Learning and Deep Learning Applications", los autores mencionan que el aprendizaje profundo utiliza lo que se denomina una "cascada" de múltiples capas de unidades de procesamiento no lineal para la extracción y transformación de características (Shinde & Shah, 2018). De esta manera el aprendizaje profundo permite desarrollar modelos que aprendan a partir de las características

obtenidas por cada capa, dependiendo de su nivel de abstracción que aumenta a mayor profundidad. Además, se usan para aprender funciones muy complejas con la cantidad necesaria de transformaciones (Lecun et al., 2015). Las capas de un modelo de aprendizaje profundo se divide en tres clases; capas de entrada, salida y ocultas. Las capas ocultas conforman una estructura capaz de realizar el mapeado de características entre las capas de entrada y salida (Khazaee et al., 2020).

3. **Redes neuronales artificiales.** Los modelos de aprendizaje profundo utilizan estructuras y capacidades de las redes neuronales artificiales (RNA), estas son algoritmos derivados del aprendizaje de máquina que se inspira en las redes neuronales biológicas. Estas redes se basan principalmente en nodos que actúan como unidades de procesamiento y son análogas a las neuronas. Los nodos reciben información por las vías de entrada, análogas a las dendritas, y la transmiten por las vías de salida, análogas al axón como se observa en la Figura 12 (R. Y. Choi et al., 2020).

Figura 12

Representación gráfica del ingreso y salida de una señal en una neurona biológica

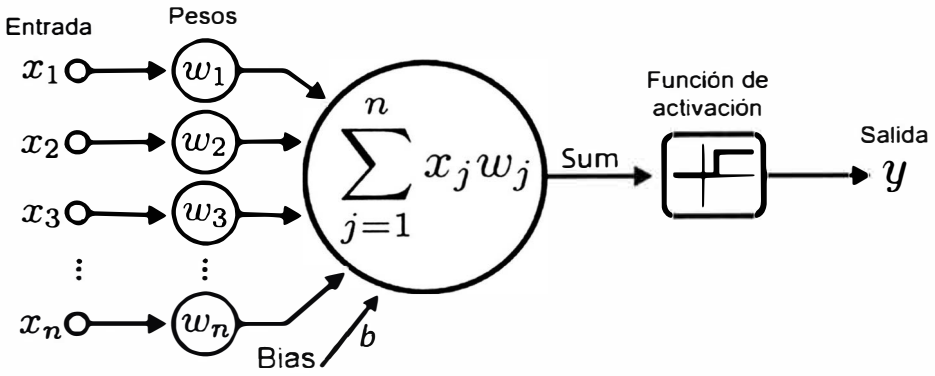


La información es enviada entre neuronas mediante conexiones, a las que se asigna un valor numérico real que define el peso o importancia de la información (Islam et al., 2019). La forma de modelar matemáticamente el funcionamiento de los nodos y la acción de los pesos, se define mediante el concepto de perceptrón. En la década de 1940, Warren McCulloch y Walter Pitts describieron un modelo para el funcionamiento de las neuronas. Definen la existencia de un valor umbral, el cual debe ser superado por la señal para ser transmitida (McCulloch & Pitts, 1943). Una aplicación de esto se da en la

implementación de una red neuronal básica o red perceptrón de una capa, que consta de una sola capa de nodos de salida. Las entradas ingresan de forma directa a las salidas mediante los valores de peso, los cuales son usados para realizar cálculos en cada nodo. Si el valor obtenido está por encima de algún umbral (normalmente 0), la red neuronal se activa y toma el valor activado (normalmente 1); en caso contrario, toma el valor desactivado (normalmente -1). Las neuronas con este tipo de función de activación también se denominan neuronas artificiales o unidades de umbral lineal (Lupu, 2021). Este modelo fue evolucionando debido a la necesidad de utilizar funciones distintas, hasta llegar a la idea de perceptrón representado en la Figura 13.

Figura 13

Representación del funcionamiento de una neurona (perceptrón) en redes neuronales



Cada unidad recibe valores de entrada y una constante, se realiza una suma ponderada de estas para luego pasar por una función no lineal que define el valor de salida. El modelo planteado tiene como entrada un vector de números reales $x = (x_1, x_2, \cdots, x_n)^T$ y un vector de pesos, $w = (\omega_1, \omega_2, \cdots, \omega_n)^T$, con el cual se ponderan los valores de entrada (Marius-Constantin et al., 2009). El mapeo resultante a la salida del nodo es el siguiente.

$$f(x) = a(w \cdot x + b) \qquad (1)$$

En la Ecuación (1), la operación "$w \cdot x$" es el producto punto de los vectores de entrada y los pesos, el resultado se obtiene al realizar la sumatoria de cada elemento de los vectores como se observa en la Figura 13. A la suma se le agrega el valor denominado *bias* (b), este parámetro define el valor que debe ser superado para activar la neurona. Por último, la función a representa la denominada "función de activación" (Fernandez, 2019).

Esta es definida como una función, con todos los números reales como dominio y rango, tal que sea diferenciable casi en toda su extensión. Esta definición da a entender que cualquier función continua y diferenciable podría ser usada para la activación, sin embargo, hay funciones comúnmente utilizadas debido a sus características y simpleza para el procesamiento (Ding et al., 2018). La función de activación tiene como objetivo principal realizar la conversión de la señal de entrada a una señal de salida, estas señales provenientes de sistemas reales, tienen características complejas que requieren del manejo de propiedades no lineales (Szandała, 2021).

La base de las redes neuronales son los perceptrones; el más básico utilizaba únicamente una función lineal para definir la activación de la neurona, generando como señal de salida un polinomio de grado uno. Si bien este es simple de resolver, brinda a la red la capacidad de aprender y reconocer señales más complejas. Precisamente son el uso de funciones de activación complejas las que diferencian a las redes neuronales de simples regresiones lineales (Sharma et al., 2020). El uso de una función no lineal dentro del modelo de neurona presentada, permite obtener resultados para el aprendizaje de sistemas de mayor dificultad, que presenten no linealidades y grandes dimensiones. De forma que, se les permita a las redes neuronales de múltiples capaz extraer conocimiento. Dentro de las funciones más usadas para esta tarea están la tangente hiperbólica, función sigmoide y la unidad lineal rectificada (ReLU).

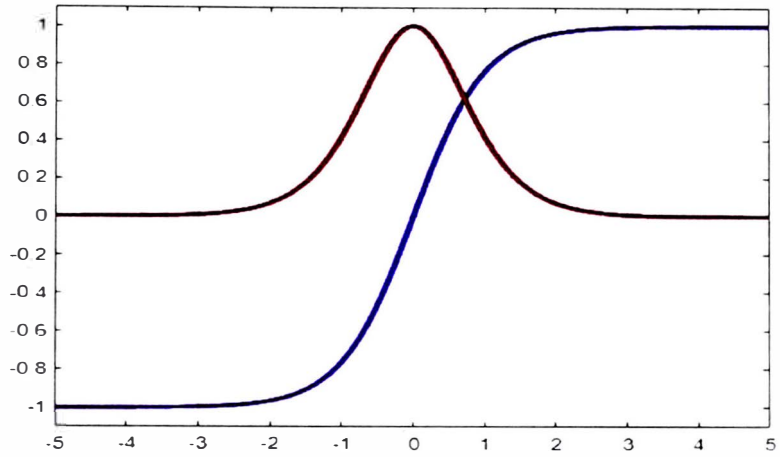
a. ***Función de activación tangente hiperbólica.*** Esta función define la relación entre las funciones seno y coseno hiperbólicos, se define en la Ecuación (**2**) (The Mathworks Inc., 2022).

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{2}{1 + e^{-2x}} - 1 \qquad (2)$$

La forma de esta función se observa en la Figura 14, es una curva suave con el rango de los valores de entrada en el intervalo de [-1,1] con estructura parecida a la de la función sigmoide.

Figura 14

Gráfica de la tangente hiperbólica en color azul y su derivada en color naranja



La derivada de la función tangente hiperbólica es pronunciada, esta se observa en la Figura 14 y define los valores máximos que se obtienen al derivar. A pesar de presentar un gradiente pronunciado y valores de salida acotados, esta función tiende a reducir su gradiente hasta desaparecer (Rasamoelina et al., 2020).

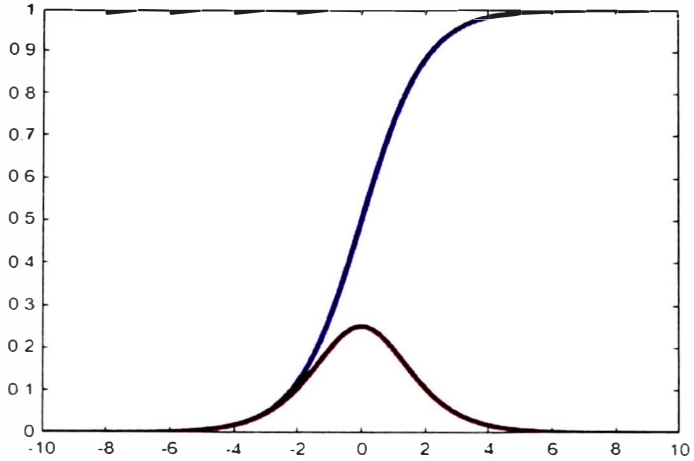
b. ***Función de activación sigmoide.*** Esta función es real, acotada y diferenciable, con derivada positiva en toda su extensión. Como se observa en la Figura 15, tiene una estructura de curva suave al igual que la tangente hiperbólica y presenta no linealidades similares de igual forma. La función sigmoidal se aproxima rápidamente a un límite superior finito fijo asintóticamente a medida que su argumento aumenta, y se aproxima rápidamente a un límite inferior finito fijo asintóticamente a medida que su argumento disminuye. Matemáticamente queda definida en la Ecuación (**3**) (Han & Moraga, 1995).

$$f(x) = \frac{1}{1 + e^{-cx}} \qquad (3)$$

Dado que esta función es acotada, permite realizar la transformación del rango de entrada infinito a valores de salida en el intervalo de $[0; 1]$. Por esta razón, las salidas de cada unidad se reducen produciendo un desvanecimiento del gradiente.

Figura 15

Gráfica de la función sigmoide en color azul y su derivada en color naranja

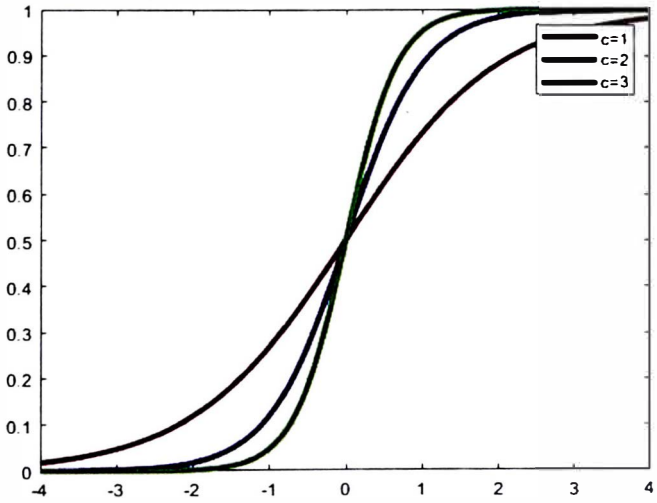


Como se observa en la Figura 15 la derivada será menos pronunciada que la de la tangente hiperbólica, por lo que toma valores menores y esto genera mayor desvanecimiento del gradiente (Rasamoelina et al., 2020).

La definición de la función sigmoide permite manipular la variable c, la cual representa la inversa de la temperatura en la función de Fermi utilizada por los físicos para describir la distribución de energía. Este parámetro permite modificar la función aproximadamente lineal entre los valores acotados, como se ve en la Figura 16 (Müller et al., 1995).

Figura 16

Variación de la pendiente según el parámetro c



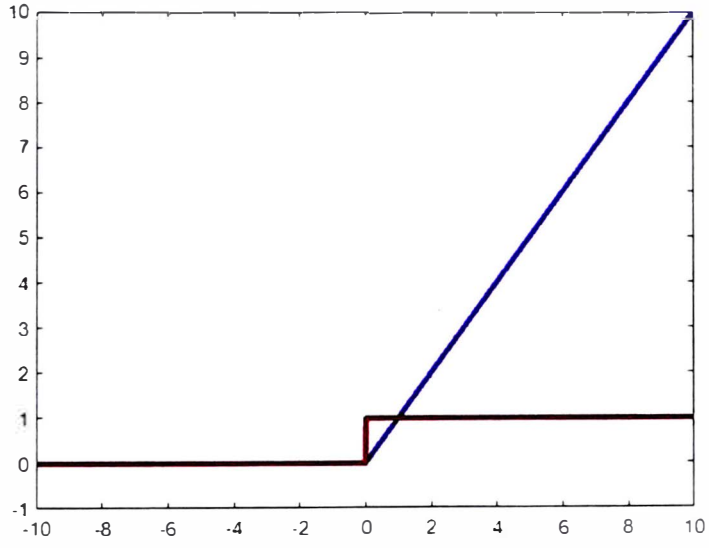
c. ***Función de activación ReLU.*** Esta es una función ampliamente utilizada para aplicaciones en redes neuronales, fue introducida en (Hahnloser et al., 2000). Su función principal es mantener valores a partir de cierto valor umbral, en este caso es 0. Funciona de forma que para valores menores a 0, la salida será 0 y para los demás valores se tendrá una función lineal (Agarap, 2018). En la Figura 17 se observa de manera gráfica el funcionamiento de la función de activación, con valores de salida de 0 al infinito al tener como función lineal la identidad, esta se define en la Ecuación (**4**).

$$ReLU(x) = \max(0, x) \qquad (4)$$

Una ventaja es la simplicidad de la función, además brinda un valor de cero real para los datos de entrada que sean negativos. Sin embargo, esto representa también una desventaja en su aplicación para redes, debido a que si una unidad se convierte en negativa no podrá recuperar el valor, provocando que se desvanezca la gradiente (Rasamoelina et al., 2020).

Figura 17

Gráfica de la función de activación ReLU en color azul y su derivada en color naranja

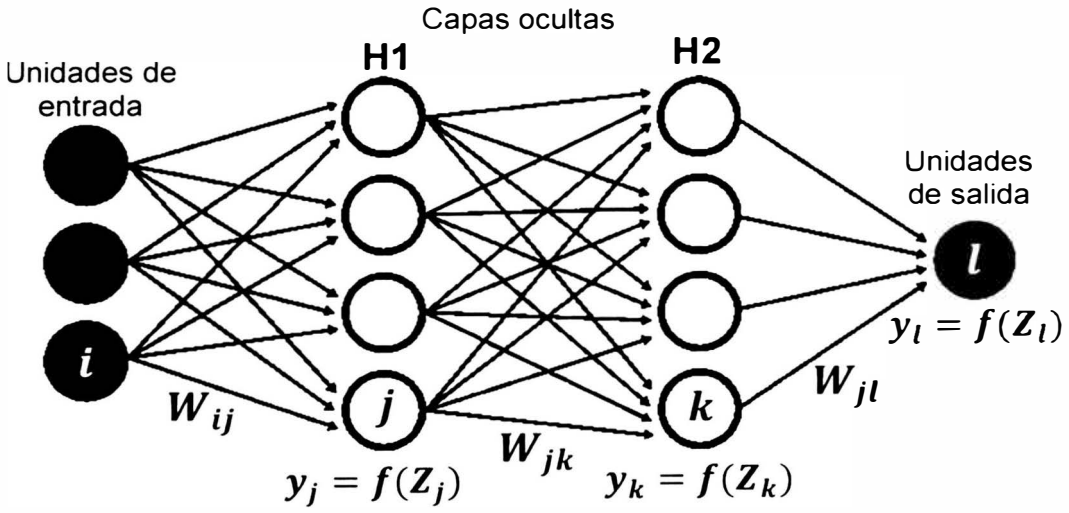


4. Redes neuronales feed-forward. Se concluye que las redes neuronales son un conjunto de unidades de procesamiento que se encargan de realizar las operaciones presentadas en la Ecuación (1). Luego de recibir las señales de entrada, estas

se ponderan antes de llegar a los nodos de salida según su importancia respectiva para que finalmente, la función de activación procese la señal combinada. Al concatenar varias neuronas se crea una estructura de red neuronal, donde se intercambia información de forma que la red aprenda (Z. Zhang, 2016). La estructura que generan las capas de neuronas se denomina red neuronal *feed-forward* o de propagación hacia adelante, en el caso de que no haya realimentación. Este tipo de redes pueden ser de una capa, donde únicamente presenta una capa de entrada y otra de salida; o multicapa, donde existirán capas intermedias denominadas "ocultas"(Sazli, 2006).

Figura 18

Esquema del funcionamiento de una red neuronal feedforward multicapa



En la Figura 18 se presenta la estructura de una red neuronal feedforward multicapa, donde cada unidad que realiza el procesamiento de información está conectada con todas las unidades de la capa contigua. Aquí se define la función de mapeo Γ, esta asigna a cada neurona i un subconjunto $\Gamma(i) \subseteq V$ que consiste en los antecesores de la neurona i. La conexión entre una neurona i con una j están definidas por el coeficiente de peso W_{ij} y el valor umbral b_j (Svozil et al., 1997). Con estos parámetros se realiza el cálculo descrito y presentado en la Ecuación (**1**) en cada capa para obtener el valor de *Z*.

$$Z_l = \sum_{k \in H2} W_{kl} * y_k + b_l \qquad (5)$$

$$Z_k = \sum_{j \in H1} W_{jk} * y_j + b_k \qquad (6)$$

$$Z_j = \sum_{i \in entrada} W_{ij} * x_i + b_j \qquad (7)$$

El valor de Z es la suma ponderada de las entradas para cada capa, para la primera se utilizan las entradas x y su valor umbral b_j, esta luego de pasar por la función de activación f, se convierte en la salida que se usa en el cálculo de la capa H_1. Esto se repite de forma sucesiva hasta obtener la salida en la capa final. Al comprender dicho proceso se establece que las redes neuronales profundas (DNN) se clasifican como una variante de redes neuronales denominada perceptrón multicapa (MLP), cuyo entrenamiento se basa en algoritmos que permiten aprender representaciones a partir de *dataset* sin necesidad de diseñar manualmente extractores de características. Como su denominación sugiere, el aprendizaje profundo implica la incorporación de una mayor cantidad de capas de procesamiento, en contraposición al modelo de aprendizaje superficial caracterizado por un menor número de capas de unidades. El paso del aprendizaje superficial al profundo ha permitido mapear funciones más complejas, en especial con el gran avance que significo la llegada del algoritmo de *backpropagation* o retro propagación (Shrestha & Mahmood, 2019).

5. Algoritmo de retro propagación. Este algoritmo es de los más usados para la implementación del aprendizaje en redes *feed-forward* multicapa (Riedmiller & Braun, 1993). Este algoritmo tiene como base teórica los siguientes principios:

- El sistema genera relaciones entre sus unidades, a través de una serie de pruebas
- El sistema puede aprender "fácilmente" otras tareas que sean similares a las que ya ha aprendido, y luego, para realizar "generalizaciones".
- Las relaciones estabilizadas entre unidades del sistema durante el aprendizaje de las tareas son la única memoria del sistema.

- Las unidades del sistema podrán ser únicamente de tres tipos, de entrada, de salida y ocultas.
- Las relaciones entre las unidades del sistema son unidireccionales; es decir, la unidad A, al activarse de una determinada manera, activa la unidad B con respecto a la aproximación en la que A se activó y a la fuerza de su conexión; pero no viceversa (Buscema, 1998).

Estas características son propias de las redes neuronales descritas, la red es una implementación particular de una función compuesta del espacio de entrada al de salida, denominada función de red. Aquí toma valor el llamado "Problema de aprendizaje", el cual tiene como objetivo hallar la forma en la que se deben combinar los pesos para que se optimice la función de red φ. Es decir, que φ sea lo más cercano o de forma aproximada a una función f. Con este objetivo se plantea la definición de la función de error, en la que se define la existencia de una red neuronal *feedforward* con un conjunto de entrenamiento $\{(x_1, t_1), \ldots, (x_p, t_p)\}$ que incluye p cantidad de pares, denominados patrones de entrada y salida. Cuando los patrones de entrada x_i son ingresados a la red, esta genera una salida o_i que usualmente será distinta al valor buscado t_i, mediante el uso del algoritmo de aprendizaje se busca igualar estos valores y esto se logrará al minimizar la función de error definida por la Ecuación (8) (Rojas, 1996).

$$E = \frac{1}{2}\sum_{i=1}^{p} \|o_i - t_i\|^2 \qquad (8)$$

El algoritmo de retro propagación entonces actúa sobre los pesos, que en un inicio son aleatorios, reajustando sus valores en las capas de salida y luego en las capas ocultas hasta que el valor de la función de error o pérdida (*loss*) sea suficientemente pequeño (Cilimkovic, 2015). Tras minimizar esta función para el conjunto de entrenamiento, se espera que la red sea capaz de interpolar nuevos vectores de entrada. Esto significa que sea capaz de determinar la existencia de similitud entre los patrones que aprendió y nuevos vectores que ingresan a la red, calcular y usar el gradiente de la función de error con el

objetivo de reajustar estos valores iniciales. La operación que se utiliza es el descenso de gradiente, en la cual se aplica la regla de la cadena para calcular la derivada parcial de cada valor de peso.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial s_i}\frac{\partial E}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}} \qquad (9)$$

En la Ecuación (**9**) w_{ij} es el peso de la neurona j hacia la neurona i, s_i es la salida y net_i es la suma ponderada de las entradas en la neurona i. Con esto se procede a obtener los pesos actualizados usando su incremento, el cual se define en la Ecuación (**10**).

$$\Delta w_{ij}(t) = -\epsilon \frac{\partial E}{\partial w_{ij}}(t) \qquad (10)$$

En la Ecuación (**10**) el parámetro ϵ define la tasa de aprendizaje, la cual escala la derivada y tiene un efecto importante en el tiempo que tarda el sistema hasta alcanzar la convergencia. Si este valor es muy pequeño, la red necesitará demasiados pasos para obtener una solución aceptable; por otro lado, si es un valor alto, producirá oscilaciones al impedir que el error caiga por debajo de cierto valor (Riedmiller & Braun, 1993). Ante esta problemática se plantea el término "*momentum*", este parámetro permite introducir información de iteraciones anteriores para la actualización de los pesos. El valor de este parámetro determina la influencia de las iteraciones anteriores, de esta forma se agrega un efecto amortiguador que reduzca las oscilaciones. Al agregar el parámetro α en la Ecuación (**10**) se obtiene la Ecuación (**11**) (Moreira & Fiesler, 1995).

$$\Delta w_{ij}(t) = -\epsilon \frac{\partial E}{\partial w_{ij}}(t) + \alpha \Delta w_{ij}(t-1) \qquad (11)$$

Luego de la aparición de este método, se plantearon optimizadores que permitieron determinar la velocidad de entrenamiento y el rendimiento del modelo final para realizar predicciones. Estos son algoritmos definidos por una regla de actualización, la cual define parámetros que varían su comportamiento, por ejemplo, la tasa de aprendizaje (D. Choi et al., 2019). En el año 2011 aparece el optimizador AdaGrad dentro de una nueva familia de

métodos de subgradiente que incorporan dinámicamente el conocimiento de la geometría de los datos observados en iteraciones anteriores para realizar un aprendizaje basado en gradientes más informativo (Duchi et al., 2011). Luego, en el año 2012 se presenta el algoritmo RMSProp como un método de tasa de aprendizaje adaptativo que escala la tasa de aprendizaje mediante una media exponencialmente decreciente de gradientes al cuadrado (D. Xu et al., 2021). Con el objetivo de combinar las ventajas de estos métodos se propuso la creación del algoritmo de optimización Adam, nombre derivado de la estimación adaptativa de momentos. Este se presenta como un método para realizar de forma eficiente la optimización estocástica que únicamente requiere gradientes de primer orden con poco uso de memoria. Los cálculos que se realizan para definir la función de actualización se presentan en el siguiente pseudocódigo.

Ingresa α: *Tamaño de paso*
Ingresa $\beta_1, \beta_2 \in [0,1)$: *Coeficientes de decaimiento exponecial*
Ingresa $f_{(\theta)}$: *Función objetivo estocástica con parámetro* θ
Ingresa θ_o: *Vector parámetro incial*
$m_o \leftarrow 0$ (*Inicializa el* 1° *vector momento*)
$v_o \leftarrow 0$ (*Inicializa el* 2° *vector momento*)
$t \leftarrow 0$ (*Inicializa el contador*)
while θ_t *no converge* ***do***
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot {g_t}^2$
$\hat{m}_t \leftarrow \dfrac{m_t}{(1 - \beta_1^t)}$
$\hat{v}_t \leftarrow \dfrac{v_t}{(1 - \beta_2^t)}$
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon$
end while
return θ_t

El pseudocódigo describe el algoritmo que permite minimizar la función objetivo $f_{(\theta)}$, considerada una función estocástica ruidosa que es diferenciable con respecto al parámetro θ. Luego de inicializar los vectores y el contador de paso de tiempo, se aumenta en una unidad el contador y se calcula el gradiente de la función f_t con respecto al parámetro θ en el paso de tiempo t. Con este valor el algoritmo actualiza los promedios

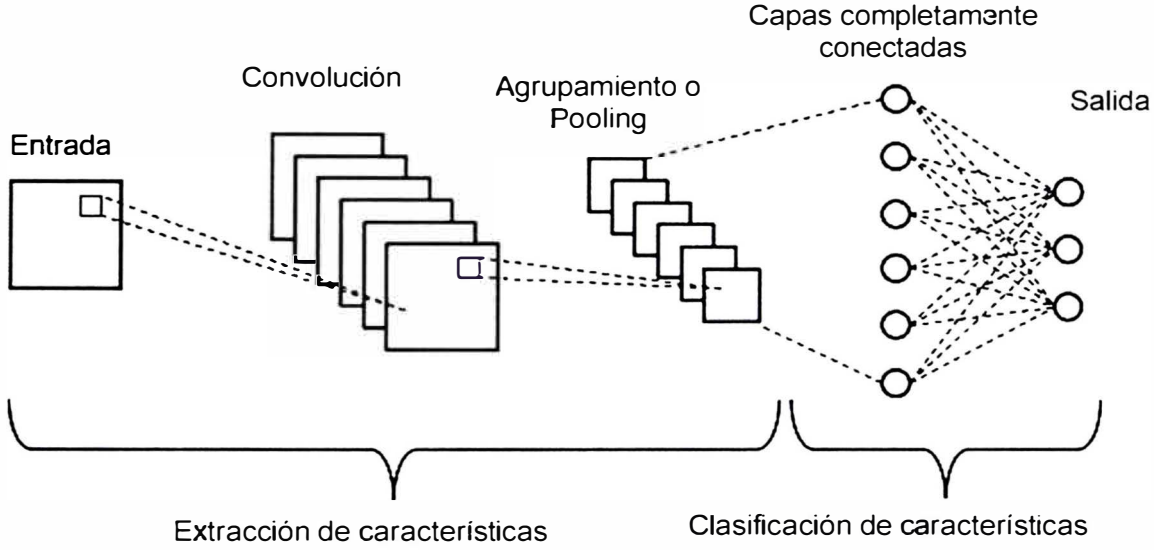
móviles exponenciales del gradiente m_t y el gradiente al cuadrado v_t , cuyos hiperparámetros (β_1 y β_2) permiten controlar las tasas de decaimiento de los promedios móviles. Dado que se inicializan como vectores nulos, las medias móviles podrían estar sesgadas hacia cero, para evitarlo se definen las estimaciones $\hat{m}_t$ y $\hat{v}_t$ que corrigen este posible sesgo (Kingma & Ba, 2014).

6. **Red neuronal convolucional.** La historia de las redes neuronales convolucionales (CNN) pasa por la neurociencia y la inteligencia artificial. Como las redes neuronales artificiales en general, son un ejemplo de ideas inspiradas en el cerebro que se materializan gracias a la convergencia de la informática y la ingeniería (Lindsay, 2021). Una CNN es una arquitectura de aprendizaje profundo inspirada en la estructura del sistema visual del cerebro humano. Las CNN han demostrado ser los mejores algoritmos de aprendizaje para procesar la información disponible en una imagen y se consideran el modelo ideal para diversas tareas relacionadas con las imágenes, como la segmentación, la clasificación y la detección (Voulodimos et al., 2018). El funcionamiento de las CNN es análogo a las tradicionales redes neuronales, dado que están compuestas por capas y neuronas que se actualizan con el aprendizaje. Una de las diferencias clave es que las neuronas que componen las capas dentro de la CNN están organizadas en tres dimensiones, la dimensionalidad espacial de la entrada (altura y anchura) y la profundidad. La profundidad no se refiere al número total de capas dentro de la RNA, sino a la tercera dimensión del volumen de activación. A diferencia de las RNA estándar, las neuronas de cualquier capa sólo se conectan a una pequeña región de la capa que la precede (Shea & Nash, n.d.). Además, este tipo de redes tienen como operación principal la convolución de capas anteriores con un número de pesos, previo a la función de no linealidad (Kriegeskorte, 2015). Esto permite reducir la cantidad de parámetros que se generan para calcular en cada capa, incluso para arquitecturas complejas con múltiples capas. La arquitectura de una CNN consiste en múltiples capas ocultas con tareas específicas según el tipo de capa. En la Figura 19 se presenta una arquitectura que permite ejemplificar el

método que utiliza una CNN para realizar la tarea de clasificación. Esta se divide en dos partes, las dos primeras capas se enfocan en realizar la extracción de características y la capa final realiza la clasificación de estas. Las capas que cumplen con la extracción del vector de características suelen comenzar con dos tipos alternos de capas, denominadas convolucionales y de agrupamiento o *pooling*. La capa completamente conectada permite realizar la clasificación de forma similar a las redes neuronales (Mane & Kulkarni, 2020).

Figura 19

Arquitectura básica de una red neuronal convolucional donde las entradas son imágenes



La capa de convolución se considera la primera en realizar la extracción de características de las imágenes que ingresan a la red. Con el objetivo de preservar la relación entre los pixeles de una imagen, se realiza el aprendizaje mediante pequeños recuadros. Para realizar los cálculos de convolución se toman dos señales o imágenes de dos dimensiones como entradas. Una es utilizada como entrada y la otra es el filtro o *kernel*, la multiplicación o convolución de estas matrices se produce para obtener la señal de salida. La operación matemática que define la convolución entre dos funciones (f y g) se observa en la Ecuación (**12**) (Patel & Patel, 2020).

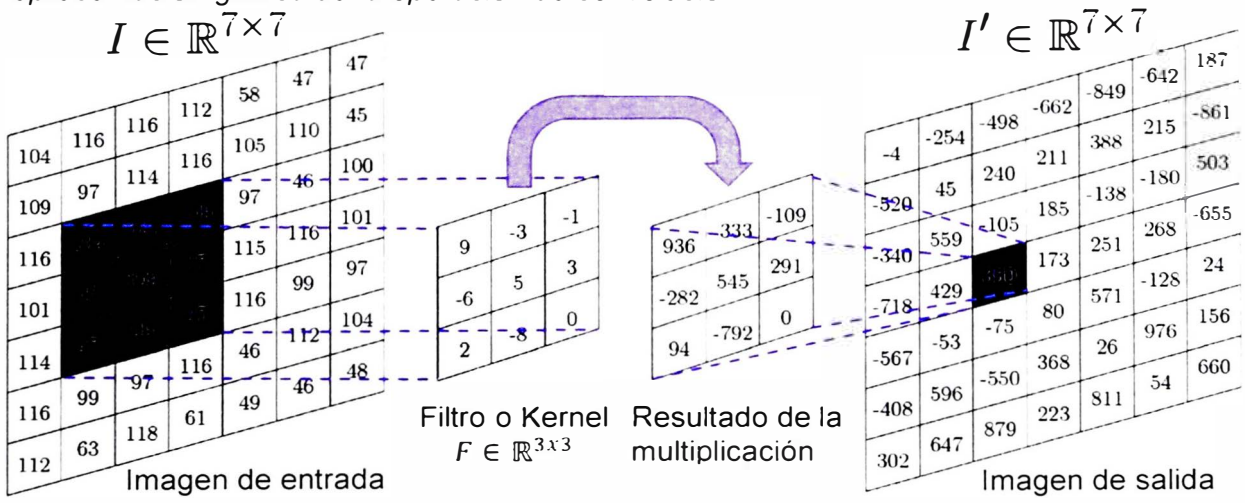
$$(f * g) = \sum_{j=1}^{m} g_{(j)} \cdot f_{(i-j+\frac{m}{2})} \qquad (12)$$

Un *kernel* es un filtro lineal de imagen definido como un elemento $F \in \mathbb{R}^{k_w \times k_h \times d}$, en donde k_w representa el ancho del filtro, k_h representa la altura y d representa la cantidad de canales de profundidad de la entrada. En la Figura 20 se observa la forma en la que se realiza la operación de convolución entre una imagen de entrada $I \in \mathbb{R}^{w \times h \times d}$ y el filtro F para obtener la imagen de salida I'. La imagen de salida presenta únicamente un canal y cada píxel se obtiene al calcular la multiplicación punto a punto de cada elemento del filtro con su correspondiente elemento de entrada. luego los valores se suman para obtener el resultado correspondiente al área que incluye el filtro. En la Ecuación (13) se presenta la explicación matemática de la operación antes presentada (Thoma, 2017).

$$I'_{(x,y)} = \sum_{i_x=1-\left\lfloor \frac{k_w}{2} \right\rfloor}^{\frac{k_w}{2}} \sum_{i_y=1-\left\lfloor \frac{k_h}{2} \right\rfloor}^{\frac{k_d}{2}} \sum_{i_c=1}^{d} I_{(x+i_x,\ y+i_y,\ i_c)} \cdot F_{(i_x,i_y,i_c)} \qquad (13)$$

Figura 20

Representación gráfica de la operación de convolución

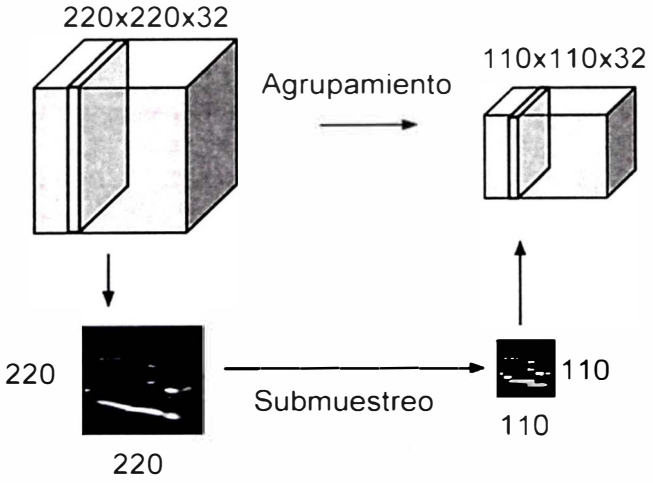


Las siguientes capas en la arquitectura de la CNN son las de agrupamiento (*pooling*), donde se realiza un proceso de discretización basado en muestras. El objetivo de utilizar estas capas es disminuir la dimensionalidad del mapa de caracteristicas de entrada, como imágenes o matrices de salida provenientes de capas ocultas. Esto se logra al reducir su tamaño, lo que permite suponer sobre el conjunto de caracteristicas que incluye cierta subregión asociada (Lee et al., 2016). La operación de agrupamiento consiste

principalmente en dos etapas, primero realiza el escaneo del mapa de características y agrupa los datos por subregiones. La segunda etapa consiste en aplicar la operación de submuestreo, la cual permite mantener parte de la información agrupada de cada canal de características, omitiendo el resto con una longitud o paso de muestreo fijo (Sun et al., 2017). Aplicando ambas etapas, el proceso de submuestreo plantea la reducción de dimensiones como se observa en la Figura 21. En el caso de la imagen 2D se reduce a la mitad en sus ejes y para el agrupamiento se cumple lo mismo sin reducir profundidad. Con esto se elimina información irrelevante y se elige una característica global que la represente (Akhtar & Ragavendran, 2020). Esta reducción en dimensiones conlleva también la reducción de conexiones neuronales, por lo que disminuye la cantidad de parámetros y mejora la eficiencia computacional, además de hacer al algoritmo más robusto ante la invariancia traslacional (Chen et al., 2017).

Figura 21

Proceso de muestreo utilizado en las capas de agrupamiento o pooling.

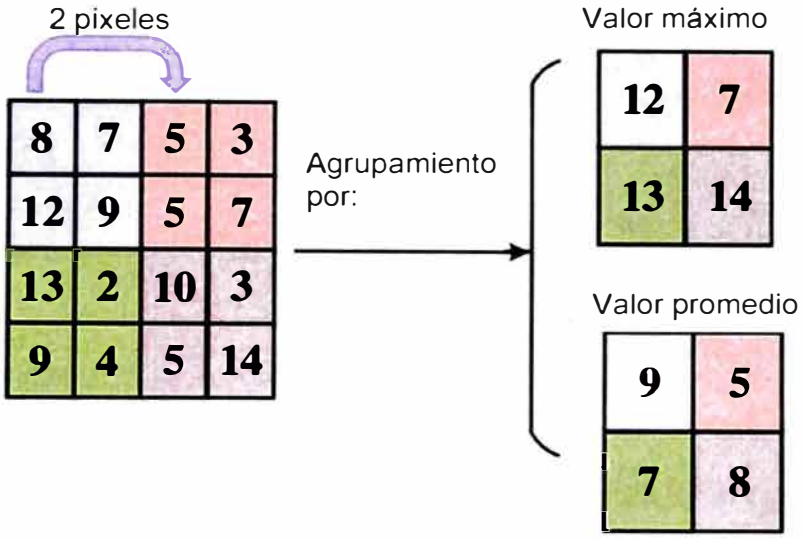


Estas capas incluyen dos parámetros que permiten modificar su comportamiento, los cuales son el tamaño del filtro (F) y el paso de muestreo (S). Las subregiones antes mencionadas son definidas por el valor de F, por ejemplo, una agrupación con un filtro 2x2 se observa en la Figura 22, donde las regiones de cada color tienen las mismas dimensiones que el filtro. Las subregiones pueden estar superpuestas o no dependiendo del valor de S, ya que este define la cantidad de pixeles desplazados entre regiones. En la

Figura 22 el paso de muestreo tiene un valor de 2, de forma que se salta desde el primer pixel hasta el tercero para la nueva región (Akhtar & Ragavendran, 2020).

Figura 22

Representación de los tipos de operaciones de agrupamiento con un filtro de 2x2 y paso de muestreo de 2

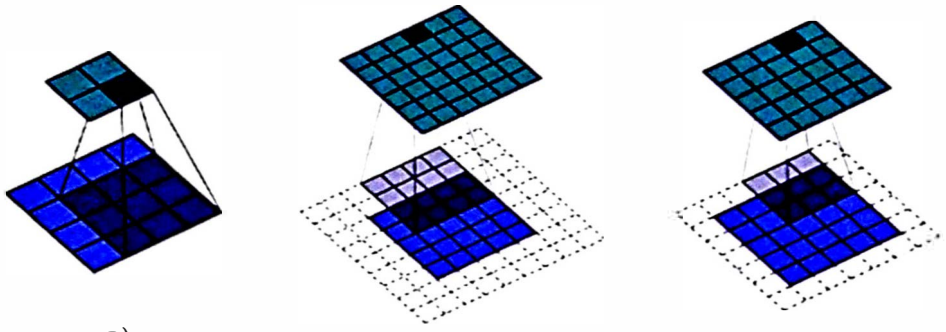


Para extraer la información se debe elegir un valor global que represente a un grupo de características, por lo que se hace uso de diversos métodos y los más comunes son el agrupamiento por valor máximo y por valor promedio. La función de valor promedio usa los datos que se encuentran en cada región y calcula el promedio de estos. Por otro lado, la función de valor máximo seleccionará el valor más alto dentro de la región, considerándolo como el valor global. Un ejemplo de esto se observa en la Figura 22, donde los valores correspondientes a cada color de la región varían dependiendo del método utilizado para el agrupamiento. Si bien estas capas permiten mejorar la eficiencia del algoritmo, se debe considerar la posibilidad de perder información que pertenece a los bordes de la imagen. La información solo puede ser analizada si el filtro la incluye (Albawi et al., 2017). Con el objetivo de asegurar la relevancia de estos datos se define el concepto de relleno o *padding*, el relleno es la cantidad de pixeles que se añaden a una imagen (matriz de entrada) cuando es procesada por el núcleo de una CNN. Añadir relleno permite que el *kernel* inicie brindando más información a los bordes de la región de entrada, logrando así

que toda la información y características ocultas en los bordes de la entrada aparezcan en la salida (El-Amir & Hamdy, 2019).

Figura 23

Representación gráfica de la implementación de relleno con valor de: a) 0, b) 2 y c) 1



a) b) c)

En la Figura 23 se presenta la función de agrupamiento con relleno nulo en "a", por lo que se utiliza la imagen de entrada directamente. En la figura "b" el valor para el relleno es de 2, por esta razón se observa como en los bordes se agregan 2 pixeles extra a la matriz que pasa a la capa de agrupamiento. De esta misma forma la figura "c" agrega 1 píxel extra en los bordes, ya que el relleno tiene el valor de 1.

Mediante las capas de la etapa de extracción de características se realiza la transformación de la entrada de dimensiones $w \times h \times d$ a un mapa de características de dimensiones $w' \times h' \times d'$. El mapa de características debe ser transformado en un vector de una sola columna con dimensión $1 \times m$, donde m se calcula utilizando la Ecuación (14). El resultado es denominado vector de características e ingresa como entrada a la primera capa de la etapa de clasificación de características (Basha et al., 2020).

$$m = w' \times h' \times d' \qquad (14)$$

La etapa de clasificación contiene diversas capas completamente conectadas, donde los pesos e información se actualizan de neurona en neurona, tomando como base el funcionamiento de las redes neuronales. Las capas tienen la particularidad de que cada neurona se encuentra conectada a la activación de las neuronas anteriores, y su activación es calculada con la Ecuación (5) como en una red multicapa (Sakib et al., 2018). Dado que

la salida de las capas convolucionales son características de alto nivel, añadir una capa totalmente conectada es un método sencillo para aprender las combinaciones no lineales de las características. Aunque la mayoría de las características de las capas convolucionales y de agrupación podrían ser suficientes para la tarea de clasificación, las combinaciones de dichas características pueden ser incluso mejores (Springenberg et al., 2015). Con el objetivo de definir la clasificación de las características procesadas se utiliza como capa final una función de activación, por ejemplo, "*Softmax*" o "*ReLU*", la cual devuelve como resultado la clase a la que pertenece la entrada. Como se presentó en la Ecuación (**4**). *ReLU* posee la característica de no activar todas las neuronas al mismo tiempo, sino que solo se activan aquellas cuya salida de la transformación lineal sea superior a 0. Por otro lado, la función *softmax* es descrita como una combinación de múltiples funciones sigmoide como la presentada en la Figura 15. En esta función se aplica el criterio de que una función sigmoide retorna valores entre 0 y 1, los cuales pueden ser tomados como probabilidades de que el dato pertenezca a una clase (Gupta, 2020).

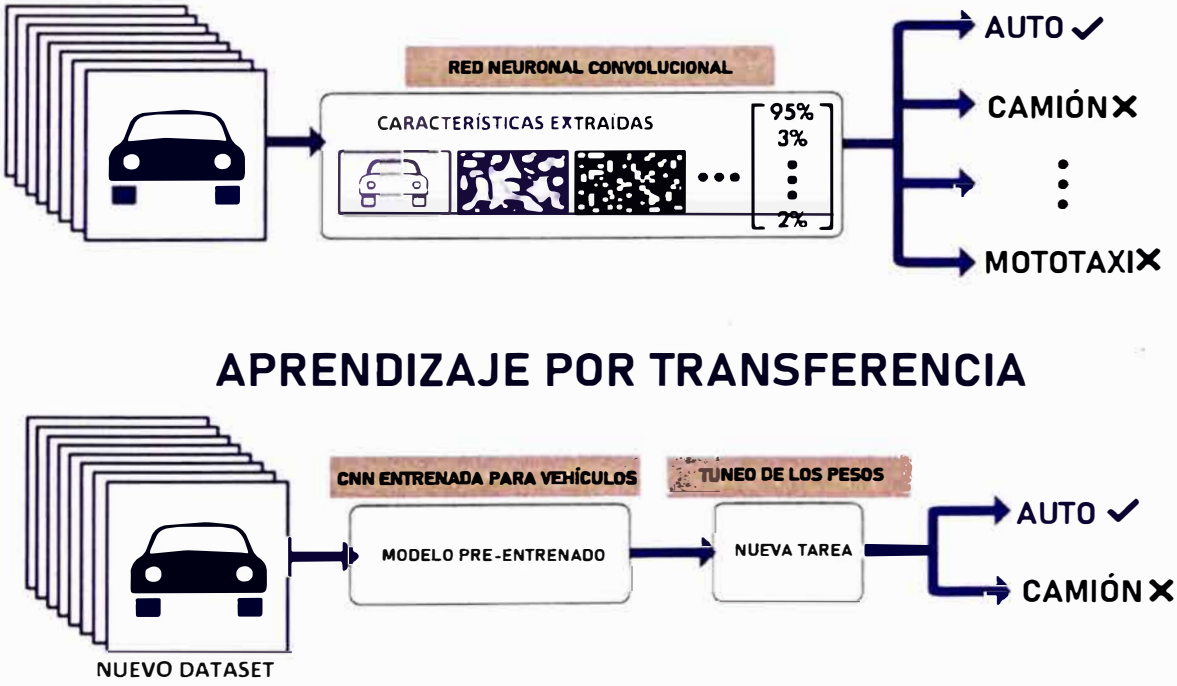
7. **Transfer Learning o Aprendizaje por transferencia (AT).** El aprendizaje por transferencia es una técnica de aprendizaje profundo mediante la cual se entrena y desarrolla un modelo para una tarea y luego se reutiliza en una segunda tarea relacionada (Hussain et al., 2019). Esta técnica surge al investigar la forma en la que el conocimiento funciona cognitivamente, resulta que el conocimiento puede ser transferido al desarrollar tareas con cierta relación, con el objetivo de mejorar el rendimiento. La definición formal del AT es la siguiente, dado un dominio fuente D_F y una tarea de aprendizaje T_F, un domino objetivo D_O y una tarea de aprendizaje T_O. El aprendizaje por transferencia tiene como objetivo mejorar el aprendizaje de una función objetivo f_O en el dominio objetivo D_O, mediante el uso del conocimiento de la tarea T_F en el dominio D_F (Pan & Yang, 2009). A manera de ejemplo, una persona puede aprender a conducir una moto (T_O) basándose en su conocimiento de manejo de bicicleta (T_F), donde ambos dominios son iguales al ser la conducción de vehículos de dos ruedas. Esto no excluye la posibilidad de aprender a

conducir moto sin saber manejar bicicleta, pero utilizar ese conocimiento previo reduce el esfuerzo requerido para entrenar y aprender. En el campo de las CNN, esta definición se aplica a nivel de parámetros. se sabe que entrenar y aprender los parámetros de una red desde cero requiere de gran gasto computacional por el tiempo que tarda y gran cantidad de datos anotados para lograr un correcto rendimiento (Kim et al., 2022). Estas dificultades son fácilmente resueltas con el uso del AT, ya que durante varios años se han desarrollado modelos complejos en arquitectura y con conjuntos de datos amplios. El AT permite transferir los conocimientos de una red pre entrenada, es decir, su capacidad para extraer características particulares y aplicarlas en una tarea similar pero no equivalente.

La mayoría de las CNN enfocadas en clasificación suelen tener en su última capa múltiples clases. Una forma de aplicar el AT es eliminar la última capa de la estructura vista en la Figura 19 y reemplazarla por una nueva, adaptada a la clase que se necesite. Luego, se entrena esta nueva capa utilizando un nuevo conjunto de imágenes. En la Figura 24 se observa gráficamente este proceso (Vrbančič & Podgorelec, 2020).

Figura 24

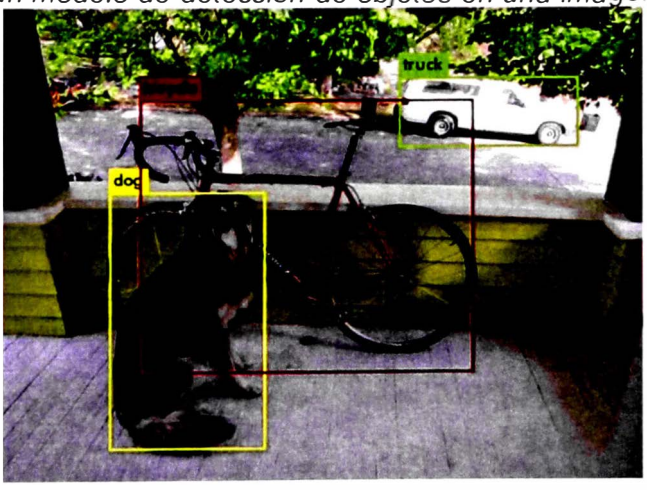
Funcionamiento del aprendizaje por transferencia para tareas de clasificación



8. **Detección de objetos**. La técnica de *Transfer Learning*, es comúnmente utilizada para realizar la tarea de detección de objetos. Esta tarea resulta un reto sencillo para el ojo humano, un niño de pocos meses es capaz de reconocer objetos comunes, sin embargo, enseñarle a realizar esto a una computadora ha sido un reto para el campo de la visión por computador (Zaidi et al., 2022). Precisamente la detección de objetos plantea brindar un completo entendimiento de la información en las imágenes, ya que la tarea no requiere únicamente clasificar imágenes, sino también estimar de forma precisa el objeto y su ubicación (Felzenszwalb et al., 2009). Para lograrlo, los modelos de detección de objetos tienen en esencia dos subtareas. La primera se encarga de localizar objetos en una imagen, mediante la delimitación de un recuadro que los encierre, este es denominado "*bounding box*". Estos recuadros son los que se presentan en la Figura 25 con diversos colores por objeto. La segunda consiste en la clasificación de los objetos encerrados, para ello se definen clases con las que se entrena el modelo. El resultado de esta clasificación se observa en la Figura 25 como una etiqueta con el nombre de la clase a la que el objeto pertenece (Rajeshwari et al., 2019).

Figura 25

Funcionamiento de un modelo de detección de objetos en una imagen



La necesidad de realizar estas dos subtareas genera que el flujo de trabajo de un modelo de detección de objetos tradicional conste de tres procesos: selección de la región de interés, extracción de características y clasificación (Z. Q. Zhao et al., 2019).

a. ***Selección de la región de interés.*** Este proceso se realiza debido a la posible existencia de múltiples objetos alrededor de la imagen, que pueden tener diversos tamaños y formas. Por esta razón resulta necesario realizar un mapeo de toda la imagen en busca de zonas con información que pueda pertenecer a un objeto, mediante el uso de una ventana multiescalar. De esta forma se obtienen los posibles *bounding box* de cada objeto.

b. Extracción de características. Luego de encontrar las regiones, se necesita extraer las características de los objetos en su interior para ser clasificados y etiquetados. Debido a la complejidad de las posibles iluminaciones y fondos de las imágenes es complicado desarrollar un modelo que describa de forma precisa gran variedad de objetos.

c. ***Clasificación.*** Con las características extraídas se hace uso de algoritmos capaces de distinguir la clase a la que estas pertenecen. En esta etapa se utiliza un clasificador para diferenciar al objeto de otros para los que ha sido entrenado, utilizando la jerarquía de la información obtenida.

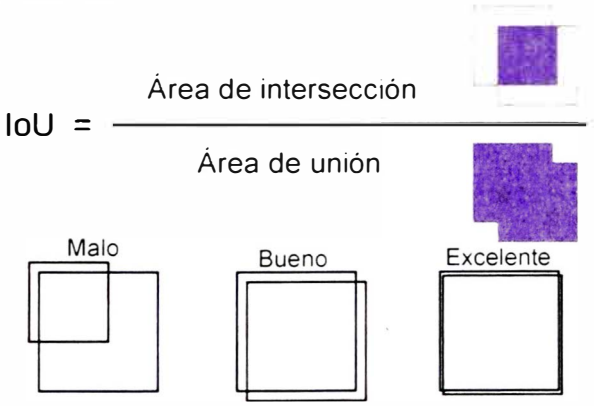
Diferentes redes se han desarrollado a lo largo de las últimas décadas, cada una con distintas arquitecturas y ventajas. Estas se suelen dividir en dos tipos, las de una y dos etapas. Los modelos de dos etapas realizan los tres procesos de forma que se eligen los posibles lugares para las regiones de interés con un algoritmo, para luego clasificar los objetos en su interior utilizando otro algoritmo. Por otro lado, el de una etapa realiza todos los procesos mediante una única estructura, un modelo que tiene una sola etapa es el denominado YOLO (*You Only Look Once*) (Rajeshwari et al., 2019).

9. YOLOv8. Este modelo ha sido ampliamente utilizado durante los últimos años para el desarrollo de sistemas de visión artificial. Esto se debe a que, dentro de la diversidad de algoritmos, YOLO ha destacado por su equilibrio entre la velocidad de procesamiento y su precisión, lo cual permite realizar detecciones de forma rápida y confiable. La velocidad del modelo proviene de la capacidad de la red para recibir una

imagen de entrada y en una sola etapa generar los resultados de la detección (Jiang et al., 2022). El modelo fue presentado como una solución a la necesidad de realizar detección de objetos en tiempo real, en la conferencia de visión artificial y reconocimiento de patrones del año 2016 (Redmon et al., 2016). La arquitectura de esta primera versión consistía en 24 capas convolucionales, seguidas por dos capas completamente conectadas. El modelo divide la imagen en celdas y luego predice múltiples cuadros delimitadores en cada una. El algoritmo de supresión no-máxima (NMS) elige aquellos cuadros que tengan el mayor valor de intersección sobre unión (IOU) con el recuadro real (Jamtsho et al., 2020). El IOU representa la relación entre el área de intersección y el área de unión de los cuadros limitadores predichos y el real, de esta forma, se indica mediante un valor el grado de superposición entre estos cuadros. En la Figura 26 se observa la división que se realiza para obtener dicha relación, así como el significado de las áreas de unión e intersección. En la parte inferior se observa que mientras más superposición exista, conlleva una predicción más precisa (Terven, 2023).

Figura 26

Representación gráfica del cálculo del IOU

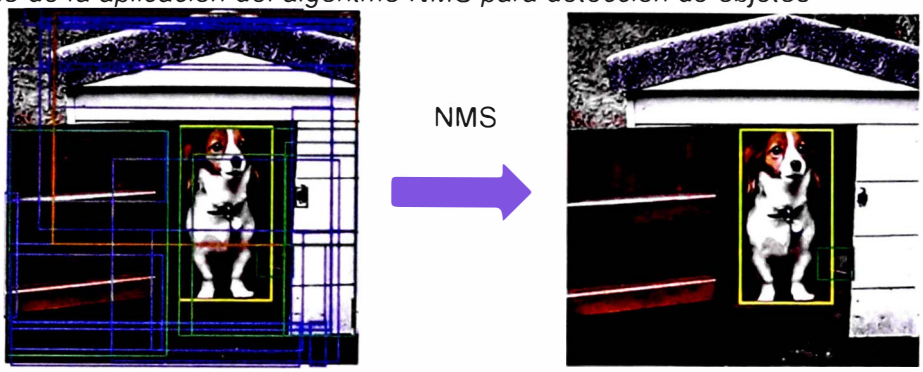


Este valor es utilizado por el algoritmo NMS, que tiene como objetivo reducir el número de cuadros superpuestos y mejorar la calidad de la detección. Yolo genera múltiples posibles cuadros alrededor de un objeto, cada uno con un valor de confianza distinto, precisamente este y el IOU son parámetros que permiten al algoritmo NMS

discriminar entre los cuadros irrelevantes y redundantes para ser filtrados como se observa en la Figura 27.

Figura 27

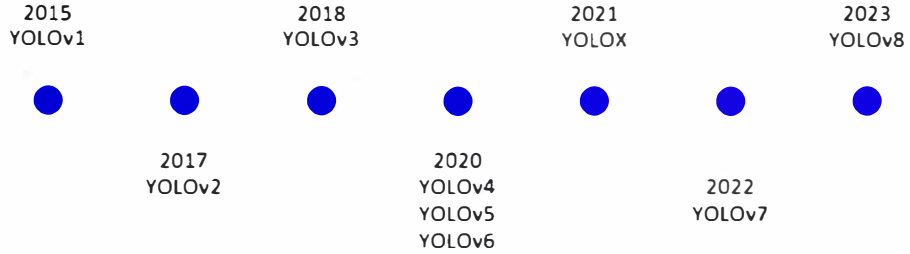
Resultado de la aplicación del algoritmo NMS para detección de objetos



Esta primera versión del modelo fue mejorada con el pasar de los años para aumentar la eficiencia y precisión del modelo. En la Figura 28 se observa que desde el año 2015 se han creado nuevas versiones del modelo que agregaban bloques convolucionales con nuevas funcionalidades.

Figura 28

Línea de tiempo de la creación de los modelos de YOLO



La segunda versión del modelo introdujo la idea de cuadros de anclaje (*anchor box*), los cuales eran *bounding box* con tamaños predefinidos para formas tipicas de objetos. En la tercera versión se propuso la utilización de regresión para predecir las coordenadas del cuadro delimitador, junto con un aumento en el número de capas convolucionales en la red central, dando lugar a la arquitectura Darknet-53. Esta arquitectura incorporó conexiones residuales para añadir información de capas anteriores a las salidas de las capas.

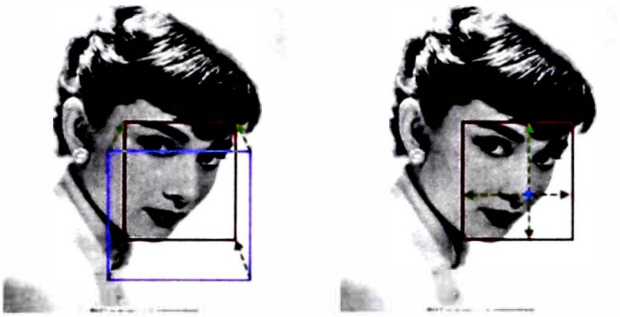
Al aumentar la complejidad de las arquitecturas de los modelos se definieron tres partes; la columna vertebral, el cuello y la cabeza. La columna del modelo o "*backbone*" se

encarga de realizar la extracción de características más importantes de la imagen de entrada, típicamente será una red neuronal convolucional entrenada con gran cantidad de imágenes. Las características poseen jerarquía según la capa; las de bajo nivel representan bordes y texturas, las de mayor nivel partes de objetos obtenidas en capas profundas. El cuello o "*neck*" actúa como intermediario entre la columna y la cabeza. Su función es la de agregar y refinar las características que provienen del *backbone*, con un particular enfoque en mejorar la información espacial y semántica en diferentes escalas. La cabeza o "*head*" será la parte final del modelo, ya que realiza la predicción utilizando las características de las anteriores etapas. Al procesar la información se generan múltiples predicciones para cada objeto y mediante un procesamiento final, como NMS, se filtran los cuadros con menor precisión. Para el caso de YOLO, la cabeza consiste en más de una subred que permitan realizar las tareas de clasificación, localización y segmentación. En la cuarta versión se utilizó como columna la arquitectura Darknet-53, al cual se le agregaron conexiones parciales entre etapas *(cross-stage connections*, CSPNet) que ayudan a la reducción de cálculos del modelo sin perder precisión. CSPNet separa el mapa de características de la capa base en dos partes, una de las cuales pasa por un bloque denso y una capa de transición; la otra parte se combina con el mapa de características transmitido a la siguiente fase (Terven, 2023; Wang et al., 2020). Para el cuello se utilizaron las estructuras denominadas agrupación piramidal espacial (*spatial pyramid pooling*, SPP) que son modificaciones del clásico agrupamiento colocados de forma concatenada. La quinta versión de YOLO fue lanzada por *Ultralytics*, esta empresa realizó importantes modificaciones en la complejidad de la arquitectura de YOLOv4. Además, desarrolló esta versión en *Pytorch* en lugar de usar *Darknet*, por lo que permitió la integración con tarjetas gráficas que mejoren la velocidad del modelo. Se agregaron capas extra de convolución a la columna Darknet53 con CSP, también mejoraron las capas SPP al definir el tamaño de los mapas de características y ahora se denominaría SPPF (SPP Fast). La versión de YOLOX planteó la idea de utilizar una arquitectura "*anchor-free*", haciendo referencia a los

cuadros de anclaje. Con esta arquitectura se detectan los objetos sin necesidad de predefinir formas o tamaños de cuadros, en cambio se utiliza un punto central o una región del objeto que permita definir detecciones correctas. Luego, se intenta predecir las distancias hacia los cuatro lados del cuadro delimitador (S. Zhang et al., 2020). En la Figura 29 se presenta de forma gráfica la diferencia de los modelos que utilizan cuadros de anclaje (*anchor based*), donde se predefine la forma del cuadro azul y debe ajustarse al rojo. Por otro lado, el modelo YOLOX calcula el centro y las flechas representan las direcciones en la que se predicen los lados del cuadro.

Figura 29

Diferencia entre modelos "anchor based" (a) y "anchor-free" (b)

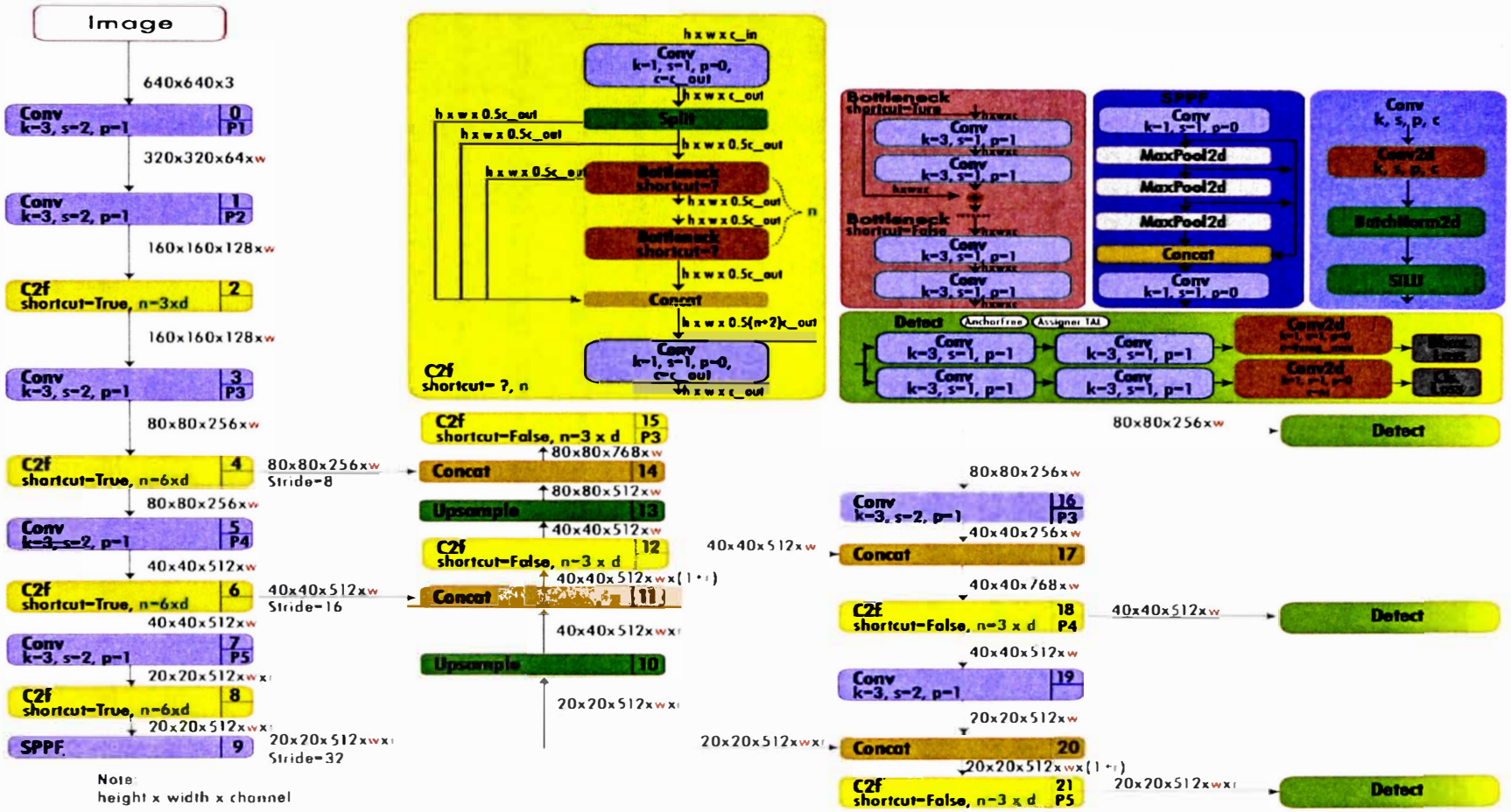


(a) (b)

Debido a la posibilidad de una desviación entre la confianza en la clasificación y la precisión en la localización del objeto, el modelo YOLOX dividió la cabeza en dos partes: una para la clasificación y otra para la regresión. El modelo YOLOv8 fue desarrollado por *Ultralytics* y fue publicado en el año 2023. Este posee una columna similar a la de YOLOv5, con la variación en las capas CSP, ahora llamadas módulo C2f. Este módulo utiliza capas denominadas *bottleneck* para reducir las dimensiones de la entrada, con conexiones parciales entre etapas y dos capas de convolución. Además, incluye una capa SPPF y funciones de activación sigmoideas para determinar la probabilidad de que un cuadro contenga un objeto, así como funciones de activación *softmax* para la probabilidad de que un objeto pertenezca a una clase especifica (Terven, 2023).

Figura 30

Modelo YOLOv8 y sus bloques "C2f" en amarillo, "Bottleneck" en marrón, "SPPF" en azul, "Conv" en celeste y "Detect" en verde.



La Figura 30 presenta la arquitectura del modelo YOLOv8, la columna es una concatenación de bloques "Conv" y "C2f", además de una capa SPPF. El bloque "Conv" está compuesto por una capa de convolución, una capa de normalización y una función de activación denominada SiLU (Elfwing et al., 2018). El bloque *bottleneck* es un conjunto de convoluciones con algunas capas que reciben información de la entrada. También se observa la estructura del bloque SPPF, que consta de una concatenación de capas de agrupamiento por valor máximo que envían información a capas posteriores. El bloque para la detección está desacoplado y posee dos ramas de convolución, una para el cuadro delimitador (Bbox) y la otra para la clase del objeto (Cls).

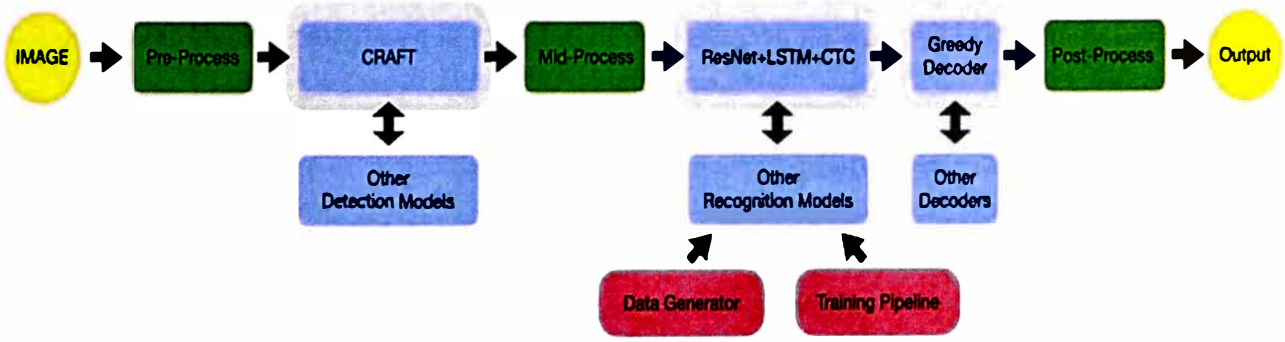
10. Reconocimiento óptico de caracteres. El OCR es una tecnología que permite transformar diferentes archivos; como documentos escaneados, archivos PDF o imágenes digitales, en datos editables por un sistema. Esto se consigue al simular la capacidad de los ojos humanos para recibir una imagen y comprender los caracteres que se encuentran en esta. Se hace uso de mecanismos ópticos y algoritmos de aprendizaje profundo para realizar las diversas tareas que comprenden el reconocimiento de caracteres. Sin embargo, las imágenes suelen presentar distorsiones en los bordes de los caracteres y variaciones en la iluminación, lo que dificulta el trabajo de los algoritmos (Mithe et al., 2013). Existen diferentes enfoques para realizar esta tarea, entre estos se tienen aquellos de una sola etapa como son TesseractOCR y EasyOCR. En anteriores estudios se han realizado comparaciones de la precisión de estos algoritmos para la tarea de ALPR. en (Awalgaonkar et al., 2021) se demostró que el uso del aprendizaje profundo para reconocer caracteres que utiliza EasyOCR lo convierte en el más robusto.

EasyOCR es un algoritmo de código abierto para reconocimiento óptico de caracteres compatible con más de 80 idiomas. El sistema recibe una imagen para realizar la primera etapa de procesamiento, en la que se elimina el ruido existente para facilitar el reconocimiento. La siguiente etapa se encarga de encontrar la región de la imagen que contenga texto, por lo que se hace uso del modelo CRAFT (*Character-Region Awareness*

For Text detection). El modelo de redes neuronales convolucionales diseñado genera dos puntuaciones con las que se define de forma precisa la localización del texto. Una es la puntuación de la región de caracteres, la cual localiza los caracteres de forma individual. Por otro lado, la puntuación de afinidad se utiliza para agrupar cada carácter en una única instancia o palabra (Louis March & Glaveanu, 2020). Luego de encontrar las regiones de interés, continúa una etapa para la extracción de características haciendo uso de las capas convolucionales y de agrupamiento del modelo ResNet. Con el objetivo de procesar las representaciones secuenciales de características obtenidas en la anterior etapa convolucional, se utiliza una red neuronal recurrente. La red planteada utiliza múltiples capas de bloques de memoria a largo plazo (*Long Short-Term Memory*-LSTM), la cual permite captar información a largo plazo, al recordar y olvidar ciertos datos de manera selectiva. De esta forma, si en la etapa de CNN se obtienen datos que codifiquen la palabra "exce_ente", la letra no detectada por ruido en la imagen será completada, ya que el modelo aprendió que anteriormente se ha detectado varias veces la palabra "excelente". Por último, se lleva a cabo la decodificación y conversión a texto de las palabras detectadas, utilizando el algoritmo CTC (Clasificación Temporal Conexionista). Las últimas etapas permiten mejorar los resultados obtenidos de forma que sean consistentes según el idioma y otras características configurables (Elyousfi, 2023). La arquitectura completa se observa en la Figura 31.

Figura 31

Arquitectura del algoritmo Easy OCR



Dentro de la documentación está definida la clase que usa como base EasyOCR denominada *READER*. Esta es creada con ciertos parámetros de entrada, los cuales se observan a continuación (JaidedAI, 2022).

- lang_list (list): lista del idioma que desea reconocer, por ejemplo ['ch_sim','en']. La lista de códigos de idioma soportados está aqui.
- gpu (bool, string, default = True): Activa la GPU
- model_storage_directory (string, default = None): Ruta al directorio para los datos del modelo. Si no se especifica, los modelos se leerán de un directorio definido por la variable de entorno EASYOCR_MODULE_PATH (preferido), MODULE_PATH (si está definido), o ~/.EasyOCR/.
- download_enabled (bool, default = True): Activa la descarga si EasyOCR no puede localizar los archivos modelo
- user_network_directory (bool, default = None): Ruta a la red de reconocimiento definida por el usuario. Si no se especifica, los modelos se leerán de MODULE_PATH + '/user_network' (~/.EasyOCR/user_network).
- detector (bool, default = True): Carga el modelo de detección en memoria
- recognizer (bool, default = True): Carga el modelo de reconocimiento en memoria

11. Distancia de Levenshtein. Dado que los algoritmos de OCR brindan como resultado cadenas de caracteres, se requiere de una métrica que permita analizar la eficiencia de estos. Para verificar el grado de similitud entre dos cadenas, se hace uso de la distancia entre ellas, para lo cual, Levenshtein desarrolló un método que permite realizar una comparación basada en la cantidad de operaciones para transformar una cadena en la otra (Fernández Marcellán, 2021). Las operaciones primitivas propuestas en este método son:

- Inserción: Cuando se agrega un carácter a la cadena.
- Supresión: Cuando se elimina un carácter de la cadena.
- Reemplazo: Cuando se sustituye un carácter por otro.

De este modo, la precisión de un texto generado por OCR con respecto al texto verdadero se calcula mediante la distancia de edición de Levenshtein. Esta distancia es igual al número mínimo de operaciones primitivas que se necesitan para corregir el texto de salida del OCR. La precisión de los caracteres viene determinada por la Ecuación (15), donde m es la cantidad de caracteres de la palabra y e es la distancia de edición de Levenshtein (Alghamdi et al., 2016).

$$d = \frac{m - e}{m} * 100 \qquad (15)$$

2.2 Marco conceptual

En el siguiente capítulo se presentan conceptos importantes a considerar para la comprensión de la metodología utilizada para esta tesis.

1. **Python**. Python es un lenguaje de programación moderno que admite los estilos de programación orientado a objetos, funcional e imperativo. Es ideal para los principiantes por su legibilidad y facilidad de uso. Python es ante todo un lenguaje de scripting, pero puede compilarse en binario legible por ordenador. La ventaja de todo esto es que puedes escribir programas en menos líneas de código que un programa equivalente en C/C++ o Java (Kelly, 2019).

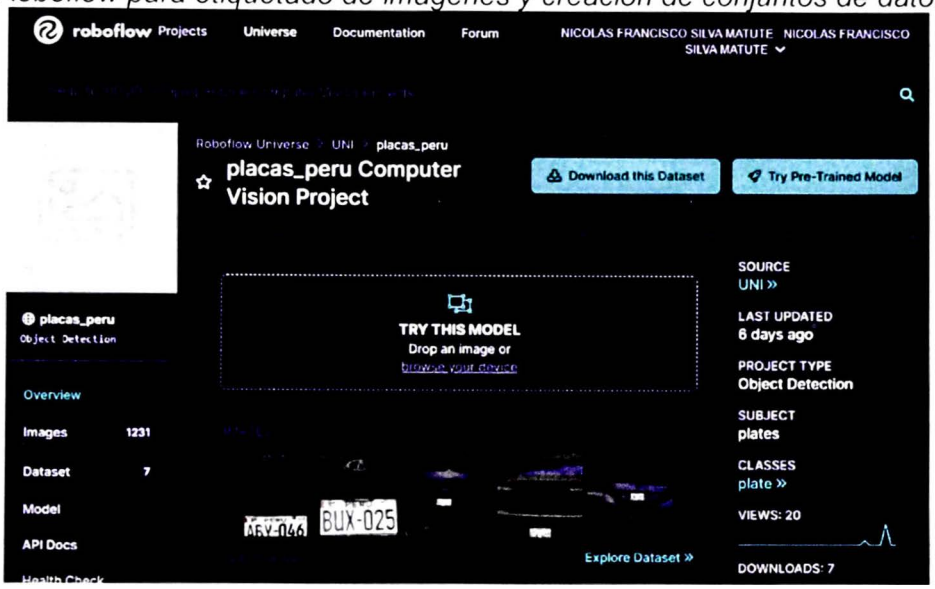
2. **Google Colab**. Google Colaboratory es un entorno de desarrollo integrado, de manera abreviada se le denomina Google Colab. Este es un recurso gratuito proporcionado por la empresa Google, la cual permite desarrollar un código de programación en el lenguaje Phyton para facilitar la búsqueda de las raíces de las ecuaciones. Esta herramienta está alojada en Google, sin necesidad de un ordenador con gran capacidad de memoria, ya que se ejecuta en la nube. Esta función tiene algunas bibliotecas ya instaladas, y los archivos se guardan luego en Drive, lo que facilita y agiliza su configuración (Alves et al., 2020; Gunawan et al., 2020).

3. **Roboflow.** Es una plataforma de inteligencia artificial diseñada para realizar tareas para construir y desplegar modelos de aprendizaje profundo. Posee métodos

sencillos para la subida de imágenes, así como su etiquetado y procesado en formatos para el entrenamiento de modelos (Dwyer et al., 2022).

Figura 32

Plataforma Roboflow para etiquetado de imágenes y creación de conjuntos de datos



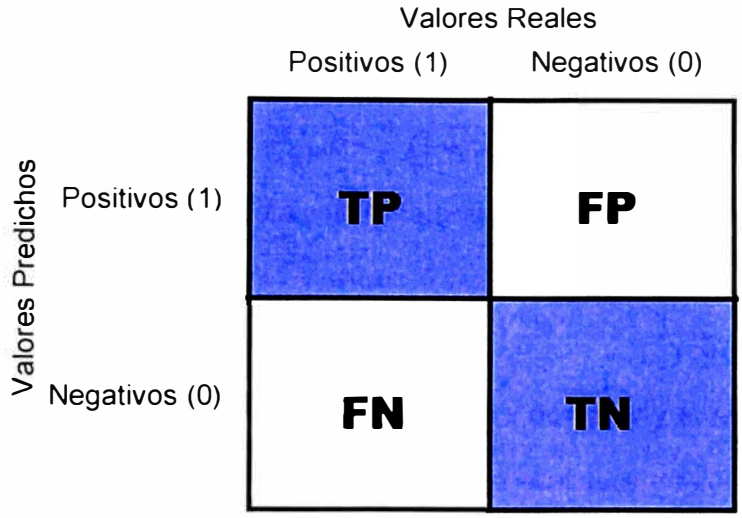
4. **Pytorch.** Es un entorno de aprendizaje automático de código abierto basado en el lenguaje de programación Python y la biblioteca Torch. Esta biblioteca de ML de código abierto que se utiliza para crear redes neuronales profundas y está escrita en el lenguaje de scripting Lua. Es una de las plataformas preferidas para la investigación en aprendizaje profundo. El marco está construido para acelerar el proceso entre la creación de prototipos de investigación y el despliegue (Ketkar et al., 2021).

5. **Matriz de confusión.** Esta herramienta tiene como objetivo evaluar errores en problemas de clasificación. La matriz de confusión codifica la capacidad de un modelo para brindar predicciones correctas, tomando el número de elementos clasificados erróneamente para cada par definido como: {clase a la que pertenece un elemento, clase a la que se clasifica un elemento de manera incorrecta} (Beauxis-Aussalet & Hardman, 2014).

Una matriz de confusión de tamaño n x n asociada a un modelo muestra la clasificación predicha y la real, donde n es el número de clases diferentes.

Figura 33

Ejemplo de matriz de confusión



A cada objeto se le asigna una etiqueta binaria que define de forma certera si pertenece (+) o no (-) a cierta clase. Luego el modelo generará una predicción que indica si considera que el objeto es correcto (+) o no (-) a cierta clase.

TP: Es el número de objetos clasificados positivos de manera correcta.

FN: Es el número de objetos clasificados negativos de manera incorrecta.

FP: Es el número de objetos clasificados positivos de manera incorrecta.

TN: Es el número de objetos clasificados negativos de manera correcta.

6. **Recall.** Este parámetro es también denominado tasa de verdaderos positivos, este define el porcentaje de objetos que el modelo clasifica como positivos de manera correcta (Yacouby & Axman, 2020).

$$r = \frac{TP}{TP + FN} \qquad (16)$$

7. **Precisión.** Este parámetro define el porcentaje de predicciones positivas que el modelo realiza de manera correcta (Yacouby & Axman, 2020).

$$p = \frac{TP}{TP + FP} \qquad (17)$$

8. **F1 score.** Este parámetro permite combinar en un único valor los resultados de la precisión y el recall. Para calcularlo se aplica la media armónica, como se observa a en la Ecuación (18).

$$F_\beta = (1 + \beta^2)\frac{p * r}{r + \beta^2 p} = \frac{(1 + \beta^2)TP}{(1 + \beta^2)TP + \beta^2 FN + FP} \qquad (18)$$

Donde el valor de beta será 1 para este parámetro (Goutte & Gaussier, 2005).

9. **Librerías de Python.** Para el desarrollo de sistemas de visión por computador las librerías más utilizadas.

- Pandas: La ciencia de datos, el análisis de datos y el aprendizaje automático han adoptado en gran medida Pandas, una biblioteca de código abierto en Python. Se basa en el paquete Numpy, que es una biblioteca de matrices multidimensionales (Agrawal et al., 2021).
- Numpy: Numerical Python es una librería que proporciona herramientas para el manejo de arreglos multidimensionales, así como, funciones para el procesamiento de estos arreglos (Vinjit et al., 2020).
- Opencv: Es una librería enfocada en resolver problemas de visión por computador, Este fue diseñado para tener gran eficiencia computacional, de forma que se aplique en sistemas en tiempo real. Tiene el objetivo de brindar las herramientas necesarias para realizar procesamiento de imágenes de forma sencilla y funcionalidades más complejas (Bradski & Kaehler, 2008).

Capítulo III. Desarrollo del trabajo de investigación

En el siguiente capítulo se abordará diferentes características relacionadas a la investigación realizada.

3.1 Metodología del trabajo de investigación

3.1.1 Tipo

La tesis es del tipo aplicada, ya que se ha centrado en el desarrollo de un sistema de detección de placas vehiculares utilizando algoritmos de aprendizaje profundo con análisis de *metadata*. El objetivo principal fue diseñar e implementar un sistema capaz de extraer metadatos de vehículos en imágenes capturadas en tiempo real, por cámaras de seguridad o dispositivos de vigilancia. Mediante el uso de estos metadatos se plantea identificar de forma única a cada vehículo. Para lograr esto, se utilizan redes neuronales convolucionales y técnicas avanzadas de procesamiento de imágenes. El enfoque de este proyecto va en línea con investigaciones previas que demuestran la eficacia del aprendizaje profundo en tareas de reconocimiento visual y detección de objetos. Un estudio relevante en este campo se encuentra en, "*A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector*", en el cual los autores hacen uso de técnicas de aprendizaje profundo y modelos de detección de objetos para obtener las placas vehiculares en tiempo real (Laroca et al., 2018). Adicionalmente por las necesidades de seguridad ciudadana se consideró oportuno seguir la línea expuesta en, "*Intelligent Stolen Vehicle Detection using Video Sensing*". Esta investigación desarrollada en Australia concluye que el uso de características propias de cada vehículo, como el color y su placa vehicular, permite identificar vehículos robados (Al-Hmouz & Challa, 2007).

La metodología propuesta para este proyecto se basa en la recopilación de un conjunto de datos de imágenes de placas vehiculares, desarrollo del algoritmo de extracción de metadatos, el entrenamiento de la red neuronal convolucional utilizando el conjunto de datos y la evaluación del rendimiento del sistema mediante pruebas

exhaustivas. Además, se exploraron técnicas de mejora de imágenes y de aumento de datos para incrementar la capacidad de generalización del modelo entrenado. Se espera que este proyecto de investigación aporte avances significativos en el campo de la identificación vehicular mediante el uso de metadatos, así como en la detección de placas vehiculares utilizando técnicas de aprendizaje profundo. Además, el desarrollo de un sistema preciso y eficiente podría tener aplicaciones prácticas en áreas como la seguridad vial, el control de tráfico y la identificación de vehículos en entornos de vigilancia.

3.1.2 Nivel

La investigación en cuestión se enmarca en un enfoque explicativo, donde se abordó el problema de estudio, se llevó a cabo un análisis exhaustivo y se propusieron metodologías potenciales para su resolución. Además, se ejecutaron la implementación y las pruebas correspondientes, las cuales provocaron modificaciones en el comportamiento del sistema objeto de estudio. Cabe destacar que la contribución significativa de este trabajo radica en la introducción de un nuevo sistema de seguridad que abordó de manera efectiva el problema de la identificación de vehículos. Paralas pruebas del sistema diseñado se instaló una cámara ip, instalada en la puerta N°5 de la Universidad Nacional de Ingeniería, de forma remota se realiza la extracción de metadatos e identificación vehicular. El diseño se fundamentó con una adecuada base teórica sobre aprendizaje profundo y procesamiento de imágenes, que permitió el logro de modificar de modo positivo la identificación vehicular y en esencia dar solución a la realidad problemática, que fue la razón fundamental de la presente investigación.

3.1.3 Etapas de la investigación

Con el objetivo de realizar la identificación de vehículos mediante la extracción de *metadata*, placa vehicular y color, se propuso las siguientes etapas para la metodología.

1. Desarrollo del marco teórico.

- Conceptos y definiciones presentados en el anterior capítulo

2. Desarrollo del conjunto de imágenes.

- Evaluar y analizar los conjuntos de imágenes de uso libre que contengan placas vehiculares.
- Obtención de imágenes para *dataset* de placas peruanas.
- Acondicionamiento y etiquetado de placas. (Roboflow)

3. Identificar y definir el algoritmo de aprendizaje profundo a utilizar.

- Creación de modelo de detección de objetos propio.
- Aplicación de Transfer Learning (Yolov8).

4. Ensayo del algoritmo de identificación.

- Implementación de la segmentación de placas.
- Implementación del procesamiento de imágenes y obtención de color de vehiculo.
- Aplicación de algoritmo de OCR (EasyOCR).
- Almacenamiento de datos para identificación de vehiculos.

5. Análisis y discusión de resultados.

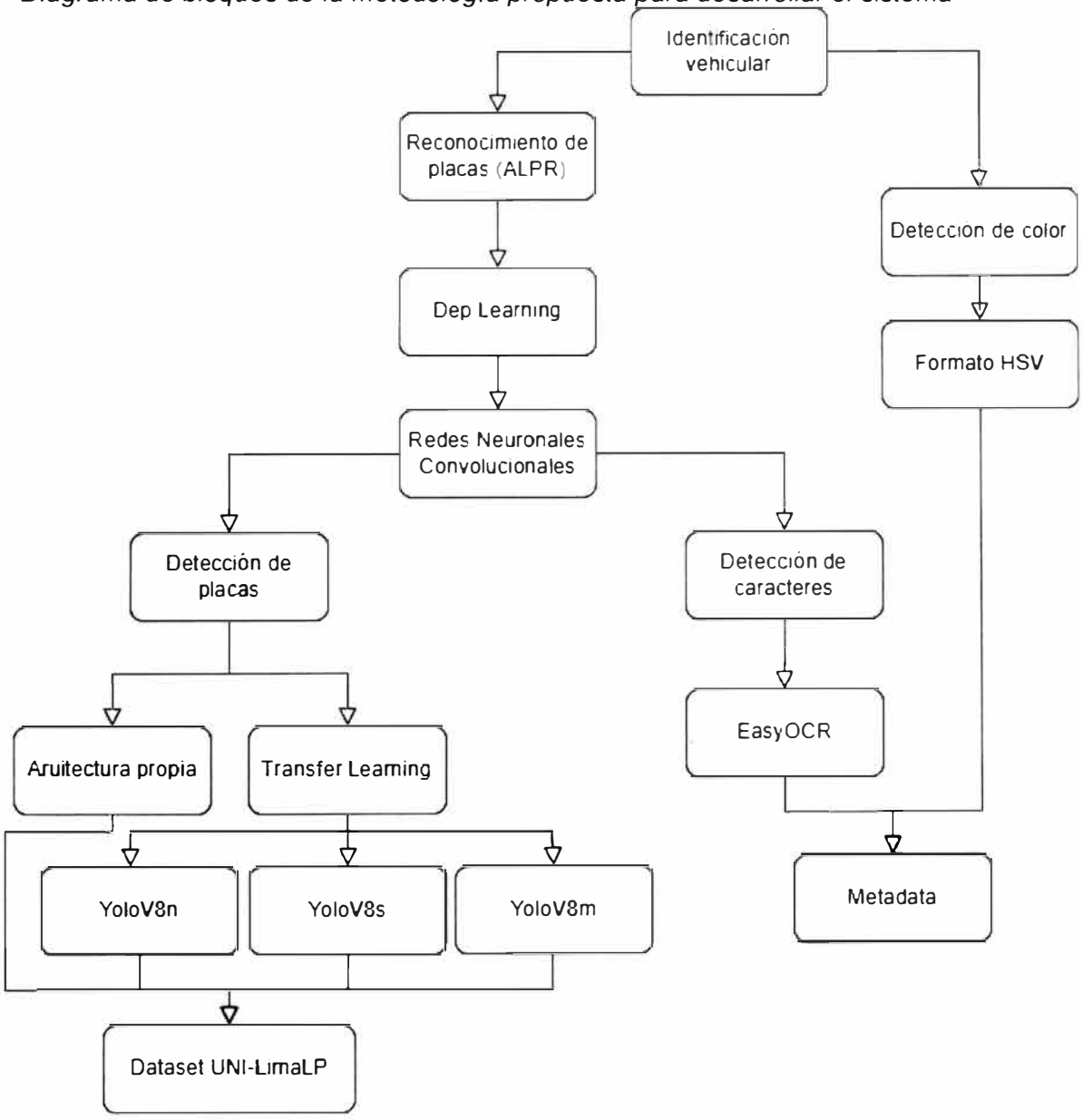
- Los resultados son presentados en el capitulo correspondiente

Como primera etapa se realizó la obtención del conjunto de imágenes para el entrenamiento del modelo de detección de objetos. Al revisar los conjuntos de datos públicos se llegaron a algunas conclusiones que permitieron definir la metodología para el desarrollo del *dataset*. Analizando las investigaciones previas, se concluyó que utilizar *dataset* de placas extranjeras reduciria la eficiencia y precisión del modelo, esto además se comprobó mediante algunas pruebas. Por esto se desarrolló un sistema que permitió capturar imágenes de vehiculos de videos, utilizando el lenguaje de programación Python y sus librerías. Para el etiquetado de imágenes se utilizó la plataforma Roboflow con lo que se obtuvo el archivo necesario para el entrenamiento del modelo. En la siguiente etapa se creó una red neuronal convolucional propia, al obtener las métricas de dicha red se notó la necesidad de utilizar la técnica de TL para utilizar modelos pre entrenados que brinden mejor precisión. El modelo utilizado fue YoloV8, como modelo con amplio desarrollo en el ámbito de la detección de objetos, además posee compatibilidad con el uso de GPU. Es

esta etapa se obtiene como resultado el recorte de la imagen original para procesar únicamente el área en el que se encuentra la placa vehicular. De esta forma se hace más sencilla la tarea que realiza la etapa de detección de caracteres usando EasyOCR, obteniendo como resultado la placa vehicular. Para la etapa de detección de color se utilizó el análisis del formato HSV de los pixeles de la imagen con lo que se obtiene un resultado del color del vehículo. Finalmente, con los datos extraídos se plantea almacenarlos para identificar cada vehículo con sus metadatos.

Figura 34

Diagrama de bloques de la metodología propuesta para desarrollar el sistema



3.1.4 Obtención de imágenes para creación del dataset

Al realizar el análisis de *dataset* públicos extranjeros, se concluyó que dadas las características particulares de las placas vehiculares de Perú es necesario crear un nuevo conjunto de imágenes que incluyan variedad de estas placas. El método presentado en anteriores investigaciones para generar estos *dataset* mayormente era la toma manual de fotos a vehículos, sin embargo, resulta ser un proceso muy lento y laborioso. Por esta razón, se plantea una metodología que automatice la obtención de imágenes. Se plantea el uso de videos de la plataforma *YouTube*, en los cuales se graba el recorrido realizado por un conductor por las calles y avenidas de Lima. En la Figura 35 se observa un *frame* del video titulado "lima 4K-Driving Downtown-Peru", en el cual se observa la vista delantera de un vehículo. Se observa en cada *frame* del video varios vehículos con sus respectivas placas, sin embargo, la nitidez de la placa depende de la distancia a la que se encuentra.

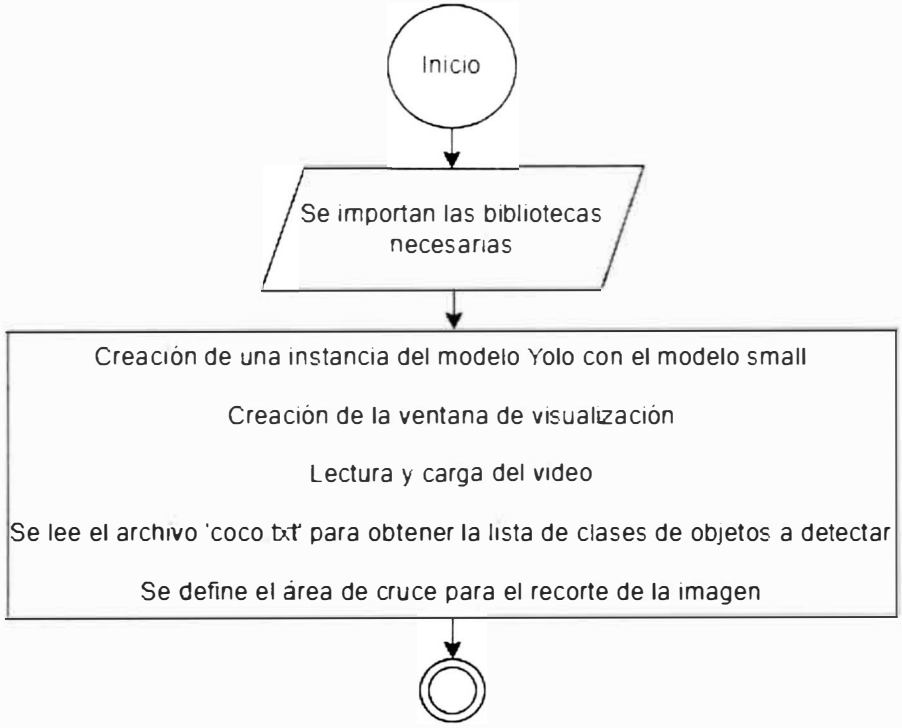
Figura 35

Vídeo de la vista frontal de un vehículo recorriendo Lima subido a YouTube.

Para el desarrollo de esta etapa es necesario un algoritmo que pueda extraer partes del *frame* en el que se encuentre la placa, de forma que la distancia brinde una correcta nitidez. Se desarrolló un sistema que reciba un video y sea capaz de detectar vehículos para recortarlos de cada *frame*, dado que cada vehículo en su parte trasera contiene la placa. En la Figura 36 se observa el diagrama de flujo del sistema que se desarrolló en Python. En esta primera parte se importan las librerías que contienen las funciones y objetos que se utilizaron. Además, se inicializan instancias, objetos y variables que permiten recibir datos y procesarlos.

Figura 36

Primera etapa del diagrama de flujo para la obtención del conjunto de imágenes



Las librerías que se han utilizado para este sistema se presentan en el código de la Figura 37. Las primeras tres librerías permiten el manejo de imágenes, datos y arreglos matriciales respectivamente.

Figura 37

Líneas de código para importar bibliotecas y archivos en Python

```
1    import cv2
2    import pandas as pd
3    import numpy as np
4    from ultralytics import YOLO
5    from tracker import *
6    from datetime import datetime
7    import os
```

La libreria *Ultralytics* permite hacer uso de modelos de YOLOv8, así como sus funciones para la detección de objetos. *Tracker* es una clase dentro de un archivo de Python que permite realizar el seguimiento de la posición de un objeto en una grabación o en tiempo real. Las últimas dos librerías permiten manejar datos de tiempo y archivos

Figura 38

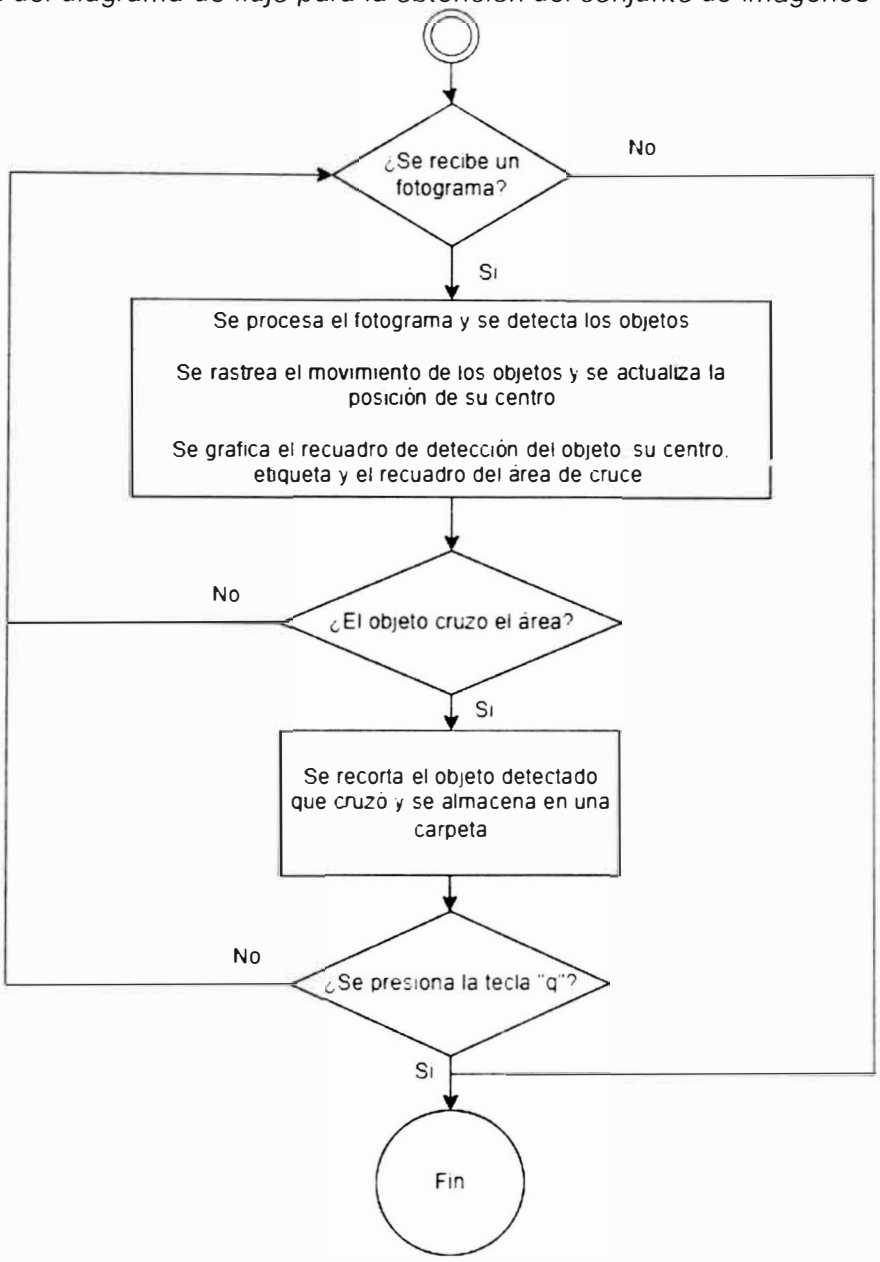
Etapa de inicialización de variables y lectura de archivos

```
10   model=YOLO('yolov8s.pt')
22   cap=cv2.VideoCapture('videoplayback.mp4')
23
24
25   my_file = open("coco.txt", "r")
26   data = my_file.read()
27   class_list = data.split("\n")
28   #print(class_list)
29   count=0
30   tracker=Tracker()
31   area=[(54,436),(41,449),(317,494),(317,470)]
32   area_c=set()
```

En la Figura 38 se muestra la creación del objeto *model*, en el cual se carga el modelo "yolov8s" mediante la función YOLO y el archivo de extensión ".pt". Para la lectura del video a procesar se utiliza la función *VideoCapture*, que almacena el archivo en una variable denominada *cap*. El archivo "coco.txt" contiene las clases que el modelo es capaz de detectar y se almacenan en una lista para definir los objetos que se requiere filtrar. Se inicia el objeto "*tracker*" y la variable "área" que define un recuadro que será utilizado luego

Figura 39

Segunda etapa del diagrama de flujo para la obtención del conjunto de imágenes



A continuación, se utiliza un bucle donde se ejecutan una serie de instrucciones como las que se observan en la Figura 39. Este bucle está controlado por la recepción de fotogramas, estos provienen del video que se carga es la etapa anterior, en caso no haya más fotogramas se rompe el bucle y termina el algoritmo. En el primer bloque se procesa el fotograma recibido realizando un redimensionamiento para adecuarse al modelo que se utiliza. Haciendo uso la instancia "*model*" creada previamente, se implementa la función

model.predict(frame) que devuelve los resultados de la predicción del modelo para detectar vehículos. Estos resultados se almacenan en la variable *results* como un arreglo. El primer elemento del arreglo contiene la información importante del objeto detectado. Estos datos se almacenan en un *DataFrame* de la librería *Pandas* y se convierten al tipo de datos denominado *float* para su uso en cálculos, como se observa en la Figura 40.

Figura 40

Etapa de extracción de información del objeto detectado

```
52          a=results[0].boxes.data
53          px=pd.DataFrame(a).astype("float")
```

Luego se itera sobre cada fila del *DataFrame* en la variable "px" utilizando un bucle *for*. De forma que se obtienen las coordenadas (x1, y1, x2, y2) del cuadro delimitador, la clase (**c**), y el índice de la clase (**d**). Luego se filtran las clases "*car*", "*motorcycle*" y "*truck*", para ser añadidas a la lista de coordenadas *list*. Con las coordenadas de los objetos almacenados se utiliza la función *update* de la clase *Tracker*, de forma que para cada objeto se define un rastreador para su movimiento sobre el fotograma. El rastreador genera un identificador relacionado con el cuadro delimitador del objeto que se actualiza en cada *frame*. En la Figura 41 se muestra el código para esta implementación.

Figura 41

Actualización del rastreador para cada objeto en la variable "list"

```
68          bbox_idx=tracker.update(list)
```

Con el objetivo de obtener un centroide del objeto, se itera sobre cada cuadro delimitador (bbox) en la variable bbox_idx, del cual se extraen las coordenadas (x3, y3, x4, y4) y el identificador (id). Para observar de forma gráfica el proceso se agrega en cada fotograma los cuadros y centroides calculados por objeto, así como el cuadro definido para el área en la que se realiza la captura.

Figura 42

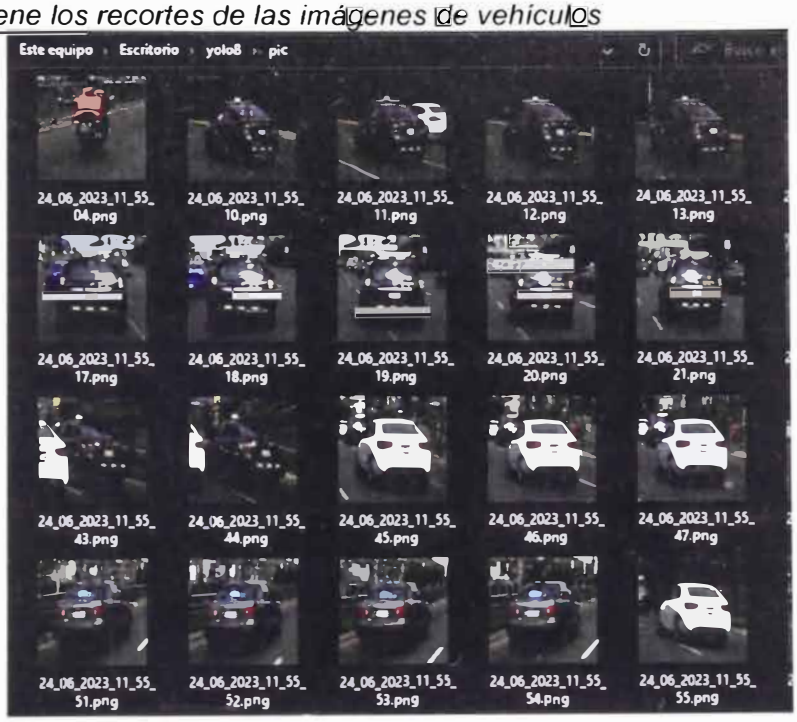
Gráfica del área de cruce en el video

El algoritmo toma la captura del vehículo en el momento que este cruce el área definida, haciendo uso de la función **cv2.pointPolygonTest()**. Si el resultado es mayor o igual a cero, confirma el cruce del vehículo por lo que se procede con el recorte de la región de interés (*crop*) en el fotograma y se guarda como una imagen en una carpeta, utilizando la función **imgwrite()**. Esta función permite definir la carpeta en la que se almacenan las imágenes y darles unas dimensiones específicas. Finalmente, se libera el objeto de captura de video con la función **cap.release()** y se cierran todas las ventanas mostradas con la función **cv2.destroyAllWindows()** si se presiona una tecla.

Este sistema permitió procesar diversos videos, los cuales se eligieron de forma que sean de zonas distintas de Lima y con iluminaciones variables. De esta manera se almacenaron las imágenes de vehículos en carpetas por cada video como se observa en la Figura 43. Luego de procesar cada video se observó la presencia de recortes con el mismo vehículo repetidas veces por lo que se pasó a una etapa de filtrado y limpieza del conjunto de imágenes.

Figura 43

Carpeta que contiene los recortes de las imágenes de vehículos

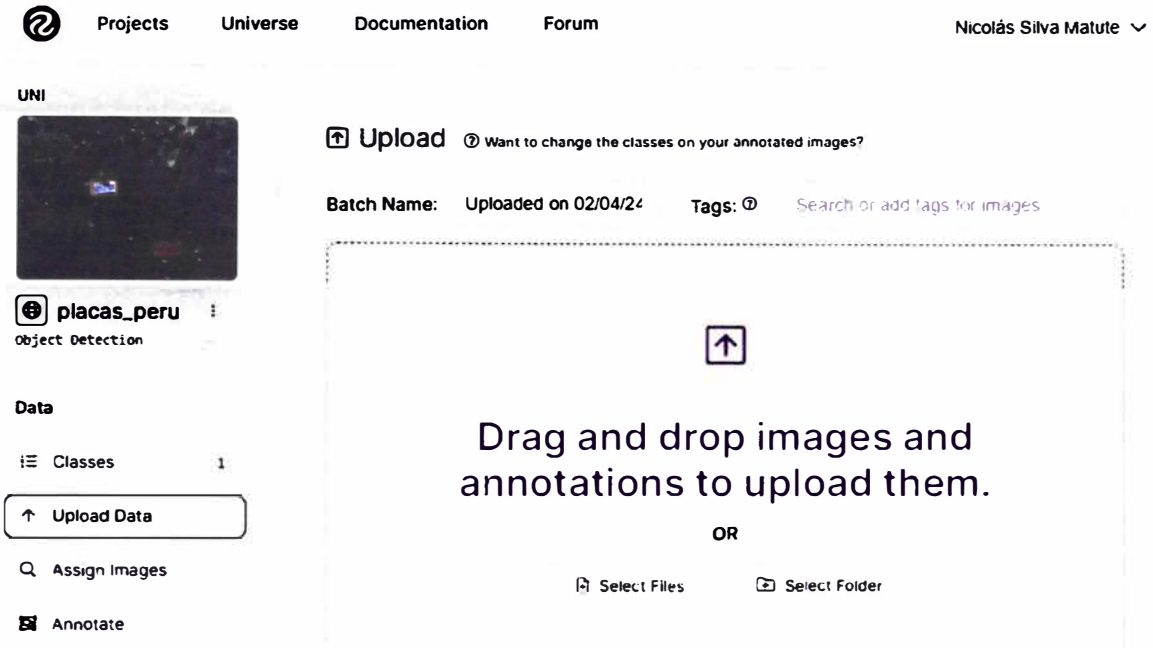


3.1.5 Etiquetado y aumento de datos para el conjunto de imágenes

Mediante el proceso de obtención de imágenes se logró recolectar la cantidad de 3,315 recortes de vehículos, lo cual representaba una cantidad considerable para el modelo de YOLO que se planteó utilizar y seguía los valores mínimos definidos para un conjunto de imágenes para un ALPR. El siguiente proceso fue el del etiquetado de las placas en cada imagen, para lo cual se hizo uso de la plataforma *Roboflow*. En ella se procede a crear un nuevo proyecto denominado "placas_peru". Como se observa en la Figura 44, en este proyecto se tiene la opción de subir las imágenes obtenidas del sistema desarrollado. Otra de las opciones es la visualización de las clases definidas en el conjunto de datos, la pestaña para el etiquetado y para asignar imágenes a personas en un equipo de trabajo.

Figura 44

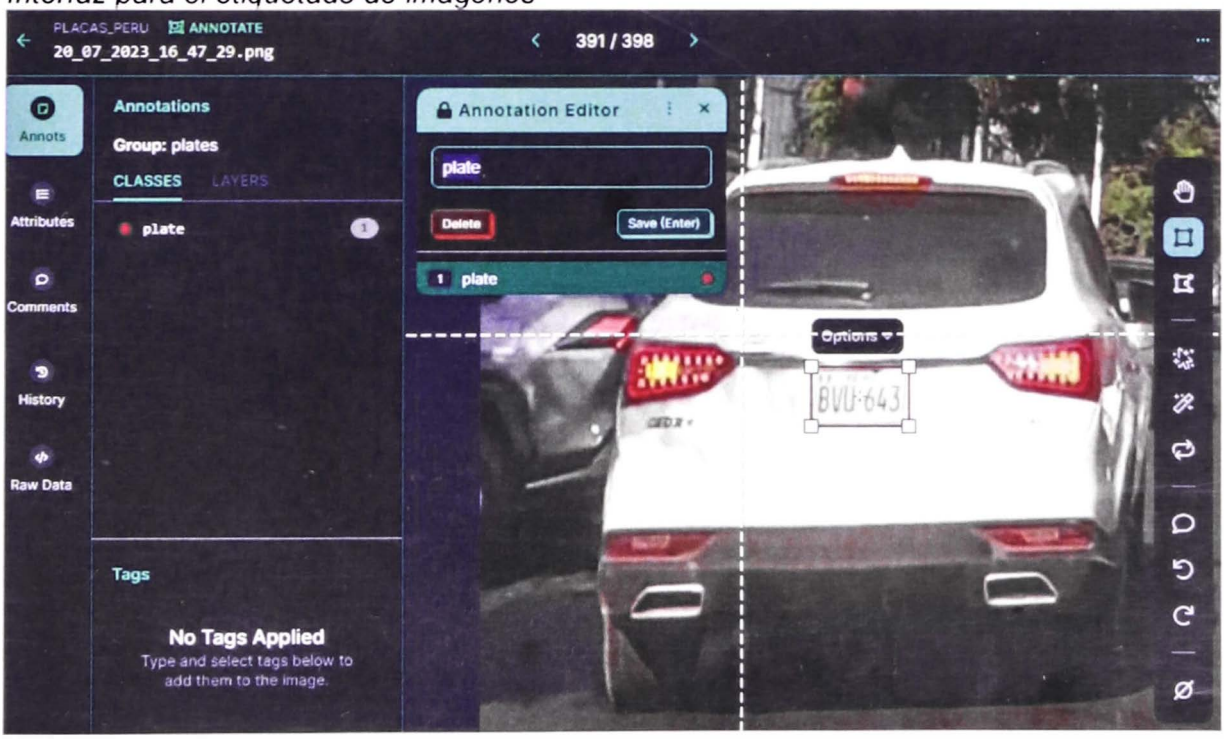
Interfaz para la creación de proyectos en Roboflow



Luego de subir las imágenes se hace uso de las herramientas que presenta *Roboflow* para etiquetar las placas vehiculares. En la Figura 45 se observa la creación de la clase "*plate*" y se encierra en un recuadro cada placa con esta etiqueta.

Figura 45

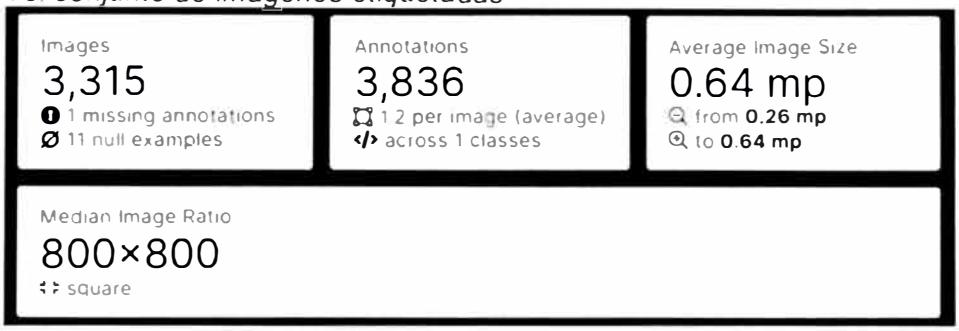
Interfaz para el etiquetado de imágenes



El cuadro delimitador debe encerrar toda la placa incluyendo la palabra "Peru", incluso si la placa se encuentra inclinada. En la Figura 46 se observa que la cantidad de anotaciones fue de 3.836 placas al tener algunas imágenes con más de una placa. El mínimo tamaño de la placa para ser considerada en el etiquetado fue de 0.26 mp, de forma que se tenga cierta nitidez. Por último, para que las imágenes sean compatibles con el modelo se definió el tamaño de 800x800 para las imágenes.

Figura 46

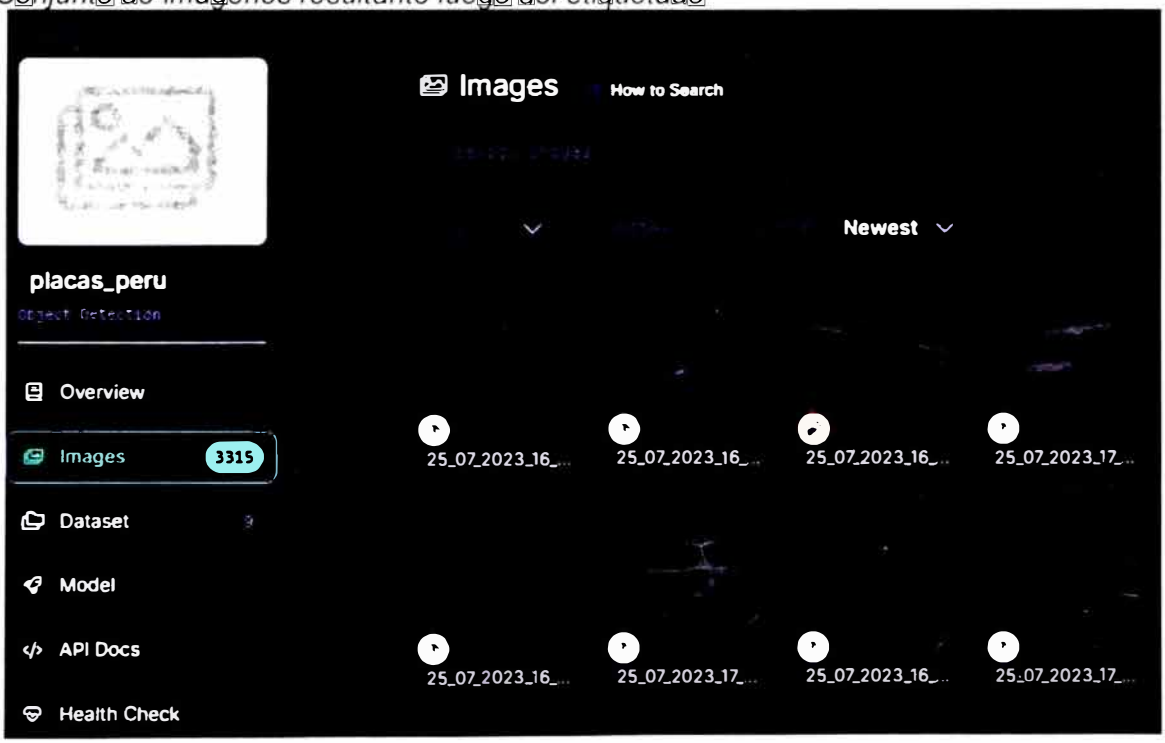
Información del conjunto de imágenes etiquetadas



Luego de realizar el etiquetado de las imágenes, estas se dividen en los subconjuntos para entrenamiento, validación y pruebas como se observa en la Figura 47.

Figura 47

Conjunto de imágenes resultante luego del etiquetado



Esta plataforma permite realizar un aumento de datos antes de finalizar con la creación del conjunto de imágenes. Se decidió únicamente utilizar una variación en el brillo de las imágenes, esto significa que de manera aleatoria se incrementa o reduce el brillo de ciertas imágenes de entrenamiento para que el *dataset* sea más grande. Con este proceso se define el conjunto de imágenes final, que contaba con las cantidades que se observan en la Figura 48, obteniendo un total de 5,634 imágenes para el *dataset* UNI-LimaLP.

Figura 48

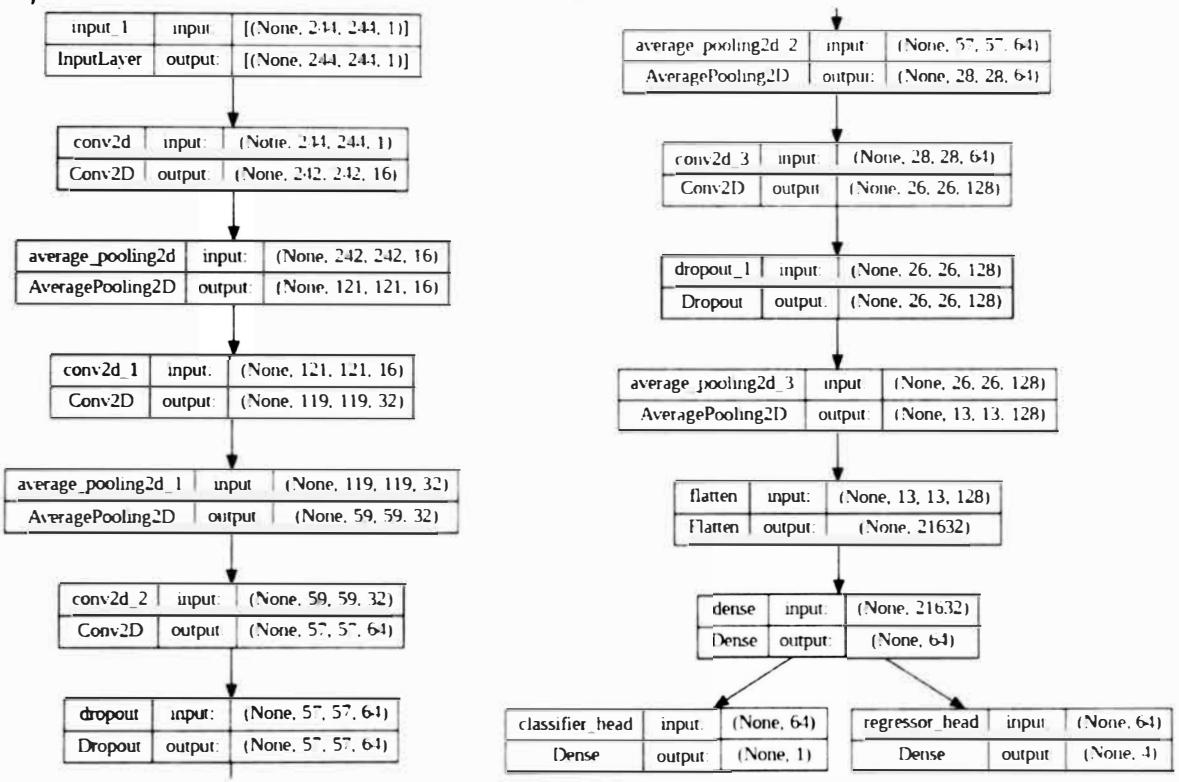
Cantidad de imágenes para entrenamiento, validación y pruebas

3.1.6 Modelos de Deep Learning propuestos

En primera instancia de planteó la creación de un modelo propio utilizando la plataforma de Google Colab, la librería *Tensorflow* y *Keras*. Se utilizó una estructura basada en los conceptos presentados para un modelo de detección de objetos con redes neuronales convolucionales de una etapa. Esta arquitectura consta de capas de convolución concatenadas con capas de agrupamiento por valor medio, en algunas se colocó capas de *Dropout* que se encarga de desactivar ciertas neuronas para mejorar el aprendizaje. La cabeza está desacoplada de forma que la clasificación del objeto y la regresión del cuadro delimitador se realice de manera separadam, como se observa en la Figura 49.

Figura 49

Arquitectura del modelo desarrollado con Tensorflow



Luego de definir la arquitectura, se codifica la función para construcción del modelo. Cada bloque contiene las capas propias de una RNA y se unen usando la función **tf.keras.Model**, en la cual se definen dos salidas, para la clase y el cuadro delimitador. Por otro lado la función **model.compile** permite definir el optimizador para el entrenamiento con la variable *optimizer*.

Figura 50

Función para generación del modelo con sus bloques

```
def build_model(inputs):

    feature_extractor = build_feature_extractor(inputs)

    model_adaptor = build_model_adaptor(feature_extractor)

    classification_head = build_classifier_head(model_adaptor)

    regressor_head = build_regressor_head(model_adaptor)

    model = tf.keras.Model(inputs = inputs, outputs = [classification_head, regressor_head])

    model.compile(optimizer=tf.keras.optimizers.experimental.RMSprop(),
                  loss = {'classifier_head' : 'binary_crossentropy', 'regressor_head' : 'mse' },
                  metrics = {'classifier_head' : 'accuracy', 'regressor_head' : 'mse' })

    return model
```

Al utilizar la función creada es posible visualizar las dimensiones de las capas y sus parámetros mediante la función **model.summary()**, y se obtiene lo presentado en la Figura 51.

Figura 51

Etapa de creación del modelo y sus caracteristicas

```
[ ] model = build_model(tf.keras.layers.Input(shape=(input_size, input_size, 1,)))

    model.summary()

Model: "model"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 244, 244, 1)]	0	[]
conv2d (Conv2D)	(None, 242, 242, 16)	160	['input_1[0][0]']
average_pooling2d (Average Pooling2D)	(None, 121, 121, 16)	0	['conv2d[0][0]']
conv2d_1 (Conv2D)	(None, 119, 119, 32)	4640	['average_pooling2d[0][0]']
average_pooling2d_1 (Avera gePooling2D)	(None, 59, 59, 32)	0	['conv2d_1[0][0]']
conv2d_2 (Conv2D)	(None, 57, 57, 64)	18496	['average_pooling2d_1[0][0]']
dropout (Dropout)	(None, 57, 57, 64)	0	['conv2d_2[0][0]']
average_pooling2d_2 (Avera gePooling2D)	(None, 28, 28, 64)	0	['dropout[0][0]']
conv2d_3 (Conv2D)	(None, 26, 26, 128)	73856	['average_pooling2d_2[0][0]']
dropout_1 (Dropout)	(None, 26, 26, 128)	0	['conv2d_3[0][0]']
average_pooling2d_3 (Avera gePooling2D)	(None, 13, 13, 128)	0	['dropout_1[0][0]']
flatten (Flatten)	(None, 21632)	0	['average_pooling2d_3[0][0]']
dense (Dense)	(None, 64)	1384512	['flatten[0][0]']
classifier_head (Dense)	(None, 1)	65	['dense[0][0]']
regressor_head (Dense)	(None, 4)	260	['dense[0][0]']

```
Total params: 1481989 (5.65 MB)
```

En el siguiente bloque se implementa el entrenamiento del modelo como se observa en la Figura 52, de forma que pueda realizar la validación y obtener resultados. En esta función se define la cantidad de ciclos de entrenamiento, así como, la ruta donde se almacenan las imágenes para validación. Las imágenes son descargadas de *Roboflow*, en el formato que corresponde a modelos de *tensorflow* y se almacenan en una carpeta. Esta carpeta luego es dividida para obtener los conjuntos de entrenamiento y validación.

Figura 52

Etapa de entrenamiento del modelo

```
history = model.fit(train_ds,
                    steps_per_epoch=(len(training_files) // BATCH_SIZE),
                    validation_data=validation_ds, validation_steps=1,
                    epochs=EPOCHS, callbacks=[tensorboard_callback])
```

La métrica que se utiliza en esta etapa es el IoU, el cual permite definir el grado de certeza con la que el modelo predice la posición del cuadro delimitador para las placas. Para obtener esta métrica se desarrolla una función que realice el cálculo como se observa en la Figura 53, considerando la ecuación que la define. La función recibe dos cuadros, luego calcula las coordenadas del vértice superior izquierdo del cuadro de intersección tomando el valor máximo entre las coordenadas "x" e "y" del cuadro A y la del cuadro B. Luego se obtienen las coordenadas del vértice inferior derecho en las variables "xB" e "yB". Con estas variables se calcula el área de intersección entre cuadros y las áreas para cada cuadro. Finalmente, en la variable "iou" se calcula la métrica realizando la división presentada en la Figura 26.

Figura 53

Bloque de código para la obtención del IoU

```
def intersection_over_union(boxA, boxB):
  xA = max(boxA[0], boxB[0])
  yA = max(boxA[1], boxB[1])
  xB = min(boxA[0] + boxA[2], boxB[0] + boxB[2])
  yB = min(boxA[1] + boxA[3], boxB[1] + boxB[3])
  interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
  boxAArea = (boxA[2] + 1) * (boxA[3] + 1)
  boxBArea = (boxB[2] + 1) * (boxB[3] + 1)
  iou = interArea / float(boxAArea + boxBArea - interArea)
  return iou
```

3.1.7 Entrenamiento de YoloV8

Dado que en el ámbito de la detección de objetos el aprendizaje por transferencia ha sido una herramienta ampliamente usada para la obtención de sistemas eficientes, se plantea el uso del modelo YOLOv8 como base para entrenarlo con el nuevo *dataset*. Este conjunto de imágenes seria el creado en esta investigación, que incluye únicamente placas peruanas y fue etiquetado en *Roboflow*.

Figura 54

Descarga del conjunto de imágenes etiquetadas desde la plataforma Roboflow

```
!pip install roboflow

%cd {HOME}/datasets
from roboflow import Roboflow
rf = Roboflow(api_key="••••••••••••••••")
project = rf.workspace("uni-phtjr").project("placas_peru")
dataset = project.version(11).download("yolov8")


/content/drive/MyDrive/2023-1/pruebas_de_yolos/yolo_robo/datasets
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in placas_peru-11 to yolov8: 100% [230750320 / 230750320] bytes
```

En la plataforma Google Colab se implementó el bloque de código que se observa en la Figura 54, para extraer el *dataset* utilizando la libreria propia de *Roboflow*. En la función se definen el nombre del proyecto y la clave que permite acceder al conjunto de datos.

Figura 55

Código para la configuración del entrenamiento con un nuevo dataset para YoloV8

```
%cd {HOME}
 yolo task=detect mode=train model=yolov8m.pt data={data_location}/data.yaml epochs=120 imgsz=800 plots=True

                                                                                                    Python

/content/drive/MyDrive/2023-1/pruebas_de_yolos/yolo_robo
Ultralytics YOLOv8.0.20 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8m.yaml, data=/content/drive/MyDrive/2023-1/pruebas_de_yo
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 18.4MB/s]
2023-06-27 02:44:51.224368: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimiz
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compi
2023-06-27 02:44:52.047660: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find
Overriding model.yaml nc=80 with nc=1

                 from  n    params  module                                       arguments
  0                -1  1      1392  ultralytics.nn.modules.Conv                  [3, 48, 3, 2]
  1                -1  1     41664  ultralytics.nn.modules.Conv                  [48, 96, 3, 2]
  2                -1  2    111360  ultralytics.nn.modules.C2f                   [96, 96, 2, True]
  3                -1  1    166272  ultralytics.nn.modules.Conv                  [96, 192, 3, 2]
  4                -1  4    813312  ultralytics.nn.modules.C2f                   [192, 192, 4, True]
  5                -1  1    664320  ultralytics.nn.modules.Conv                  [192, 384, 3, 2]
  6                -1  4   3248640  ultralytics.nn.modules.C2f                   [384, 384, 4, True]
  7                -1  1   1991808  ultralytics.nn.modules.Conv                  [384, 576, 3, 2]
  8                -1  2   3985920  ultralytics.nn.modules.C2f                   [576, 576, 2, True]
  9                -1  1    831168  ultralytics.nn.modules.SPPF                  [576, 576, 5]
 10                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11           [-1, 6]  1         0  ultralytics.nn.modules.Concat                [1]
```

Luego en la Figura 55, se utiliza la linea de código que define la tarea de detección (task=detect) y el modo entrenamiento, primero se utiliza el archivo "yolov8m.pt" que contiene los pesos del modelo antes del entrenamiento. La "m" (*medium*) hace referencia

a la escala que define la profundidad, ancho y cantidad máxima de canales para la arquitectura de YOLOv8. De la misma forma se realiza el entrenamiento del modelo para las escalas "s" (*small*) y "n" (*nano*).

En la Figura 56 se observan los valores para los parámetros de la arquitectura YOLOv8 según su escala, además, se presentan la variación en la cantidad de capas y parámetros.

Figura 56

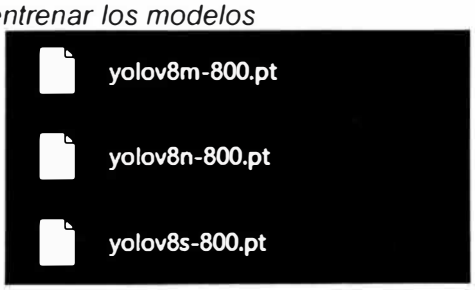
Parámetros y características de las escalas del modelo YOLOv8.

```
n: [0.33, 0.25, 1024] # YOLOv8n summary: 225 layers,  3157200 parameters,
s: [0.33, 0.50, 1024] # YOLOv8s summary: 225 layers, 11166560 parameters,
m: [0.67, 0.75, 768] # YOLOv8m summary: 295 layers, 25902640 parameters,
```

Luego de realizado el entrenamiento el archivo resultante se almacena para su uso en el algoritmo del sistema. Este archivo tiene la extensión ".pt" dado que tiene como base el sistema *Pytorch*, además, contiene los pesos del modelo YOLOv8 para la detección de placas.

Figura 57

Archivos obtenidos luego de entrenar los modelos



3.1.8 Algoritmo de identificación vehicular

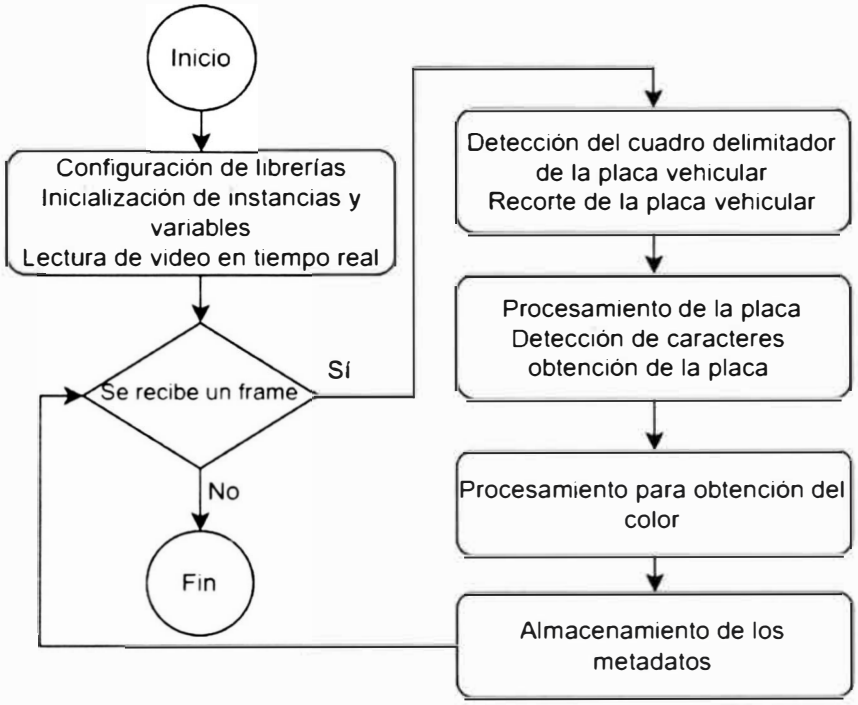
Para cumplir con el principal objetivo de esta investigación, se desarrolló un sistema capaz de extraer *metadata* de vehículos. El algoritmo implementado se observa en la Figura 58, donde se presentan los bloques definidos para cada tarea.

Como primera etapa se definen las librerías a utilizar tanto para el procesamiento de imágenes y el manejo de los modelos de redes convolucionales. Por otro lado, se definen valores predeterminados para ciertas variables, como colores para cuadros

delimitadores o textos, valores umbrales para ciertas métricas de los modelos de detección y tamaños para las ventanas de visualización de resultados.

Figura 58

Algoritmo de extracción de metadatos para identificación vehicular



Hay tres instancias importantes que se inicializan, en la Figura 59 se observan las líneas de código utilizadas en este bloque.

Figura 59

Bloque de inicialización de instancias

```
# Inicialización
read = easyocr.Reader(['en'], gpu=True)
model = YOLO(config['model_path'])
cap = cv2.VideoCapture(config["video_path"])
```

El objeto *read* se crea usando la función *Reader* de la biblioteca de EasyOCR y se configura para hacer uso de una tarjeta gráfica, que optimice su velocidad de procesamiento. Luego, el objeto *model* se implementa utilizando la biblioteca de *ultralytics* para el uso del modelo YOLOv8, el modelo se configura en la variable *model_path* que contiene el archivo de pesos del modelo entrenado. Por último, se crea el objeto *cap* con

la librería OpenCV y la función *VideoCapture*, esta recibe la ruta a una fuente de video que se configura previamente en *video_path*. Con esto se consigue realizar la lectura de las imágenes de video en tiempo real.

El bucle se mantiene funcionando siempre que se reciba un *frame* con el objeto *cap*. Dentro de este bucle, el primer bloque realiza la detección de la placa vehicular. El objeto *model* permite utilizar la función *predict* para almacenar los resultados de la predicción en la variable *results*. Como se observa en la Figura 60, en esta función se incluyen los parámetros de confianza (*conf*) y el umbral de IoU, ambos valores se definen al inicio del algoritmo. La variable *results* contiene diversos valores en un vector, donde el primer elemento posee la opción de utilizar la función *plot* que devuelve automáticamente el *frame* con el recuadro delimitador incluido, además del valor de precisión y la etiqueta. Luego, usando la función *boxes* se almacenan los valores que definen el cuadro delimitador de la placa detectada. Utilizando esta variable se realiza el recorte del *frame* para obtener únicamente el área donde se encuentra la placa detectada.

Figura 60

Bloque para realizar la detección de placa vehicular

```
# -------------------- Prediccion --------------------
results = model.predict(
                            frame,
                            conf=CONFIDENCE_THRESHOLD,
                            iou=NMS_THRESHOLD,
                            max_det=1,)
frame_predicted = results[0].plot()
boxes = results[0].boxes
```

La variable *boxes* permite utilizar la función "xyxy", la cual devuelve las coordenadas de la esquina inferior izquierda y la esquina superior derecha. Utilizando estos puntos se obtiene el ancho (w) y la altura (h), de forma que en la variable "*crop_img*" se almacena el recorte del *frame*, como se observa en la Figura 61. Esto se logra al cambiar las dimensiones de la imagen.

Figura 61

Bloque para el recorte de la placa vehicular

```
x, y, x_1, y_1 = box.xyxy[0].tolist()
w = x_1 - x
h = y_1 - y
crop_img = frame[y:y+h, x:x + w]
```

La imagen recortada se utiliza para la extracción de los caracteres y obtener la placa vehicular. Para analizar la imagen se implementa una etapa de procesamiento con el objetivo de preparar las imágenes para el sistema EasyOCR. La librería que se utiliza es OpenCV, cuyas funciones son implementadas sobre la imagen recortada. Una función muy útil es *cv2.cvtColor*, que se utiliza para realizar la conversión del espacio de color de una imagen. Esta función recibe como parámetro la variable código_de_color, este se define utilizando constantes proporcionadas por OpenCV. En este caso se utilizó *cv2.COLOR_BGR2GRAY* para convertir a escala de grises, la imagen que se espera obtener sería similar a lo presentado en la Figura 62.

Figura 62

Conversión a escala de grises de una placa vehicular

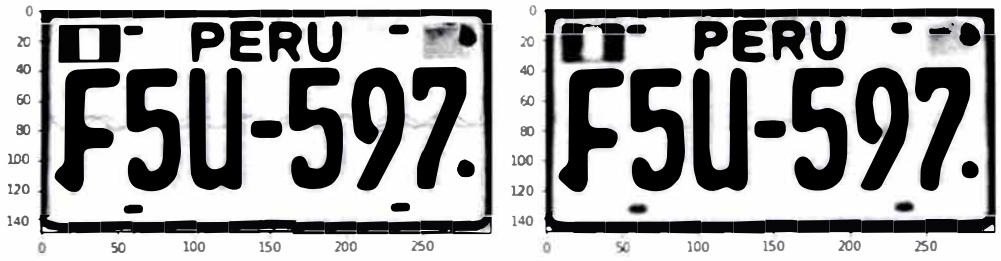
A continuación, se utiliza la función *cv2.resize*, esta permite obtener una versión de la imagen con un aumento en sus dimensiones sin agregar deformaciones. La necesidad de implementar esta función radica en que el sistema requiere mantener unas dimensiones definidas para el sistema de OCR. Esta función recibe como parámetros las dimensiones para la nueva imagen y el método que se utiliza para el redimensionamiento. Los métodos que define OpenCV son importantes, pues definen la forma en la que se calculan los nuevos valores para pixeles creados para completar las dimensiones. Dado que mayormente las placas vehiculares poseen un tamaño menor al requerido, se utiliza el

método de interpolación bicúbica. En la función se agrega el parámetro *inter_cubic*, el cual utiliza matrices de 4 pixeles para realizar la interpolación.

Luego de aplicar procesamientos en el dominio de las intensidades de los pixeles, se puede procesar en el dominio de la frecuencia, en el cual se descompone una imagen en su contenido frecuencial desde las frecuencias más bajas a las más altas. En el contexto del análisis frecuencial, un filtro es una operación que amplifica o elimina determinadas bandas de frecuencia. Estas bandas de frecuencia no deseadas son denominadas ruido y resultan un obstáculo para el sistema de OCR. Opencv brinda la función *cv2.GaussianBlur*, para implementar un filtro que utiliza un *kernel* gaussiano. Los parámetros que ingresan en esta función especifican las medidas 'x' e 'y' del *kernel*, que deben ser positivas e impares. Otro valor que se ingresa es la desviación estándar, definido como σ_x por estar en la dirección X y σ_y por estar en la dirección Y. Estos valores deben ser calculados usando el *kernel*, en el caso de que se les asigne valor nulo. En la Figura 63 se observa cómo afecta a una imagen este filtro, por ejemplo, las lineas de fondo de la placa se difuminan y en ciertas zonas desaparecen.

Figura 63

Aplicación del filtro Gaussiano para una placa vehicular



Otro método para la reducción del ruido es *cv.bilateralFilter*, este posee la particularidad de mantener los bordes nítidos y cumple con eliminar frecuencias no deseadas. Este utiliza los mismos principios que el filtro Gaussiano, sin embargo, posee la capacidad de reconocer pixeles con valores similares que pertenezcan a un borde de manera que tenga un método de cálculo particular para no deformarlo. Con la imagen de la placa preparada se utiliza la función *readtext* de la instancia *read*, creada previamente.

Esta recibe como parámetros la imagen en la que se identifican los caracteres, así como la lista de caracteres que se pueden ser detectados. Como se observa en la Figura 64, será la variable *res* la que almacena el texto detectado por el sistema EasyOCR.

Figura 64

Bloque de identificación de caracteres

```
res = read.readtext(gran, allowlist='0123456789ABCDEFGHIJKLMNOPQRSTUWXYZV')
```

Sin embargo, si bien el sistema utilizado es robusto para ser implementado en tiempo real, se planteó una función que verifique los caracteres que se detectaron. La forma en la que se verifica está basada en la estructura típica de las placas vehiculares del Perú. Estas poseen seis caracteres donde el primero y el tercero son letras, mientras que los tres últimos son números y el segundo carácter podrá ser cualquiera. En la función implementada se aprovecha la idea de que el texto obtenido del OCR es una cadena, por lo que se puede trabajar con cada elemento de dicha cadena. En la Figura 65 se observa un primer condicional para verificar que la placa posea mínimo seis caracteres. Luego, el segundo condicional verifica que cada carácter corresponda a una letra o número según el formato presentado.

Figura 65

Bloque de verificación de la placa vehicular

```
if len(text) < 6:
    return False

elif (text[0] in string.ascii_uppercase or text[0] in dict_int_to_char.keys()) and \
   (text[2] in string.ascii_uppercase or text[2] in dict_int_to_char.keys()) and \
   (text[3] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or text[3] in dict_char_to_int.keys()) and \
   (text[4] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or text[4] in dict_char_to_int.keys()) and \
   (text[5] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or text[5] in dict_char_to_int.keys()):
    return True
```

Con el objetivo de hacer más robusto el sistema se agrega un diccionario con letras y número que son más probables que identifique de forma errónea, de forma que automáticamente se realice la corrección. Esta se creó basándose en los caracteres que posiblemente serían confundidos por el modelo.

Figura 66

Diccionarios de caracteres para corregir

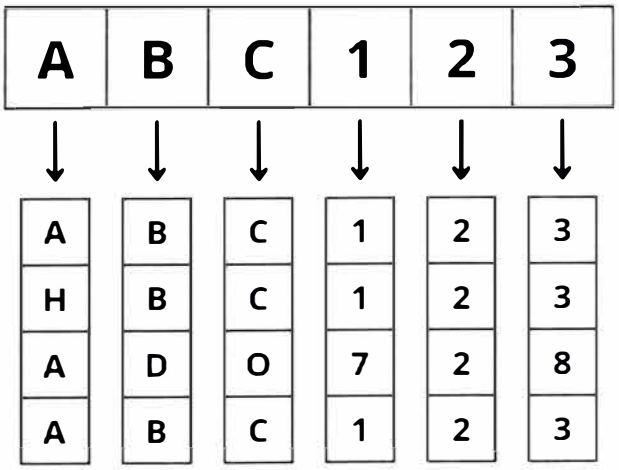
```
dict_char_to_int = {'O': '0',
                    'I': '1',
                    'J': '3',
                    'A': '4',
                    'G': '6',
                    'S': '5'}

dict_int_to_char = {'0': 'O',
                    '1': 'I',
                    '3': 'B',
                    '4': 'A',
                    '6': 'G',
                    '5': 'S'}
```

Dado que el sistema se desarrolla para su implementación en tiempo real, se considera que la identificación de caracteres se realiza en cada *frame* desde que se detecta la placa. Por esta razón, se obtienen múltiples resultados para una misma placa y dependiendo de diversos factores (luz, ruido, daños en la placa) se producen errores. Para solucionar esta problemática se implementa un bloque de código que almacene cada carácter en una lista según su orden, como se observa en la Figura 67. De esta forma, para el primer carácter se obtiene una lista con sus múltiples identificaciones y así para los demás. El carácter que se considera como correcto es el que más se repite en la lista. En la Figura 67 se muestra cómo puede existir algún *frame* donde la placa detectada no corresponde a la real, sin embargo, estos errores no afectan a la predicción final al verificar el carácter que más se repite en cada lista.

Figura 67

Extracción de caracteres en listas



La detección del color fue implementada con el objetivo de brindar una característica más para realizar la identificación vehicular, por lo que se planteó el uso de

un algoritmo que analice el valor de los pixeles que conforman parte del vehículo. Los pixeles contienen la información del color en diferentes formatos, al usar la librería OpenCV la información es codificada en el espacio BGR. Dado que los colores en este espacio resultan de la combinación de valores de tres canales distintos, puede ser poco intuitivo y complejo realizar la identificación de los colores. En cambio, el espacio HSV presenta el valor del hue, para dividir de forma clara los colores principales por rangos. Luego, los valores de saturación y valor permiten definir colores derivados. Con esta idea se procedió a implementar una función de OpenCV que permite transformar el espacio de colores a HSV. Como se observa en la Figura 68, se transforma cada *frame* en una variable sobre la que se realiza el procesamiento.

Figura 68

Bloque de código para la conversión del espacio de color y selección del píxel

```
hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
cx = punto_seleccionado[0]
cy = punto_seleccionado[1]
pixel = hsv_frame[cy, cx]
```

Se define como área de foco la puerta de los vehículos, ya que, generalmente se mantiene el color principal de la pintura en esta zona. Mediante las coordenadas "x" e "y" de los pixeles se puede elegir cual procesar, este pertenece al *frame* transformado como se presenta en la Figura 68. La variable píxel contiene los valores de HSV en un vector, lo que permite procesarlos de forma independiente. Se hace uso de rangos de valores que identifican algunos colores, los cuales se almacenan en un diccionario que incluye el nombre del color y sus respectivos valores máximos y mínimos. En la Figura 69 se muestra ejemplos de algunos colores y sus rangos, almacenados en el diccionario "colores". En este caso se presenta que el color rojo posee dos rangos que lo identifican; uno se encuentra de 0 a 15, el otro va de 160 a 180. En ambos casos se definen mínimos de 100 para la saturación y la iluminación, mientras que el máximo será de 255. Esto se observa

en la Figura 10, donde el rojo se encuentra dividido y el rango menor a 100 de iluminación serían los tonos más oscuros en la base.

Figura 69

Diccionario de rangos HSV para colores

```
colores = {
    "ROJO": [(0, 100, 100), (15, 255, 255)],
    "ROJO": [(160, 100, 100), (180, 255, 255)],
    "GUINDA": [(0, 70, 30), (10, 255, 150)],

    "NARANJA": [(15, 100, 100), (25, 255, 255)],

    "AMARILLO": [(20, 100, 100), (35, 255, 255)],
```

Un rango de tonos de 15 a 25 define el color naranja, al aumentar a un rango de 20 a 35 se tendría el color amarillo. Como ejemplo de color derivado se define el guinda, este se obtiene en el rango de tono para rojo, pero se encuentra en una iluminación de 30 a 150 (oscuro) y en un rango de 70 a 255 de saturación. Con estos rangos definidos se procede a implementar la comparación de los valores del pixel que se analiza.

Figura 70

Bloque para la obtención del color del pixel

```
color = "No Definido"
for nombre_color, (rango_minimo, rango_maximo) in colores.items():
    if rango_minimo[0] <= pixel[0] <= rango_maximo[0] and \
       rango_minimo[1] <= pixel[1] <= rango_maximo[1] and \
       rango_minimo[2] <= pixel[2] <= rango_maximo[2]:
        color = nombre_color
        break
```

Como se presenta en la Figura 70, la variable "color" se mantiene como "No definido" hasta que ingrese al bucle for. Dentro del bucle se extraen los elementos del diccionario "colores" en sus respectivas variables. Las variables de rango contienen los vectores con sus valores de HSV, dentro del bucle se comparan los valores del pixel con los rangos mediane una condicional. El color detectado es aquel que tenga todos los valores dentro de los rangos correspondientes. Para visualizar directamente el color que se detecta se agregan funciones para colocar como un texto el nombre del color que se detecta en el *frame*. Esta función es encapsulada, de manera que el *frame* que analice sea

únicamente el área de interés. Dado que se requiere incluir esta detección en el sistema en tiempo real, se utiliza un recorte que este enlazado al recorte de la placa vehicular. Usando las coordenadas del cuadro limitador de la placa, se definen valores que permitan obtener un área cercana a la puerta del vehiculo para extraer un pixel.

Finalmente, los metadatos obtenidos se almacenan de manera que cada vehiculo sea identificado mediante estos. Para almacenarlos se crea un archivo de texto, en el cual se puede agregar los datos de cada vehiculo siguiendo una estructura. En esta investigación, el archivo de texto incluye la fecha y hora en la que se identifica el vehiculo, la localización de la cámara, la placa vehicular y el color. En la Figura 71 se observa que el código para almacenar los datos apertura un documento ".txt" y luego se define la variable "data". Esta variable es la que almacena los metadatos que se escriben en el archivo y al ser de fácil implementación se podrán modificar si es requerido.

Figura 71

Bloque para la escritura de los metadatos en el archivo de texto

```
with open("LOG.txt", "a") as file:
    data = [str(now.strftime("%Y-%m-%d %H:%M:%S")) + ", " + "Puerta 5" + ", " +
            + placa + ", " + color + "\n"]
    file.writelines(data)
```

El formato esperado para el archivo es el que se aprecia en la Figura 72, donde se puede encontrar de manera ordenada la *metadata* que identifica cada vehiculo.

Figura 72

Formato del archivo de texto para almacenar la metadata

```
2023-01-10 15:49:22, Puerta 5, BCJ123, Azul
2023-01-10 15:50:39, Puerta 5, A0N562, Negro
2023-01-10 15:51:11, Puerta 5, BXT642, Plata
```

3.1.9 Sistema de adquisición de imágenes

El sistema desarrollado fue planteado para funcionar en tiempo real y de manera remota. Por esta razón, se utilizó una cámara capaz de conectarse a internet y que sea de fácil implementación. La cámara DS-2CD1043G0-I de la marca Hikvision se observa en la Figura 73. Esta brinda la posibilidad de conectarse mediante un cable ethernet a la red,

posee una calidad de imagen de 4 megapíxeles y se accede mediante un enlace que se agrega en el código explicado en la anterior sección.

Figura 73

Cámara utilizada para la implementación del sistema

El principal acceso para vehículos en la Universidad Nacional de Ingeniería es la puerta N°5, en este lugar se encuentran dos barreras que permiten la toma manual de datos para los encargados de seguridad. Se decidió instalar la cámara en este lugar con vista directa al ingreso de vehículos, como se observa en la Figura 74. Además, se preparó un soporte móvil para colocar la cámara y realizar las pruebas de manera remota.

Figura 74

Posición de la cámara instalada con un soporte

Capítulo IV. Análisis y discusión de resultados

En el siguiente capítulo se presentan y analizan los resultados obtenidos para cada algoritmo propuesto para el sistema de identificación. Estos resultados ilustran la conveniencia o no de aplicarlos; para lo cual se presentan las métricas que demuestren el rendimiento de cada uno de los citados algoritmos.

4.1 Análisis y discusión de resultados del modelo propio

La arquitectura desarrollada en esta investigación permite analizar si es necesario aplicar el aprendizaje por transferencia para utilizar modelos más complejos. Esto se daría en el caso de que las métricas no brinden la confianza necesaria para cumplir con el objetivo del sistema.

4.1.1 Resultados del modelo propio

Al realizar el entrenamiento del modelo existe la opción de seleccionar el algoritmo de optimización como se observa en la Figura 50, este afecta directamente al entrenamiento en velocidad y eficiencia. Los optimizadores que se implementaron en la búsqueda del modelo más preciso fueron Adagrad, RMSProp y Adam.

1. **Modelo optimizado con el algoritmo Adagrad.** Al implementar el optimizador se obtuvieron los siguientes resultados.

Figura 75

Valores de IoU para imágenes de prueba usando Adagrad

Para las imágenes de prueba en la Figura 75, se observa que los valores de IoU son muy bajos. Al encontrar valores nulos o cercanos a cero, se entiende que el modelo no es capaz de localizar correctamente la placa pues no existe intersección de la predicción con el cuadro real. Para el valor de 0.3 se aprecia que existe cierta intersección sin llegar a ser muy precisa la localización. Si se define un valor umbral de 0.4 de IoU para decir que la predicción tiene cierto grado de precisión y se analizan todas las imágenes de prueba se obtiene el porcentaje de predicciones que cumplen este umbral. De las 560 imágenes de prueba el 1.79% poseen un IoU superior a 0.4, siendo este un porcentaje muy pequeño el modelo con el optimizador Adagrad no resulta ser preciso.

2. **Modelo optimizado con el algoritmo RMSProp.** Con el objetivo de obtener mejoras en la precisión de los cuadros limitadores se implementó el optimizador RMSProp y se obtuvo lo siguiente.

Figura 76

Valores de IoU para imágenes de prueba usando RMSProp

Para el caso de este optimizador se observan valores más altos de IoU para las imágenes de prueba en la Figura 76. Si bien se puede apreciar una predicción con un valor cercano a 0, también se encuentran valores superiores al umbral 0.4 antes definido. Esto indicaría que al utilizar el algoritmo RMSProp el modelo fue capaz de mejorar su precisión al localizar las placas. Al realizar el mismo análisis que para el algoritmo anterior, el porcentaje de predicciones superiores a 0.4 será del 28.93% de las imágenes de prueba.

3. **Modelo optimizado con el algoritmo Adam.** Si bien se obtuvo una mejora considerable, el porcentaje obtenido continúa siendo bajo y generaría gran cantidad de errores. Para buscar una mejora en los porcentajes, se implementó el optimizador Adam y se obtuvieron los siguientes resultados.

Figura 77

Valores de IoU para imágenes de prueba usando Adam

En la Figura 77 se presentan valores de IoU superiores a los obtenidos con los anteriores optimizadores, lo cual daría indicios de una gran mejora. La presencia de valores superiores a 0.6 indican mayor intersección entre la predicción y lo real, sin embargo, aún se aprecian valores de 0.1 o 0.2. Para verificar si el modelo realmente incrementó en su precisión usó el umbral de 0.4 y se obtuvo que un 69.11% lo superaban. De esta forma se aprecia que al usar el optimizador Adam el porcentaje aumenta en un 40% al compararlo con RMSProp y brinda una mayor confianza en sus predicciones.

Pese a obtener un porcentaje superior al 65%, se debe considerar que el umbral es de un 0.4, y si este aumenta para buscar mayor precisión, los porcentajes no resultarían tan favorables. Debido a estos resultados, se decide utilizar modelos con arquitecturas más robustas, con los que se pueda realizar el proceso de aprendizaje por transferencia con el conjunto de imágenes creado.

4.2 Análisis y discusión de resultados de los modelos YOLOv8

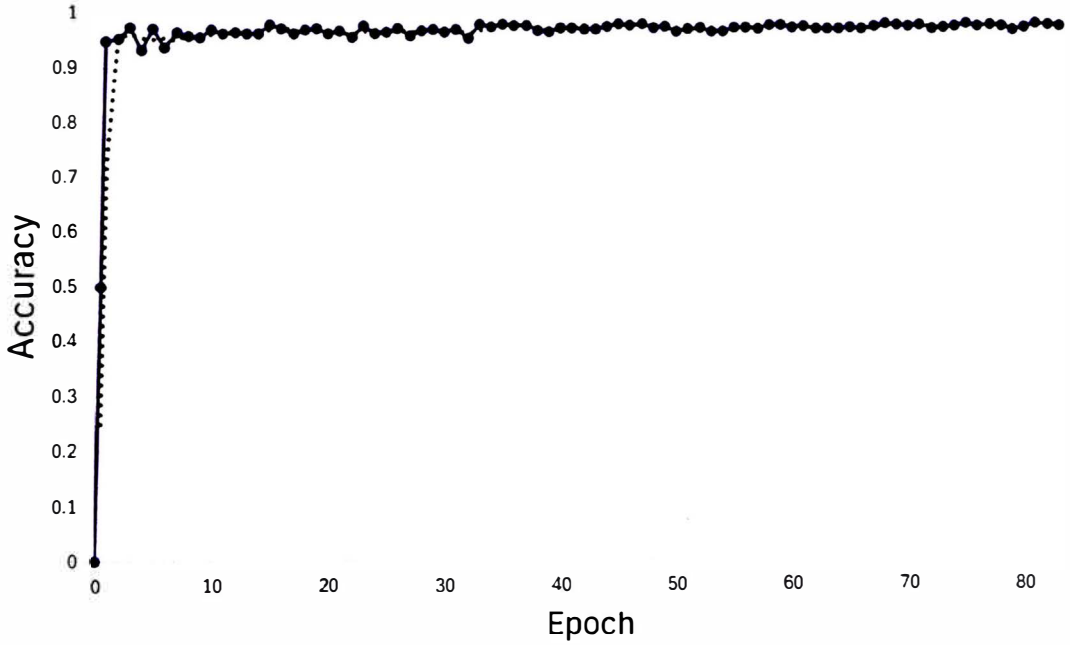
Se realizó el entrenamiento del modelo de *Ultralytics* YOLOv8 en sus versiones "m", "s" y "n"; la diferencia entre estas es la cantidad de parámetros y capas que reducen el peso del modelo. La inclusión de bloques especializados para la extracción de características en la arquitectura de esta versión de Yolo permite realizar detecciones con alto grado de efectividad. Además, la compatibilidad con el uso de una tarjeta gráfica aumenta considerablemente la velocidad de las detecciones.

4.2.1 Resultados de YOLOv8-m

El modelo en su versión mediana (m), fue entrenado con 120 ciclos de entrenamiento (*epoch*), tamaño de imagen de 800x800 pixeles. La tasa de aprendizaje definida de forma inicial es de 0.01, además cuenta con parada temprana.

Figura 78

Valores de validación para la precisión del modelo YOLOv8-m, donde "accuracy" corresponde a la precisión.



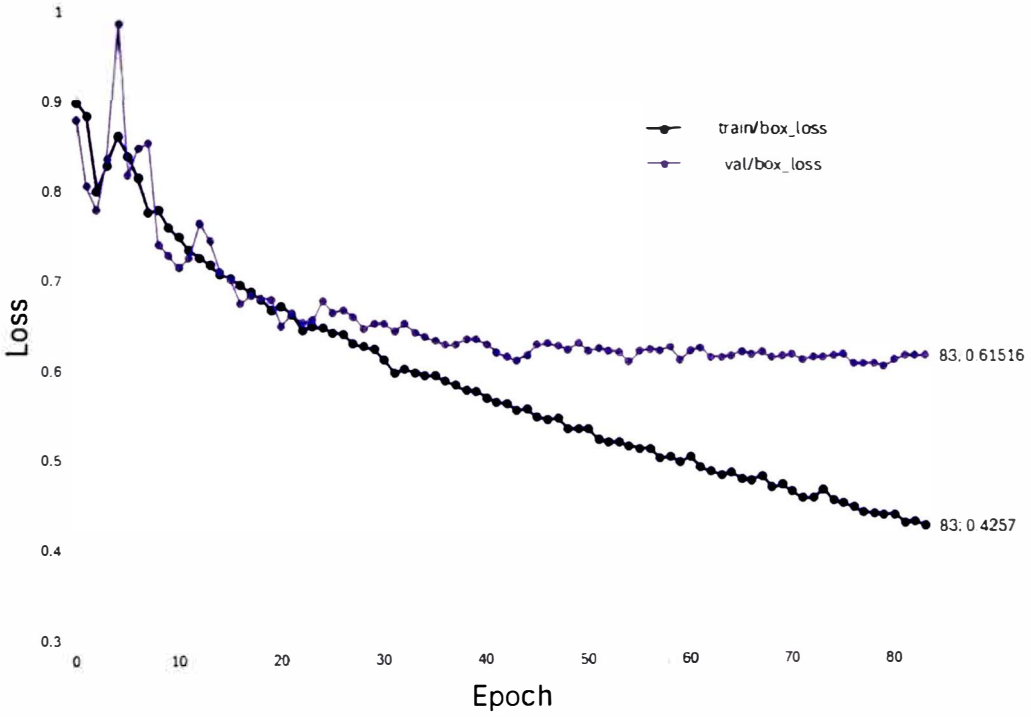
Durante cada ciclo de entrenamiento el modelo realiza una validación de los pesos con un grupo de imágenes, con los que se obtiene un valor de precisión en cada uno, como se observa en la Figura 78. El modelo presenta valores altos de precisión desde los primeros ciclos, luego presenta variaciones que tienden a hacerse menos notorias,

generando curvas suaves. El entrenamiento se detiene antes de completar los ciclos definidos, ya que obtiene el valor de 0.97657 y al detectar que el aprendizaje no mejora se detiene. Esta detención temprana evita un uso excesivo de los recursos computacionales para un entrenamiento innecesario, que podría incluso incluir ruido en los datos. El modelo, entonces, aprende de forma correcta del conjunto de imágenes de placas peruanas y posee la capacidad de realizar la tarea de detección con una precisión del 97.7%.

Dado que las arquitecturas para modelos de detección de objetos poseen dos salidas, se puede visualizar la reducción de la pérdida para la localización del recuadro delimitador (box) y para la identificación de la clase (cls).

Figura 79

Reducción de la función de pérdida (loss) para la localización del cuadro delimitador durante el entrenamiento y validación del modelo YOLOv8-m

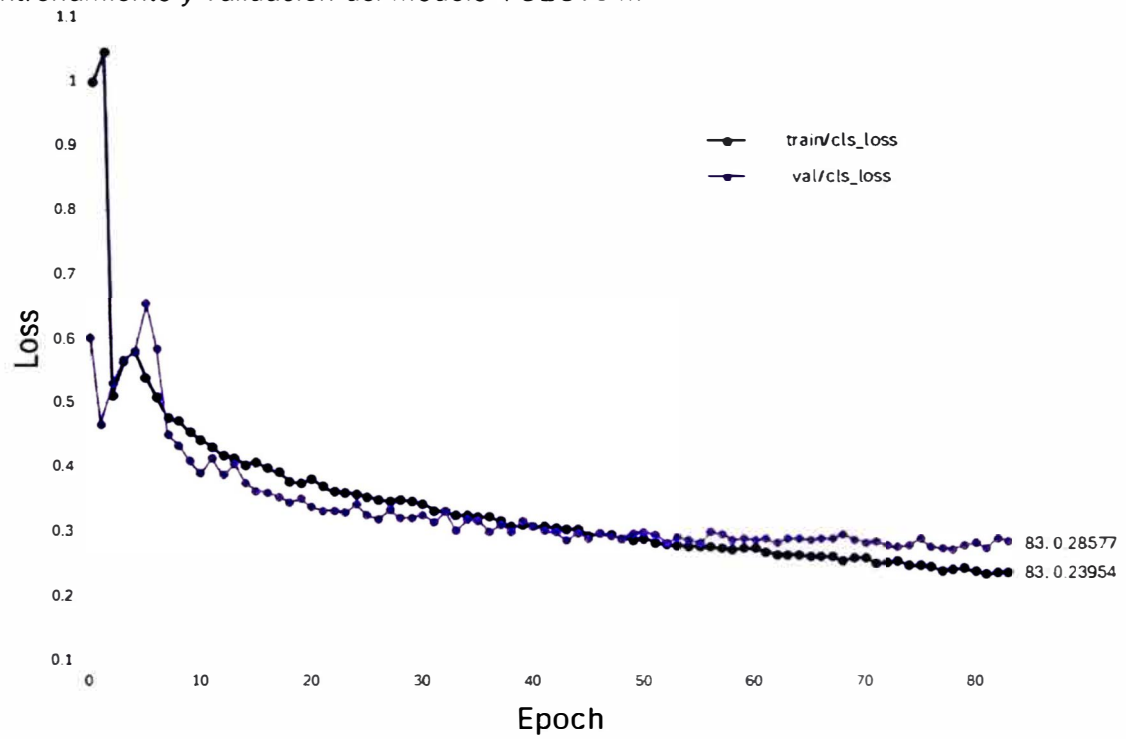


En la Figura 79 se presenta el entrenamiento, donde la tasa de pérdida experimenta ciertos picos que dan la apariencia de un modelo con sobreajuste. Sin embargo, esto ocurre únicamente en los primeros ciclos, ya que luego el modelo reduce la perdida de manera casi constante hasta tener un valor de 0.4257. Por otro lado, durante la validación los picos aparecen de forma pronunciada; hasta que la pérdida comienza a tener pequeñas

disminuciones para llegar al valor final de 0.615. Si bien, la reducción de la pérdida en la validación presenta algunos pequeños incrementos, los valores de pérdida no se alejan demasiado. En los últimos ciclos se puede apreciar un ligero aumento de la pérdida, sin embargo, la parada temprana evita que exista un mayor incremento en la pérdida.

Figura 80

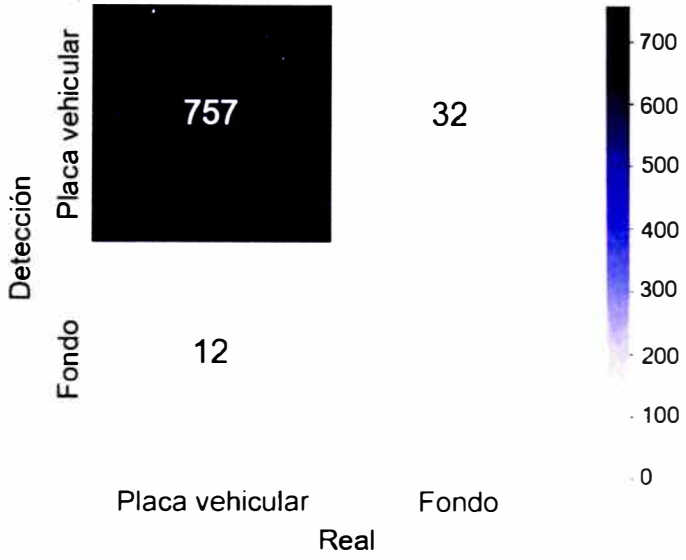
Reducción de la función de pérdida (loss) para la identificación de la clase durante el entrenamiento y validación del modelo YOLOv8-m



Para la identificación del objeto localizado, la función de pérdida experimenta una reducción rápida de un valor cercano a 1 hacia valores cercanos a 0.5 en menos de 10 épocas, como se observa en la Figura 80. Luego, este valor se reduce de forma suave y constante hasta el valor final de 0.239. Para el caso de la validación se presentan picos que tienden a hacerse pequeños, hasta reducirse a mínimas variaciones en los últimos ciclos. El valor final que se obtiene para la validación es de 0.28577, tanto este valor como el de entrenamiento presentan un correcto decremento.

Figura 81

Matriz de confusión para el modelo YOLOv8-m



En la matriz de confusión de la Figura 81 se presentan las predicciones realizadas, dado que la única etiqueta es la de la placa, se tendrá el fondo como una etiqueta extra para el análisis. En este caso el modelo detectó de forma correcta 757 placas vehiculares y no logró detectar 12. Por otro lado, se obtuvieron 32 detecciones de placas vehiculares que realmente eran partes del fondo. Con estos valores se calcula la precisión, obteniendo así un 95.9% y un valor de 98.4% para el *recall*.

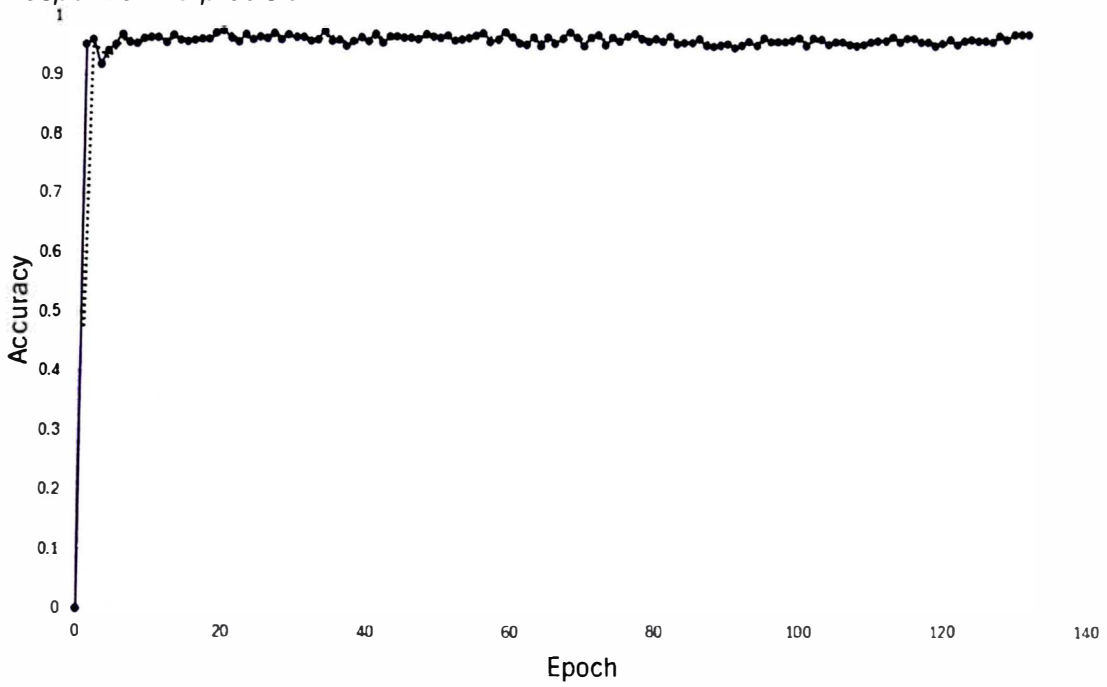
4.2.2 Resultados de YOLOv8-s

El modelo en su versión pequeña (s), fue entrenado con 200 ciclos de entrenamiento, tamaño de imagen de 800x800 pixeles. La tasa de aprendizaje elegida de forma inicial es de 0.01, además cuenta con parada temprana.

La cantidad de ciclos de entrenamiento necesarios para obtener métricas correctas para esta versión de YOLOv8 resulta ser mucho mayor para su versión "m", como se observa en la Figura 82. Se puede apreciar la misma tendencia del modelo a rápidamente alcanzar altos valores de precisión y, en un inicio, existen variaciones pronunciadas que disminuyen posteriormente sin desaparecer. Por lo que, a diferencia de la versión mediana, esta aparenta tener mayor complicación para ajustar los pesos y lograr mejorar la detección.

Figura 82

Valores de validación para la precisión del modelo YOLOv8-s, donde "accuracy" corresponde a la precisión

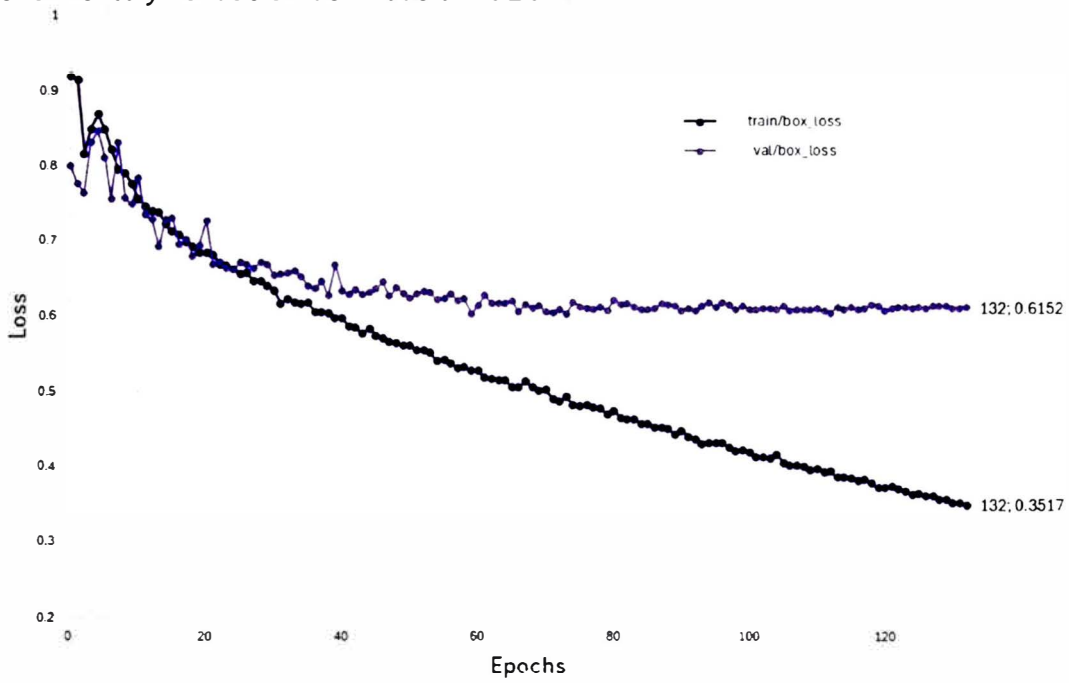


En los últimos ciclos, los valores de precisión aumentan lentamente hasta obtener un valor de 0.97448, donde el proceso se detiene al detectar que no hay mejoras en el aprendizaje. El valor de la precisión resulta ser muy cercano al obtenido en el entrenamiento de la versión mediana; demostrando que a pesar de reducir en gran medida la cantidad de parámetros, no se pierde precisión en la detección. Con un porcentaje de 97.5% de precisión, el modelo reduce el gasto computacional al realizar detecciones por su menor peso; sin embargo, toma mayor cantidad de ciclos de entrenamiento para obtener buenos resultados.

A continuación, se analiza la reducción de la pérdida para la localización del recuadro delimitador y para la identificación de la clase.

Figura 83

Reducción de la función de pérdida para la localización del cuadro delimitador durante el entrenamiento y validación del modelo YOLOv8-s

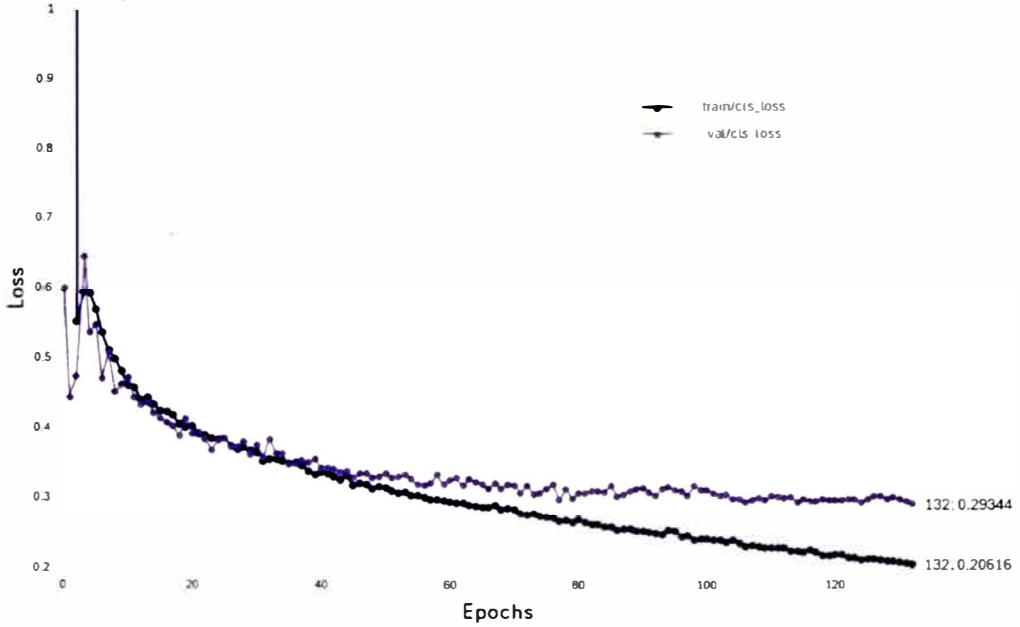


En la Figura 83 se observa que la tasa de pérdida experimenta picos menos pronunciados que en la versión "m", tanto en el entrenamiento como en la validación. Durante la validación se presenta mayor cantidad de estos picos, pero una vez que el modelo logra ajustar los pesos, estos desaparecen y la pérdida comienza a reducirse en pequeñas cantidades. En el entrenamiento se presenta una reducción suave hasta 0.3517, mientras que, en la validación termina en un valor de 0.6152. En ambos casos se produce una reducción de la pérdida por lo que el modelo aprende de forma correcta.

Para la identificación del objeto localizado, la función de pérdida mantiene el comportamiento de reducir rápidamente su valor en pocos ciclos y luego presenta una reducción más lenta. Para el caso del entrenamiento se produce una reducción suave hasta el valor de 0.20616, como se observa en la Figura 84. En la validación se genera mayor cantidad de picos, estos son menos pronunciados que los del modelo "m". Las variaciones se vuelven menos frecuentes hasta que la pérdida se reduce lentamente y toma un valor final de 0.29344.

Figura 84

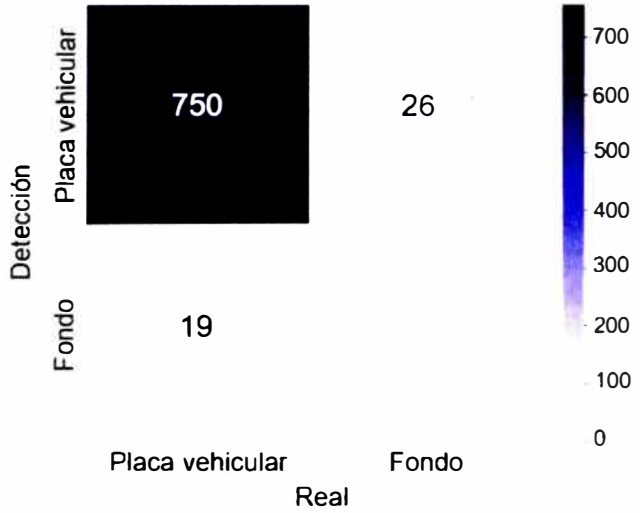
Reducción de la función de pérdida para la identificación de la clase durante el entrenamiento y validación del modelo YOLOv8-s



En la matriz de confusión de la Figura 85 se presenta que el modelo detecto de forma correcta 750 placas vehiculares y no logró detectar 19. Por otro lado, se obtuvieron 26 detecciones de placas vehiculares que realmente eran partes del fondo. Al comparar esta versión con la "m" se puede apreciar que este modelo presenta mayor cantidad de placas que no detecta, mientras que los errores de confundir el fondo con una placa se reducen. La precisión obtenida con la matriz es de 96.6% y el *recall* es de 97.5%.

Figura 85

Matriz de confusión para el modelo YOLOv8-s

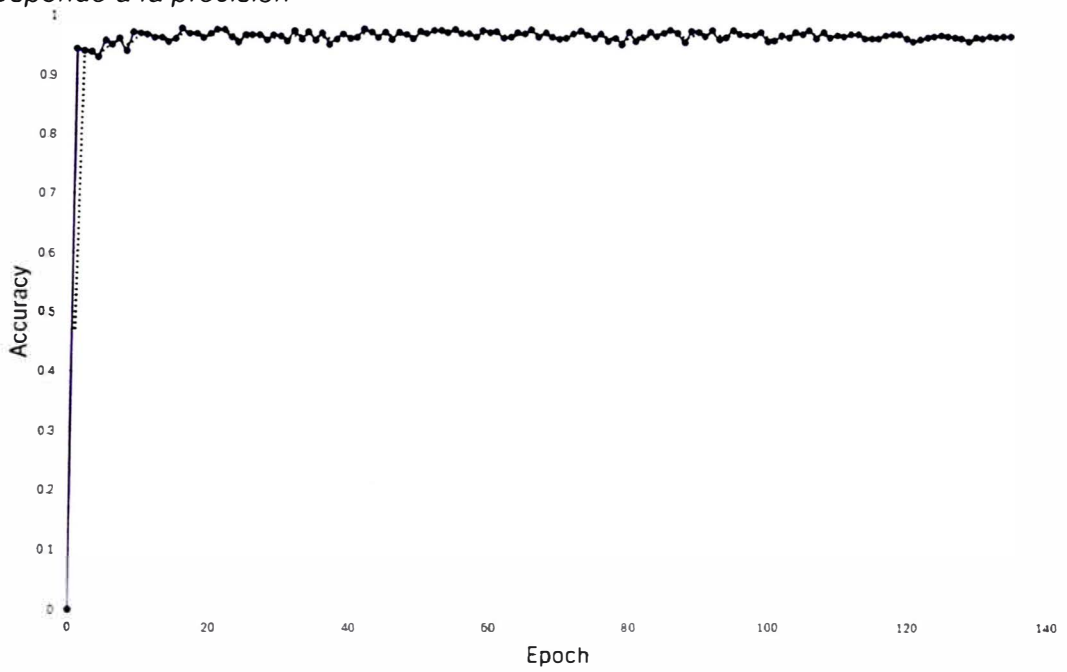


4.2.3 Resultados de YOLOv8-n

El modelo en su versión nano (n), fue entrenado con 200 ciclos de entrenamiento, tamaño de imagen de 800x800 pixeles. La tasa de aprendizaje definida de forma inicial es de 0.01, además cuenta con parada temprana.

Figura 86

Valores de validación para la precisión del modelo YOLOv8-n, donde "accuracy" corresponde a la precisión



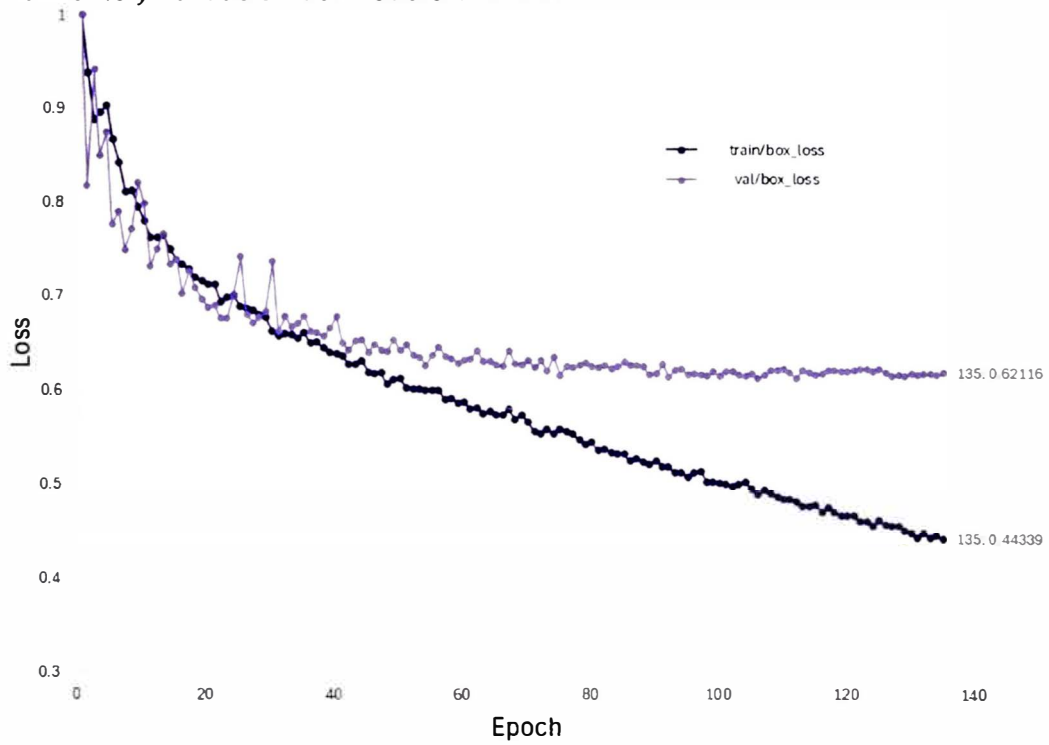
Para esta versión del modelo se requiere una cantidad similar de ciclos de entrenamiento que las realizadas por la versión "s", como se observa en la Figura 86. Por lo que se puede decir que no brinda una mejora en la cantidad de gasto computacional requerido para entrenar el modelo. De igual manera alcanza altos valores de precisión en los primeros ciclos, luego se presentan aumentos y decrementos variados sin llegar a generar grandes picos. Posee un comportamiento similar a la versión pequeña, sin embargo, en la última etapa del gráfico se observa que se vuelven más suaves los incrementos. El valor final que se obtiene es de 0.96663, en este momento el modelo se detiene al no obtener mejoras en el aprendizaje. Con un porcentaje de 96.7% de precisión, el modelo posee un porcentaje menor a los anteriores modelos por 1%. Esto no representa

una gran diferencia, mientras que la cantidad de parámetros y peso del modelo entrenado si se reducen en gran medida.

A continuación, se analiza la reducción de la pérdida para la localización del recuadro delimitador y para la identificación de la clase.

Figura 87

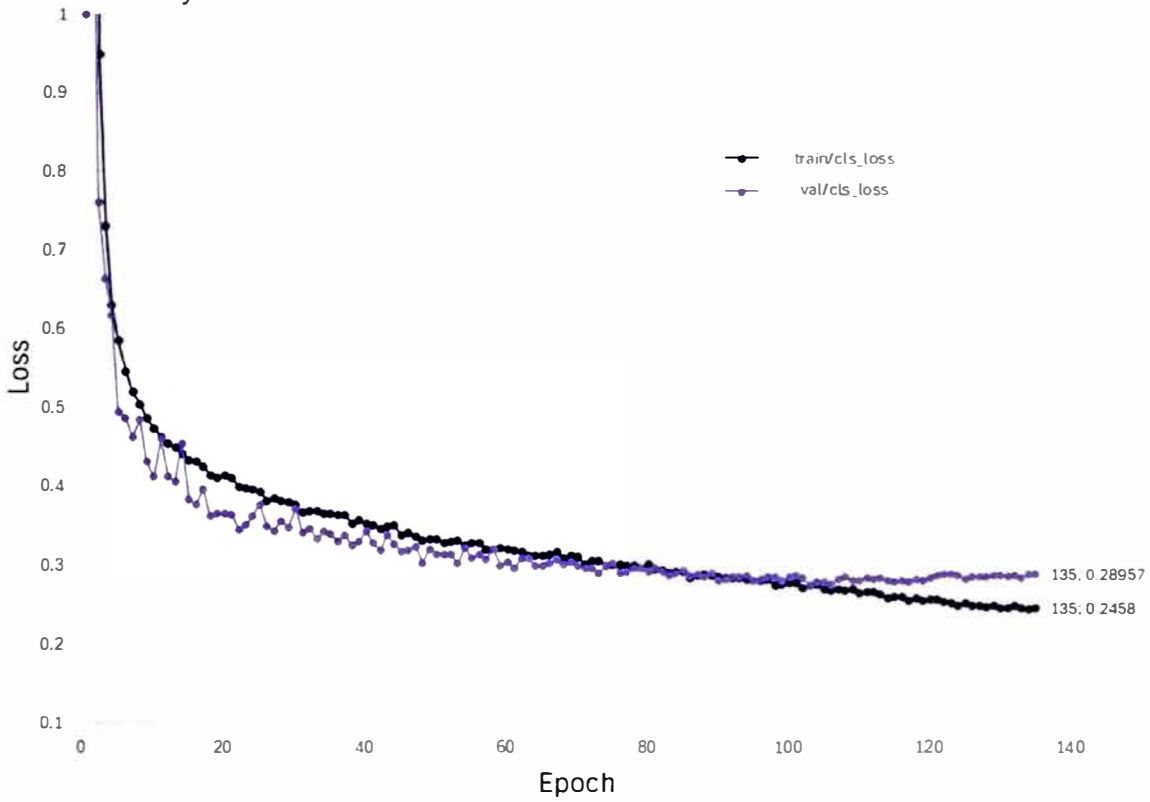
Reducción de la función de pérdida para la localización del cuadro delimitador durante el entrenamiento y validación del modelo YOLOv8-n



Para la localización del cuadro delimitador se presenta la gráfica de su función de perdida durante el entrenamiento en la Figura 87. Donde se observa que el comportamiento es similar a los anteriores casos, manteniendo una reducción prácticamente suave y constante. El valor final que se obtienen del entrenamiento es de 0.44339 y termina antes de completar los ciclos definidos para evitar la interferencia del ruido. Durante la validación se observan picos en los ciclos iniciales, sin embargo, estos siguen siendo considerables durante más ciclos que en la versión "s". En la última etapa, el modelo logra una reducción de la pérdida de forma suave hasta el valor de 0.62116. Ambos valores resultan ser similares a las versiones de mayor tamaño, pero el objetivo de reducir la pérdida se cumple.

Figura 88

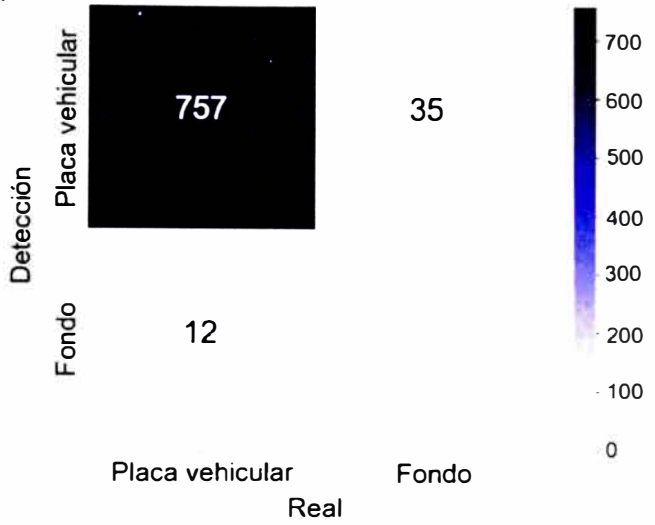
Reducción de la función de pérdida para la identificación de la clase durante el entrenamiento y validación del modelo YOLOv8-n



En el caso de la identificación de la clase se ha demostrado que el modelo reduce rápidamente la pérdida en pocos ciclos, esto se mantiene para la versión nano. El entrenamiento genera una curva suave que reduce su valor hasta 0.2458. Los picos que generalmente se aprecian al inicio de la validación se aprecian de igual forma en la Figura 88, sin embargo, estos se vuelven menos notorios en menor cantidad de ciclos. Al desaparecer las grandes variaciones, el decremento de la perdida es más lento y llega a un valor de 0.28957. Ambos valores son pequeños por lo que el modelo realiza el aprendizaje de forma correcta.

Figura 89

Matriz de confusión para el modelo YOLOv8-n



Los valores que se presentan en la matriz de confusión para esta versión del modelo resultan ser muy similares a los del modelo "m". Como se observa en la Figura 89 se detectó de forma correcta 757 placas vehiculares y se detectaron 12. Por otro lado, se obtuvieron 35 detecciones de placas vehiculares que realmente eran partes del fondo. Con estos valores se calcula la precisión, obteniendo así un 95.5% y el valor de 98.4% para el *recall*.

4.2.4 Comparación de los modelos descritos

Luego de presentados los modelos se podrá analizar las características de cada uno para definir la mejor opción para el sistema propuesto. Las características que se muestran en la Tabla 10 permiten visualizar cual es el modelo que posee el mejor valor en cada columna. En la primera se incluye el nombre de la versión del modelo, en la siguiente columna se coloca el porcentaje de precisión obtenido de la validación. En este caso el mejor porcentaje se obtuvo con la versión mediana, sin embargo, la diferencia con la versión pequeña es de 0.2% y con la versión nano es de 1%. Luego, el *recall* define qué versión es capaz de detectar correctamente las placas en un *frame*, las versiones "m" y "n" son las que poseen el mayor porcentaje. En las dos últimas columnas se tienen características relacionadas directamente con el coste computacional en cuanto al

almacenamiento y cálculos realizados por el procesador. En este campo se observa que la versión mediana posee 8 veces más parámetros que el nano y la versión pequeña tiene casi 4 veces más. Por consecuencia, el archivo que contiene los pesos del modelo de la versión nano es el que requiere menor cantidad de espacio de almacenamiento, siendo 10 y 4 veces menos que las versiones "m" y "s respectivamente.

Tabla 10

Comparación de características de las versiones de YOLOv8

Modelo	Precisión	Recall	Parámetros	Almacenamiento (MB)
YOLOv8-m	**97.7%**	**98.4%**	25902640	49 6
YOLOv8-s	97 5%	97 5%	11166560	21 5
YOLOv8-n	96 7%	**98.4%**	**3157200**	**5.99**

La versión nano posee características que lo hacen mucho más eficiente para la implementación del sistema en tiempo real, además, permite tener gran velocidad para realizar las detecciones y reduce la necesidad de tener altos requerimientos computacionales. Esta eficiencia, suele conllevar una menor capacidad para realizar detecciones. Sin embargo, en el porcentaje de *recall* no hay variación y en la precisión la diferencia es muy pequeña. Por esta razón el uso de la versión nano representa una mejor opción cuando existen limitaciones de hardware, pero el modelo mediano resulta ser la opción más precisa para las detecciones.

4.3 Análisis y discusión de resultados del sistema EasyOCR

La implementación del modelo para la identificación de caracteres permite extraer el contenido de la placa vehicular de manera directa. Además, los bloques de código que se encargan de procesar y corregir ciertos errores hacen que los resultados posean mayor confianza.

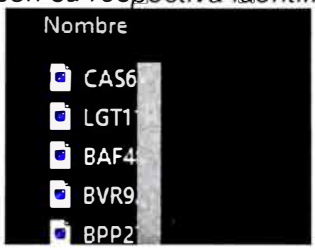
4.3.1 Resultados de la detección de placas vehiculares

Para ensayar los algoritmos se implementó un proceso de detección vehicular en la zona de ingreso a la UNI (puerta N°5). Con tal objetivo, se ejecutó el algoritmo durante 2 horas, iniciando a las 10:00 am, en este periodo de tiempo se consiguió realizar la

detección de 105 placas. La data del área de placa vehicular se almacenó en una carpeta en la que se incluye el texto detectado por el OCR como nombre del archivo.

Figura 90

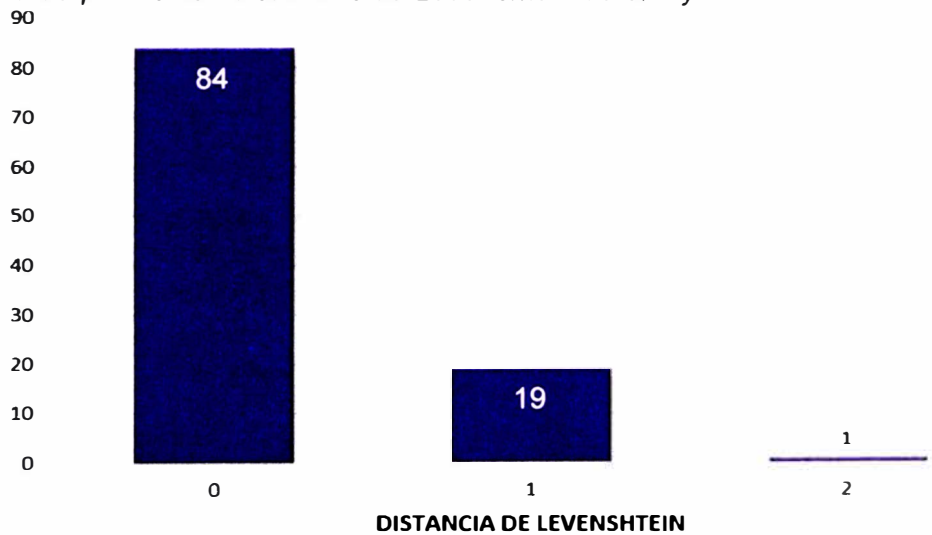
Carpeta de imágenes de placas con su respectiva identificación



A continuación, se define el valor de la distancia de Levenshtein para cada placa identificada comparándola con la imagen real. Dado el procesamiento que se realiza y la robustez del modelo, la única operación primitiva requerida para obtener la placa correcta es el reemplazo de caracteres. Al calcular la distancia para las placas en la carpeta de muestra, se obtiene la gráfica que se observa en la Figura 91. En esta se presenta que la mayoría de las placas se identifican de manera correcta, por lo que no requiere ningún reemplazo y la distancia será 0. Esto se cumple para 84 placas, mientras que para 19 de las placas se comete un error y la distancia será 1. Por último, en esta muestra únicamente se tendrá una placa con 2 errores. Finalmente, al hacer uso de la Ecuación (15) se calcula la precisión del sistema de OCR.

Figura 91

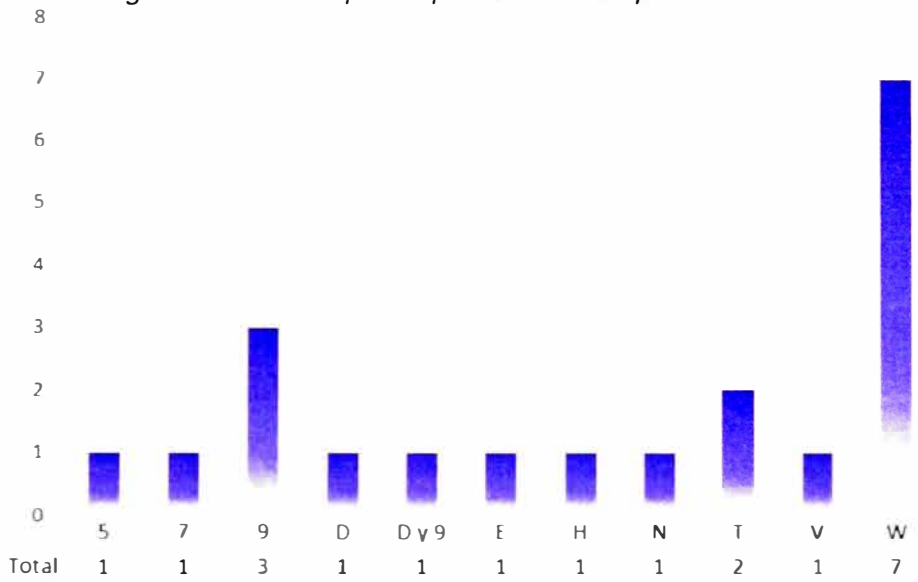
Cantidad de placas con distancias de Levenshtein de 0, 1 y 2.



Para las placas con distancia 0, el valor de precisión es de 100%; a la distancia de 1 le corresponde una precisión de 83.33% y a la distancia de 2 se le calcula una precisión de 66.67%. Con estos porcentajes se obtuvo una precisión promedio de 96.63% para la muestra adquirida. Esto representa un porcentaje alto, demostrando que los errores del sistema resultan ser mínimos.

Figura 92

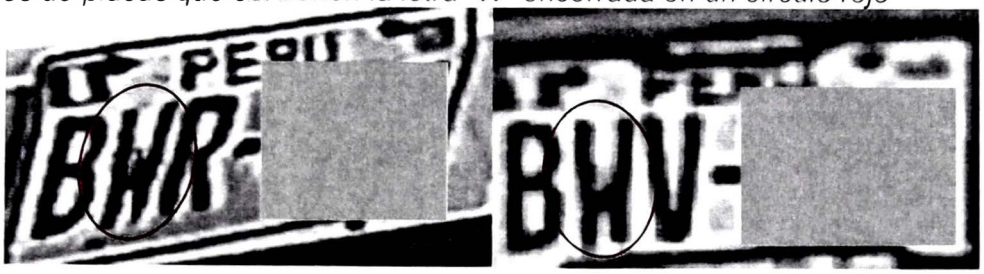
Cantidad de errores según el carácter que requiere ser reemplazado



En la Figura 92 se observa los caracteres que requieren ser reemplazados para corregir las placas erróneas, seis de los errores corresponden a caracteres numéricos y 15 se dan debido a caracteres alfabéticos. Por esta razón, se puede afirmar que los últimos tres caracteres de la placa (números) se identifican con mayor éxito que los tres primeros (2 letras y el carácter central). Además, se puede afirmar que la letra con mayor incidencia en los errores es la "W". Este error se da principalmente por la forma que posee la letra al ser capatada y procesada en el sistema, como se observa en la Figura 93, dando la posibilidad de confundirla con otras letras como la "H".

Figura 93

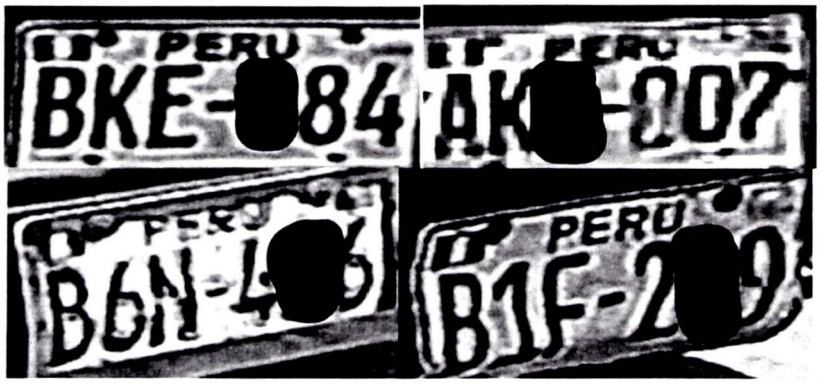
Ejemplos de placas que contienen la letra "W" encerrada en un círculo rojo



Se puede concluir que el sistema EasyOCR y el algortimo desarrollado para procesar sus predicciones, brindan la precisión requerida para la identificación de las placas vehiculares. La robustez del sistema se demuestra al realizar correctamente identificaciones en placas en buen y mal estado. Como se observa en la Figura 94, en el Perú no hay un control estricto sobre el mantenimiento de las placas vehiculares, lo cual permite la circulación de vehículos con placas dañadas o despintadas.

Figura 94

Ejemplos de placas identificadas



4.4 Resultado de la detección de color

Si bien los caracteres de la placa vehicular permiten realizar la identificación del vehículo, otra característica importante y que destaca visualmente es el color de este. Con el objetivo de analizar el funcionamiento del algoritmo desarrollado se utilizó la cámara instalada en la puerta N°5. Para comprobar si el color detectado es correcto, se usa la placa del vehículo para buscarlo en la base de datos de SUNARP y obtener su color real. En la Figura 95 se observan algunas detecciones que se realizaron de manera correcta, los colores mostrados son básicos y poseen rangos de valores amplios.

Figura 95

Ejemplos de colores detectados durante la prueba del algoritmo

La característica de los colores básicos en sus rangos, permite que sean detectados pese a tener cambios en la iluminación. Sin embargo, dada la variedad de colores de vehículos registrados, se deben definir rangos de valores para colores más específicos y derivados. Un ejemplo es el color "plata". la identificación de este color se presenta en la Figura 96. Se observa que para distintos momentos del día se logró la identificación de este color, sin embargo, cabe destacar que se realizaron ajustes en su rango.

Figura 96

Identificación del color "plata" en vehículos

Los ajustes consisten en variar valores que generan confusión con otro tono de color o que no están incluidos en los rangos propuestos. esto es necesario para mejorar los resultados ante efectos de iluminación. En la Figura 97 se observa que el algoritmo es susceptible a la presencia de sombras y reflejos en el área de extracción del pixel.

Figura 97

Variación en el color identificado debido a los reflejos y sombras. a) color errado b) color correcto

a) b)

Para el caso del auto de color "plata", si se toma un pixel que se encuentra en un área con mayor iluminación, los valores HSV se encuentran en el rango de "gris metálico". Esto se produce al tener que identificar colores muy similares, cuyos rangos varian únicamente en la iluminación. Por otro lado, en el caso del vehiculo amarillo, al elegir un pixel cubierto por sombras se detecta un color más oscuro que sale del rango de amarillos y entra en el color naranja. Por tal motivo, se puede afirmar que producto de una iluminación irregular, el algoritmo puede confundir únicamente colores que se encuentran cercanos en el espacio HSV. Al comparar con la percepción humana, el color "plata" y el "gris metálico" no son tan distintos como seria un "rojo" con un "verde".

Por último, se puede observar que resulta complicado obtener de manera precisa los colores que aparecen en los registros de SUNARP. Un ejemplo de esto se aprecia en la Figura 98, donde el algoritmo identifica un derivado oscuro del color azul y al lado se aprecia que el color con el que fue registrado es denominado "azul profundo".

Figura 98

a) Identificación del color "azul oscuro", b) Datos registrados para el vehículo F4Y---

a)

DATOS DEL VEHICULO:

N° PLACA:	F4Y[redacted]
N° SERIE:	[redacted]
N° VIN:	[redacted]
N° MOTOR:	[redacted]
COLOR:	AZUL PROFUNDO
MARCA:	KIA
MODELO:	RIO
PLACA VIGENTE:	F4Y[redacted]
PLACA ANTERIOR:	[redacted]
ESTADO:	EN CIRCULACION
ANOTACIONES:	NINGUNA
SEDE:	LIMA

b)

Luego de realizadas las pruebas se puede afirmar que el algoritmo cumple con el propósito de brindar la información sobre el color principal de un vehículo, además, los errores que se producen son producto de la iluminación en la imagen lo cual se corregiría al utilizar procesos robustos para la identificación. Si bien, se pueden obtener nombres no precisos del color o tonos distintos al real, no se presentan confusiones significativas al compararlo con lo que podría percibir el ojo humano.

4.5 Contrastación de la Hipótesis

4.5.1 Hipótesis General

Para esta investigación se consideró la siguiente hipótesis general:

Con la aplicación de algoritmos de aprendizaje profundo con análisis de metadata se realizará la identificación vehicular.

Para contrastar esta hipótesis se considera la hipótesis alternativa y la hipótesis nula, como a continuación se presenta.

- **H1:** Con la aplicación de algoritmos de aprendizaje profundo con análisis de *metadata* se realizará la identificación vehicular.

- **Ho:** Con la aplicación de algoritmos de aprendizaje profundo con análisis de *metadata* no se realizará la identificación vehicular.

Vistos los resultados del entrenamiento e implementación de modelos y algoritmos basados en aprendizaje profundo para la localización de placas vehiculares, utilizando técnicas como el aprendizaje por transferencia, se lograron alcanzar métricas que superan el 96.5% y el 98% en precisión y recall, respectivamente (Tabla 10). Además, al hacer uso de algoritmos basados en aprendizaje profundo para extracción de *metadata*, como son los caracteres de la placa, se obtuvo una precisión media del 96.6%. Este estudio también abordó la detección del color del vehículo como parte de la *metadata* asociada a la placa, lo que posibilitó la identificación precisa de los vehículos. En virtud de los resultados, se rechaza la hipótesis nula y se acepta la hipótesis alterna.

Figura 99

Resultado de los algoritmos de Deep learning para identificación vehicular

4.5.2 Hipótesis Específicas

Las hipótesis específicas que comprende esta investigación serán contrastadas planteando las hipótesis alternativa y nula respectivamente, como a continuación se indica.

1. *La implementación de arquitecturas de aprendizaje profundo basadas en redes neuronales convolucionales es eficaz en la identificación vehicular.*

- **H1:** La implementación de arquitecturas de aprendizaje profundo basadas en redes neuronales convolucionales es eficaz en la identificación vehicular.

- **Ho:** La implementación de arquitecturas de aprendizaje profundo basadas en redes neuronales convolucionales no es eficaz en la identificación vehicular.

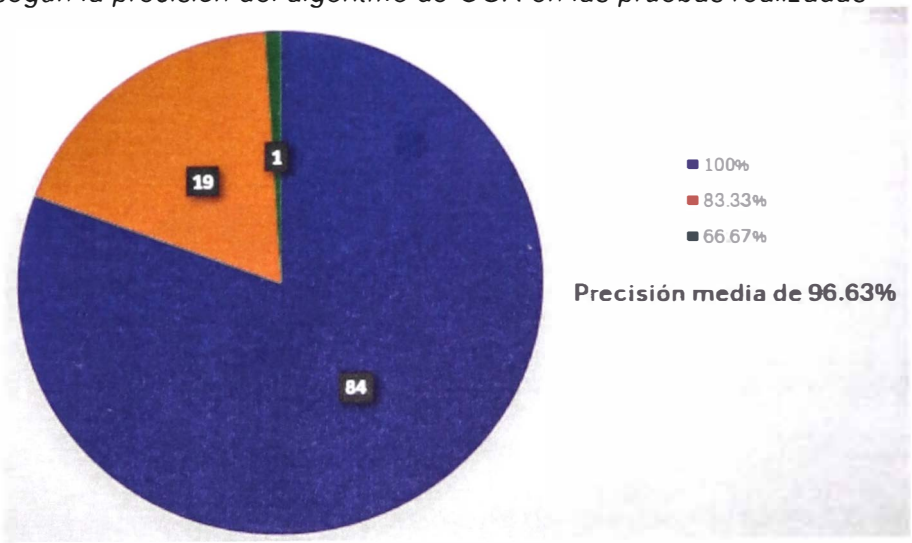
Visto los resultados mostrados en la Figura 100, las arquitecturas aprendizaje profundo utilizadas para la localización de la placa vehicular alcanzan valores de precisión por encima del 95%, siendo el modelo YOLOv8-m el que posee mejores métricas. Luego, al observar la Figura 101, la arquitectura de aprendizaje profundo implementada para la extracción de caracteres obtiene una precisión media de 96.6%. En consecuencia, vistos estos resultados, se descarta la hipótesis nula y se acepta la hipótesis alterna, más aún cuando estas arquitecturas brindan altos valores de confianza en los datos extraídos para la identificación vehicular.

Figura 100

Gráfica de la precisión para las versiones del modelo YOLOv8

Figura 101

Distribución según la precisión del algoritmo de OCR en las pruebas realizadas



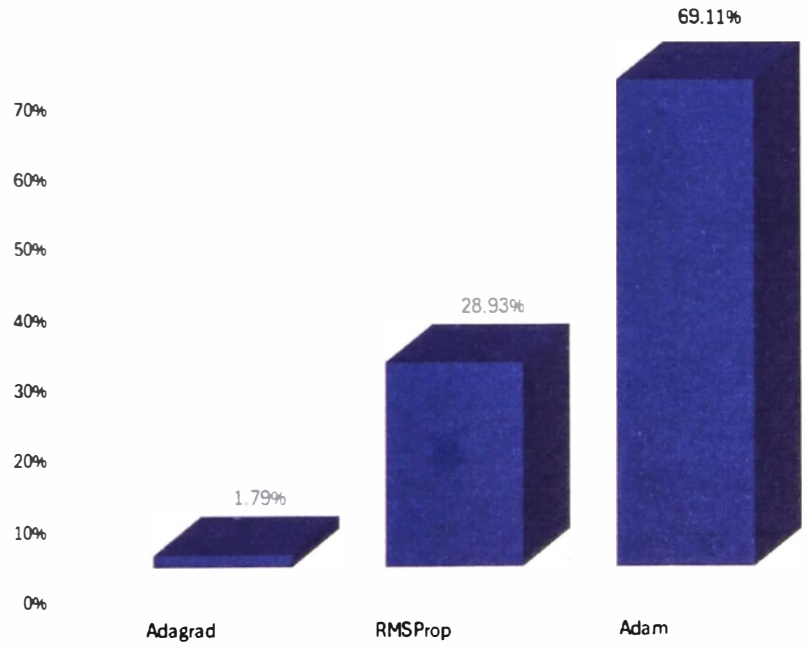
2. *La implementación de algoritmos de optimización aplicados a redes neuronales convolucionales es relevante para la identificación vehicular.*

- **H1:** La implementación de algoritmos de optimización aplicados a redes neuronales convolucionales es relevante para la identificación vehicular.
- **Ho:** La implementación de algoritmos de optimización aplicados a redes neuronales convolucionales no es relevante para la identificación vehicular.

La implementación de diferentes algoritmos para la optimización del modelo demostró generar considerables mejoras en la precisión con la que se realiza la localización de la placa vehicular. En la Figura 102 se observa que el optimizador Adam aumenta en un 40% y 68% en comparación a los otros algoritmos. Por tales motivos, la hipótesis alterna es aceptada y se descarta la hipótesis nula.

Figura 102

Porcentaje de predicciones que superan el umbral de 0.4 en IoU



3. *La aplicación de un correcto algoritmo de reconocimiento de caracteres es significativa para la identificación vehicular.*

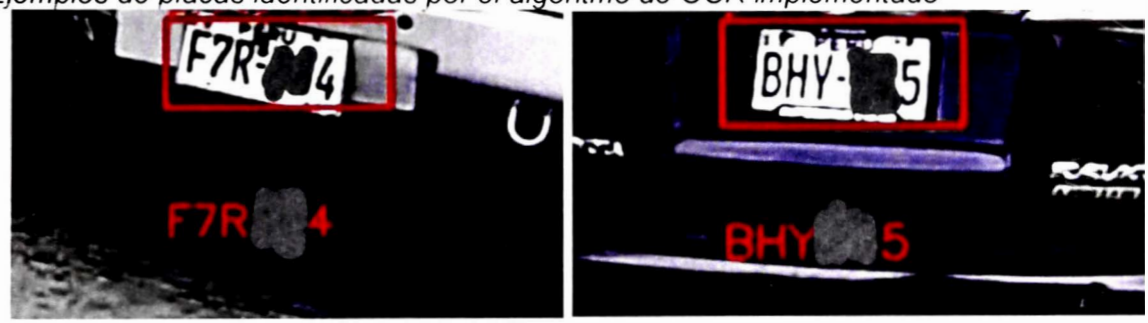
- **H1:** La aplicación de un correcto algoritmo de reconocimiento de caracteres es significativa para la identificación vehicular.

- **Ho:** La aplicación de un correcto algoritmo de reconocimiento de caracteres no es significativa para la identificación vehicular.

Para realizar la tarea de extracción de caracteres se recomienda utilizar algoritmos especializados que brinden información de la localización de caracteres en la imagen y sepan identificarlos. Es así que el uso de CRAFT para la ubicación de áreas que contienen textos, su integración con capas de procesamiento de imágenes y capas para la identificación de caracteres con modelos como Resnet, hacen de EasyOCR un algoritmo de OCR robusto para obtener la placa que identifica un vehiculo. Por esta razón, se rechaza la hipótesis nula y se acepta la hipótesis alterna.

Figura 103

Ejemplos de placas identificadas por el algoritmo de OCR implementado



4. *La evaluación de la metadata y algoritmos resulta eficaz para la identificación vehicular.*

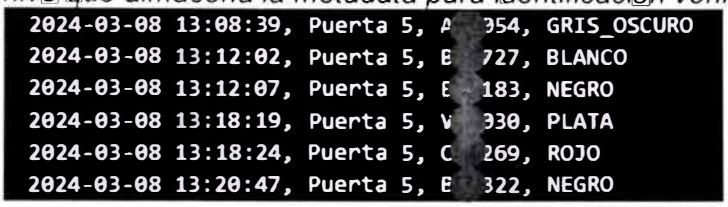
- **H1:** La evaluación de la *metadata* y algoritmos resulta eficaz para la identificación vehicular.
- **Ho:** La evaluación de la *metadata* y algoritmos no resulta eficaz para la identificación vehicular.

Al utilizar modelos que pueden cometer ciertos errores, es importante realizar un análisis de los metadatos que se obtienen con el sistema y compararlos con lo registrado en Sunarp. El sistema se diseñó para almacenar los metadatos extraídos en un archivo de texto. Este archivo tiene una estructura definida que permite diferenciar las características

de cada vehículo al encontrarse en líneas distintas, además los datos se encuentran separados por una coma como se presenta en la Figura 104.

Figura 104

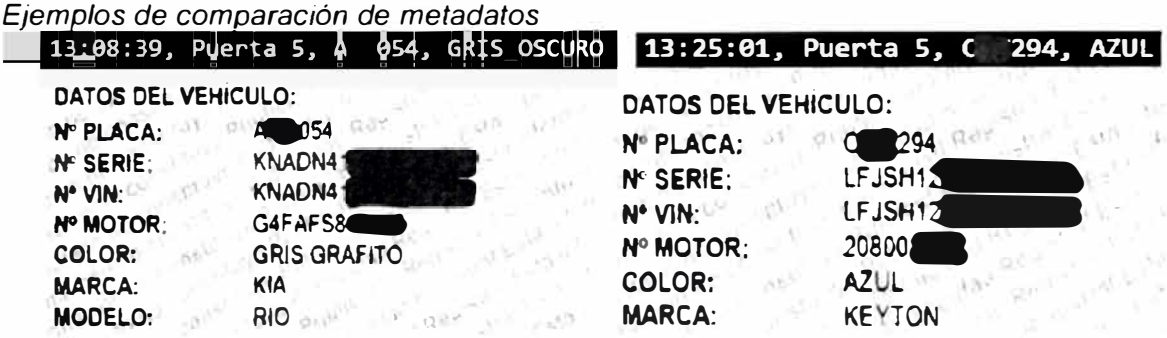
Estructura del archivo que almacena la metadata para identificación vehicular



Estos datos podrán ser evaluados al contrastarlos y verificar su validez, por lo que se podrán hallar el nivel de acercamiento a los datos reales que identifican a un vehículo. Además, si se usa como sistema de seguridad se podrá encontrar discrepancias con la placa, color, modelo o marca. Por ello, se acepta la hipótesis alterna y se descarta la hipótesis nula.

Figura 105

Ejemplos de comparación de metadatos



Conclusiones

- El desarrollo de un modelo personalizado para la detección de objetos ha resaltado la importancia de emplear la técnica de aprendizaje por transferencia para obtener altos valores de confianza, utilizando arquitecturas sólidas y probadas previamente.
- La introducción de un algoritmo para adquirir imágenes de vehículos junto con sus placas ha disminuido significativamente los costos en recursos y el tiempo necesario para la formación del conjunto de imágenes UNI-LimaLP.
- La inclusión del conjunto de imágenes UNI-LimaLP mejora significativamente los resultados que se podrían obtener con conjuntos de imágenes de placas extranjeras debido a sus grandes diferencias con las de Perú.
- La investigación reveló que la precisión de las versiones de YOLOv8 son superiores al 95%, siendo la versión mediana la que obtuvo una precisión del 97.7%.
- La versión de YOLOv8 nano posee la menor cantidad de parámetros y solo es 1% menor en precisión a la versión mediana, convirtiéndose en un modelo robusto y liviano.
- De las pruebas para la extracción de caracteres y el algoritmo implementado para usar EasyOCR, se obtuvo una precisión media de 96.6%.
- El uso de una arquitectura OCR que incluya modelos encargados de realizar tareas específicas, como CRAFT para localizar textos y Resnet para identificarlos, resulta una solución que brinda la confianza necesaria en sus resultados.
- EL uso del espacio HSV resulta eficaz para obtener el color de los vehículos ante condiciones de iluminación que permitan definir rangos de valores para cada color que se quiere detectar.

Recomendaciones

- Dado que el modelo YOLOv8-m posee las mejores métricas se recomienda su uso cuando no haya limitación en los recursos de hardware.
- Si se desea utilizar el sistema para identificación vehicular en equipos con recursos reducidos se recomienda el uso de la versión nano.
- Se recomienda usar una tarjeta gráfica para el entrenamiento e implementación de los modelos estudiados, con el objetivo de aumentar la velocidad de procesamiento de las imágenes.
- Se recomienda enlazar la base de datos de Sunarp para la identificación vehicular y en su defecto crear una base de datos relacional del tipo SQL para acceder a ella cuando se requiera.

Referencias bibliográficas

ActivaResearch. (2023). *Pulso Ciudadano*. *6*, 39–40. https://media.elmostrador.cl/2020/02/PULSO-CIUDADANO-FEBRERO-2020_Q1.pdf

Agarap, A. F. (2018). *Deep Learning using Rectified Linear Units (ReLU)*. *1*, 2–8. http://arxiv.org/abs/1803.08375

Agrawal, A. K., Shrivas, A. K., & kumar Awasthi, V. (2021). A Robust model for handwritten digit recognition using machine and deep learning technique. *2021 2nd International Conference for Emerging Technology (INCET)*, 1–4.

Aguilar Anaya, J. P. (2022). *Prototipo de reconocimiento de placas vehiculares para detección de vehiculos alertados en el complejo de control aduanero de Tomasiri, Tacna*. http://hdl.handle.net/20.500.12969/2429

Akhtar, N., & Ragavendran, U. (2020). Interpretation of intelligence in CNN-pooling processes: a methodological survey. *Neural Computing and Applications*, *32*(3), 879–898. https://doi.org/10.1007/s00521-019-04296-5

Al-Hmouz, R., & Challa, S. (2007). Intelligent stolen vehicle detection using video sensing. *Conference Proceedings of 2007 Information, Decision and Control, IDC*, 302–307. https://doi.org/10.1109/IDC.2007.374567

Al-Shemarry, M. S., Li, Y., & Abdulla, S. (2018). Ensemble of adaboost cascades of 3L-LBPs classifiers for license plates detection with low quality images. *Expert Systems with Applications*, *92*, 216–235. https://doi.org/https://doi.org/10.1016/j.eswa.2017.09.036

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, *2018-Janua*, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

Alghamdi, M. A., Alkhazi, I. S., & Teahan, W. J. (2016). Arabic OCR evaluation tool. *2016 7th International Conference on Computer Science and Information Technology*

(CSIT), 1–6.

Alves, F. R. V., Vieira, R. P. M., & Catarino, P. M. M. C. (2020). Visualizing the Newtons Fractal from the Recurring Linear Sequence with Google Colab: An Example of Brazil X Portugal Research. *International Electronic Journal of Mathematics Education*, *15*(3).

American Humanist Association. (2023). *Definition of Humanism*. https://americanhumanist.org/what-is-humanism/definition-of-humanism/

Asociación Automotriz del Perú. (2016a). *Oficinas Registrales*. Placas Ordinarias. https://aap.org.pe/placas/tipos/ordinarias/oficinas-registrales/

Asociación Automotriz del Perú. (2016b). *Placas ordinarias*. Tipos de Placas. https://aap.org.pe/placas/tipos/ordinarias/tipo-de-placas/

Asociación Automotriz del Perú. (2023). *Informe del sector automotor-Abril 2023*.

ASOCIACIÓN AUTOMOTRIZ DEL PERÚ. (2023). *DESEMPEÑO DE LA VENTA DE VEHÍCULOS NUEVOS SIGUE EL COMPORTAMIENTO DE LA ECONOMÍA NACIONAL*. https://aap.org.pe/sunarp-desempeno-de-la-venta-de-vehiculos-nuevos-sigue-el-comportamiento-de-la-economia-nacional/

Awalgaonkar, N., Bartakke, P., & Chaugule, R. (2021). Automatic License Plate Recognition System Using SSD. *2021 International Symposium of Asian Control Association on Intelligent Robotics and Industrial Automation, IRIA 2021*, *September*, 394–399. https://doi.org/10.1109/IRIA53009.2021.9588707

Bargmeyer, B. E., & Gillman, D. W. (2000). Metadata standards and metadata registries: An overview. *International Conference on Establishment Surveys II, Buffalo, New York*.

Barthélémy, J.-H. (2010). What New Humanism Today? *Cultural Politics*, *6*(2), 237–252. https://doi.org/https://doi.org/10.2752/175174310X12672016548441

Basha, S. H. S., Dubey, S. R., Pulabaigari, V., & Mukherjee, S. (2020). Impact of fully connected layers on performance of convolutional neural networks for image

classification. *Neurocomputing*, *378*, 112–119. https://doi.org/10.1016/j.neucom.2019.10.008

Beauxis-Aussalet, E., & Hardman, L. (2014). Simplifying the visualization of confusion matrix. *26th Benelux Conference on Artificial Intelligence (BNAIC)*.

Bernstein, D., & Kanaan, A. Y. (1993). Automatic vehicle identification: technologies and functionalities. *IVHS Journal Print*, *1*(2), 191–204. https://doi.org/10.1080/10248079308903792

Bonett, D. C., & Altamirano, P. A. (2023). *Estadística de Criminalidad, Seguridad Ciudadana y Violencia Informe N°3*.

Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc."

Bredies, K., Lorenz, D., & others. (2018). *Mathematical image processing*. Springer.

Burger, W., & Burge, M. J. (2016). Digital Image Processing an Algorithmic Introduction. In *Springer* (Vol. 24, Issue 11). Springer. http://www.springer.com/series/3191%0Ahttp://link.springer.com/10.1007/978-1-4471-6684-9%0Ahttp://digital-library.theiet.org/content/journals/10.1049/ep.1978.0474%5Cnhttp://link.springer.com/10.1007/978-1-4471-6684-9

Buscema, M. (1998). Back propagation neural networks. *Substance Use and Misuse*, *33*(2), 233–270.

Cabral, J. P., Dos Santos, V. G., Souza, C., Silva, A., Dantas, A., Cacho, N., Lopes, F., & Araujo, D. (2021). An Efficient CNN-based Approach for Automatic Brazilian License Plate Recognition. *Proceedings - 2021 International Conference on Computational Science and Computational Intelligence, CSCI 2021*, 1891–1894. https://doi.org/10.1109/CSCI54926.2021.00077

Campo Romero, J., & Cruz Camelo, J. (2016). *Prototipo de un sistema para controlar el acceso de vehículos y sus ocupantes al parqueadero de un conjunto residencial,*

implementando RFID y detección de huella digital (Issue 3). UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS.

Carhuavilca Bonett, D., & Escribano Pinaud, B. (2023). *Flujo vehicular por unidades de peaje*. *9*, 1–2.

Carhuavilca Bonett, D., & Peña Aldazabal, R. (2022). *Estadistica de la Criminalidad. Seguridad Ciudadana y Violencia*.

Chen, J., Hua, Z., Wang, J., & Cheng, S. (2017). A convolutional neural network with dynamic correlation pooling. *2017 13th International Conference on Computational Intelligence and Security (CIS)*, 496–499.

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., & Dahl, G. E. (2019). On empirical comparisons of optimizers for deep learning. *ArXiv Preprint ArXiv:1910.05446*.

Choi, R. Y., Coyner, A. S., Kalpathy-Cramer, J., Chiang, M. F., & Peter Campbell, J. (2020). Introduction to machine learning, neural networks, and deep learning. *Translational Vision Science and Technology*, *9*(2), 1–12. https://doi.org/10.1167/tvst.9.2.14

Cilimkovic, M. (2015). Neural networks and back propagation algorithm. *Institute of Technology Blanchardstown. Blanchardstown Road North Dublin*, *15*(1).

Descartes, R. (1970). Discourse on the method of rightly directing one's Reason and of seeking Truth in the Sciences. *Elizabeth Anscombe \& Peter T. Geach (Trans. Eds.). Descartes Philosophical Writings*, 5–57.

Dey, S. (2018). *Hands-On Image Processing with Python: Expert techniques for advanced image analysis and effective interpretation of image data*. Packt Publishing Ltd.

Ding, B., Qian, H., & Zhou, J. (2018). Activation functions and their characteristics in deep neural networks. *Proceedings of the 30th Chinese Control and Decision Conference, CCDC 2018*, 1836–1841. https://doi.org/10.1109/CCDC.2018.8407425

Du, S., Ibrahim, M., Shehata, M., & Badawy, W. (2013). Automatic license plate recognition (ALPR): A state-of-the-art review. *IEEE Transactions on Circuits and Systems for*

Video Technology, *23*(2), 311–325. https://doi.org/10.1109/TCSVT.2012.2203741

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(7).

Dwyer, B., Nelson, J., & Solawetz, J. (2022). *Roboflow*. https://roboflow.com.

El-Amir, H., & Hamdy, M. (2019). *Deep learning pipeline: building a deep learning model with TensorFlow*. Apress.

Elfwing, S., Uchibe, E., & Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, *107*, 3–11.

Elkhatib, Z., Ben Mnaouer, A., Mashaal, O., Azman Ismail, N., Azman Bin Abas, M., Abdulgaleel, F., & El Khatib, Z. (2021). *Parking System License Plate Detection Based on Convolution Neural Networks GPU Optimization*.

Elyousfi, M. (2023). *EasyOCR pipeline from A to Z*. https://medium.com/@mohamed5elyousfi/-277e9c685578

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2009). Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(9), 1627–1645.

Fernandez, E. (2019). *Análisis automático de radiografías torácicas por medio de redes neuronales profundas*. Universidad de Buenos Aires.

Fernández Marcellán, M. (2021). *Análisis de imagen aplicada a la gestión de aeropuertos*. Universitat Politècnica de Catalunya.

Gonazales, A. (2018). Aplicaciones de tècnicas de inteligencia artificial basadas en aprendizaje profundo (deep learning) al análisis y mejora de la eficiencia de procesos industriales. *Universidad De Oviedo*, *July*, 13–17. https://doi.org/10.13140/RG.2.2.15144.72967

Gonçalves, G. R., da Silva, S. P. G., Menotti, D., & Schwartz, W. R. (2016). A benchmark for license plate character segmentation. *Journal of Electronic Imaging*, *25*(5), 053034. https://doi.org/10.1117/1.jei.25.5.053034

González-Cepeda, J., Ramajo, Á., & Armingol, J. M. (2022). Intelligent Video Surveillance Systems for Vehicle Identification Based on Multinet Architecture. *Information (Switzerland)*, *13*(7). https://doi.org/10.3390/info13070325

Goutte, C., & Gaussier, E. (2005). A probabilistic interpretation of precision, recall and f-score. *European Conference on Information Retrieval*, 345–359.

Gunawan, T. S., Ashraf, A., Riza, B. S., Haryanto, E. V., Rosnelly, R., Kartiwi, M., & Janin, Z. (2020). Development of video-based emotion recognition using deep learning with Google Colab. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, *18*(5), 2463–2471.

Gundimeda, V., Murali, R. S., Joseph, R., & Naresh Babu, N. T. (2019). An automated computer vision system for extraction of retail food product metadata. *First International Conference on Artificial Intelligence and Cognitive Computing: AICC 2018*, 199–216.

Gupta, D. (2020). *Fundamentals of Deep Learning – Activation Functions and When to Use Them?* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/

Guptill, S. C. (1999). Metadata and data catalogues. *Geographical Information Systems*, *2*, 677–692.

Hahnloser, R. H. R., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., & Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, *405*(6789), 947–951.

Han, J., & Moraga, C. (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *930*, 195–201. https://doi.org/10.1007/3-540-59497-3_175

Hsu, G. S., Chen, J. C., & Chung, Y. Z. (2013). Application-oriented license plate

recognition. *IEEE Transactions on Vehicular Technology*, *62*(2), 552–561. https://doi.org/10.1109/TVT.2012.2226218

Hussain, M., Bird, J. J., & Faria, D. R. (2019). A study on CNN transfer learning for image classification. *Advances in Intelligent Systems and Computing*, *840*, 191–202. https://doi.org/10.1007/978-3-319-97982-3_16

Islam, M., Chen, G., & Jin, S. (2019). An Overview of Neural Network. *American Journal of Neural Networks and Applications*, *5*(1), 7. https://doi.org/10.11648/j.ajnna.20190501.12

JaidedAI. (2022). *EasyOCR*. https://github.com/JaidedAI/EasyOCR?tab=readme-ov-file

Jamtsho, Y., Riyamongkol, P., & Waranusast, R. (2020). Real-time Bhutanese license plate localization using YOLO. *ICT Express*, *6*(2), 121–124.

Jiang, P., Ergu, D., Liu, F., Cai, Y., & Ma, B. (2022). A Review of Yolo algorithm developments. *Procedia Computer Science*, *199*, 1066–1073.

Kahie, S. A., Nor, A. A., Hasan, A. H., Abdi, A. M., Hassan, L. M., & Mohamud, M. A. (2021). A smart access control for restricted buildings using vehicle number plates recognition system. *2021 1st International Conference on Emerging Smart Technologies and Applications, ESmarTA 2021*. https://doi.org/10.1109/eSmarTA52612.2021.9515752

Kelly, S. (2019). *Python, PyGame and raspberry Pi game development*. Springer.

Kessentini, Y., Besbes, M. D., Ammar, S., & Chabbouh, A. (2019). A two-stage deep neural network for multi-norm license plate detection and recognition. *Expert Systems with Applications*, *136*, 159–170. https://doi.org/10.1016/j.eswa.2019.06.036

Ketkar, N., Moolayil, J., Ketkar, N., & Moolayil, J. (2021). Introduction to pytorch. *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, 27–91.

Khazaee, S., Tourani, A., Soroori, S., Shahbahrami, A., & Suen, C. Y. (2020). A real-time license plate detection method using a deep learning approach. *International Conference on Pattern Recognition and Artificial Intelligence*, 425–438.

Kim, H. E., Cosa-Linan, A., Santhanam, N., Jannesari, M., Maros, M. E., & Ganslandt, T. (2022). Transfer learning for medical image classification: a literature review. *BMC Medical Imaging*, *22*(1), 1–13. https://doi.org/10.1186/s12880-022-00793-7

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*.

Kogalovsky, M. R. (2013). Metadata in computer systems. *Programming and Computer Software*, *39*(4), 182–193. https://doi.org/10.1134/S0361768813040038

Kriegeskorte, N. (2015). Deep neural networks: a new framework for modeling biological vision and brain information processing. *Annual Review of Vision Science*, *1*, 417–446.

Laroca, R., Cardoso, E., Lucio, D., Estevam, V., & Menotti, D. (2022). *On the Cross-dataset Generalization in License Plate Recognition*. 166–178. https://doi.org/10.5220/0010846800003124

Laroca, R., Severo, E., Zanlorensi, L. A., Oliveira, L. S., Goncalves, G. R., Schwartz, W. R., & Menotti, D. (2018). A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector. *Proceedings of the International Joint Conference on Neural Networks*, *2018-July*. https://doi.org/10.1109/IJCNN.2018.8489629

Laroca, R., Zanlorensi, L. A., Gonçalves, G. R., Todt, E., Schwartz, W. R., & Menotti, D. (2021). An efficient and layout-independent automatic license plate recognition system based on the YOLO detector. *IET Intelligent Transport Systems*, *15*(4), 483–503. https://doi.org/10.1049/itr2.12030

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lee, C. Y., Gallagher, P. W., & Tu, Z. (2016). Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016*, *51*, 464–472.

Li, H., Wang, P., & Shen, C. (2019). Toward End-to-End Car License Plate Detection and Recognition with Deep Neural Networks. *IEEE Transactions on Intelligent*

Transportation Systems, *20*(3), 1126–1136. https://doi.org/10.1109/TITS.2018.2847291

Lindsay, G. W. (2021). Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of Cognitive Neuroscience*, *33*(10), 2017–2031. https://doi.org/10.1162/jocn_a_01544

Louis March, P., & Glaveanu, V. (2020). Character Region Awareness for Text Detection. *Manual of Evidence-Based Admitting Orders and Therapeutics*, 215–221. https://doi.org/10.1016/B978-0-12-809324-5.23718-8

Lupu, M. (2021). Single-Layer Perceptron: the basic principles of construction and functioning. *The 12th International Conference on Intrinsic Josephson Effect and Horizons of Superconducting Spintronics*, 75.

Lytras, M. D., & Sicilia, M.-A. (2007). Where is the value in metadata? *International Journal of Metadata, Semantics and Ontologies*, *2*(4), 235–241.

Mäkelä, T. H. (2017). Technology and humanism. *6th International Alvar Aalto Meeting on Contemporary Architecture*, *10*(1), 6. https://doi.org/10.5935/1518-0557.2006.10.1.01

Mane, D. T., & Kulkarni, U. V. (2020). A survey on supervised convolutional neural network and its major applications. In *Deep Learning and Neural Networks: Concepts, Methodologies, Tools, and Applications* (pp. 1058–1071). IGI Global.

Mariani, M. M., Machado, I., Magrelli, V., & Dwivedi, Y. K. (2023). Artificial intelligence in innovation research: A systematic review, conceptual framework, and future research directions. *Technovation*, *122*(May 2022). https://doi.org/10.1016/j.technovation.2022.102623

Marius-Constantin, P., Balas, V. E., Perescu-Popescu, L., & Mastorakis, N. (2009). Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, *8*(7), 579–588.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133.

https://doi.org/10.1007/BF02478259

Medina, M. (1995). Tecnología y filosofía: más allá de los prejuicios epistemológicos y humanistas. *Isegoría*, *12*, 180–197. https://doi.org/10.3989/isegoria.1995.i12.249

Mena Roa, M., & Statista Research Department. (2023). *La producción mundial de vehículos aumentó un 6% en 2022*. https://es.statista.com/grafico/29579/evolucion-anual-del-numero-de-vehiculos-producidos-a-nivel-mundial/

Ministerio de Transporte y Comunicaciones. (2003). *Decreto Supremo N° 058- 2003-MTC, Reglamento nacional de vehículos*.

Decreto Supremo N° 017-2008-MTC, Reglamento de placa única nacional de rodaje, (2008). https://aap.org.pe/compendio/download/reglamento-de-placa-unica-nacional-de-rodaje/

Mithe, R., Indalkar, S., & Divekar, N. (2013). Optical character recognition. *International Journal of Recent Technology and Engineering (IJRTE)*, *2*(1), 72–75.

Montero Calderon, S., & Ponce Diaz, R. (2022). *Sistema de monitoreo para la detección automática de placas de vehículos en Lima Metropolitana utilizando redes neuronales*.

Moreira, M., & Fiesler, E. (1995). *Neural networks with adaptive learning rate and momentum terms*.

Morozov, E. (2013). The folly of technological solutionism. *LSE Public Lecture. March*, *21*.

Müller, B., Reinhardt, J., & Strickland, M. T. (1995). *Neural networks: an introduction*. Springer Science \& Business Media.

Nicanor Mariotti, E., Relloso, J., & Rojo Lapalma, F. (2020). *Detección e identificación de características de vehículos utilizando algoritmos de machine learning* (Issue 043). Universidad Nacional de Cuyo.

OpenALPR. (2016). *Openalpr-Eu Dataset*. https://github.com/openalpr/benchmarks/tree/master/endtoend

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359.

Patel, R., & Patel, S. (2020). A comprehensive study of applying convolutional neural network for computer vision. *International Journal of Advanced Science and Technology*, *29*(6 Special Issue), 2161–2174.

Prem, E. (2019). Artificial intelligence for innovation in Austria. *Technology Innovation Management Review*, *9*(12), 5–15. https://doi.org/10.22215/timreview/1287

Quoc, K. N., Van, D. P., & Bich, V. P. T. (2021). An efficient method to improve the accuracy of Vietnamese vehicle license plate recognition in unconstrained environment. *2021 International Conference on Multimedia Analysis and Pattern Recognition, MAPR 2021 - Proceedings*, 1–6. https://doi.org/10.1109/MAPR53640.2021.9585279

Rajeshwari, P., Abhishek, P., & Vinod, P. S. | T. (2019). Object Detection: An Overview. *International Journal of Trend in Scientific Research and Development*, *Volurne-3*(Issue-3), 1663–1665. https://doi.org/10.31142/ijtsrd23422

Ramirez Mejía, B., & Tito Apaza, M. (2020). *Reconocimiento automático de placas de rodaje utilizando una red neuronal convolucional para el ingreso de vehículos en la universidad Ricardo Palma* [Universidad Ricardo Palma]. https://hdl.handle.net/20.500.14138/4972

Rasamoelina, A. D., Adjailia, F., & Sincak, P. (2020). A Review of Activation Function for Artificial Neural Network. *SAMI 2020 - IEEE 18th World Syrnposiurn on Applied Machine Intelligence and Informatics, Proceedings*, 281–286. https://doi.org/10.1109/SAMI48414.2020.9108717

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 779–788.

Ley N° 27181, Ley General de Transporte y Tránsito Terrestre, 5 (2020). https://www.gob.pe/institucion/congreso-de-la-republica/normas-legales/9868-27181

Riedmiller, M., & Braun, H. (1993). Direct adaptive method for faster backpropagation learning: The RPROP algorithm. *1993 IEEE International Conference on Neural*

Networks, 586–591. https://doi.org/10.1109/icnn.1993.298623

Rojas, R. (1996). The backpropagation algorithm. *Neural Networks: A Systematic Introduction*, 149–182.

Sakib, S., Ahmed, N., Kabir, A. J., & Ahmed, H. (2018). An Overview of Convolutional Neural Network: Its Architecture and Applications. *Preprints 2018, February*. https://doi.org/10.20944/preprints201811.0546.v4

Salazar Márquez, M. B. (2014). Desarrollo de una algoritmo para la localización automática de placas vehiculares peruanas usando técnicas de procesamiento de imágenes. In *Pontificia Universidad Católica del Perú*. http://hdl.handle.net/20.500.12404/6215

Sazli, M. H. (2006). A brief review of feed-forward neural networks. *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*, *50*(01).

Sharma, S., Sharma, S., & Athaiya, A. (2020). Activation Functions in Neural Networks. *International Journal of Engineering Applied Sciences and Technology*, *04*(12), 310–316. https://doi.org/10.33564/ijeast.2020.v04i12.054

Shashirangana, J., Padmasiri, H., Meedeniya, D., & Perera, C. (2021). Automated license plate recognition: A survey on methods and techniques. *IEEE Access*, *9*, 11203–11225. https://doi.org/10.1109/ACCESS.2020.3047929

Shea, K. O., & Nash, R. (n.d.). *An Introduction to Convolutional Neural Networks*. 1–11.

Shih, F. Y. (2017). *Image processing and mathematical morphology: fundamentals and applications*. CRC press.

Shinde, P. P., & Shah, S. (2018). A Review of Machine Learning and Deep Learning Applications. *Proceedings - 2018 4th International Conference on Computing, Communication Control and Automation, ICCUBEA 2018*, 1–6. https://doi.org/10.1109/ICCUBEA.2018.8697857

Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE Access*, *7*, 53040–53065. https://doi.org/10.1109/ACCESS.2019.2912200

Silvano, G., Ribeiro, V., Greati, V., Bezerra, A., Silva, I., Endo, P. T., & Lynn, T. (2021). Synthetic image generation for training deep learning-based automated license plate recognition systems on the Brazilian Mercosur standard. *Design Automation for Embedded Systems*, *25*(2), 113–133. https://doi.org/10.1007/s10617-020-09241-7

Simondon, G. (1989). *Du mode d'existence des objets techniques*. Aubier.

Soh, Y. S., Chun, B. T., & Yoon, H. S. (1994). Design of real time vehicle identification system. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, *3*, 2147–2152. https://doi.org/10.1109/icsmc.1994.400182

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, 1–14.

Statista Research Department. (2023). *Number of cars sold worldwide from 2010 to 2022, with a 2023 forecast*. https://www.statista.com/statistics/200002/international-car-sales-since-1990/

Sun, M., Song, Z., Jiang, X., Pan, J., & Pang, Y. (2017). Learning Pooling for Convolutional Neural Network. *Neurocomputing*, *224*, 96–104. https://doi.org/10.1016/j.neucom.2016.10.049

Svozil, D., Kvasnicka, V., & Pospichal, J. (1997). Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems*, *39*(1), 43–62.

Szandała, T. (2021). Review and comparison of commonly used activation functions for deep neural networks. *Bio-Inspired Neurocomputing*, 203–224.

Tapia, H., Ghorai, M., Hsu, Y. C., & Jahic, A. (2022). 2022 Special Report on Human Security. In *2022 Special Report on Human Security*. https://doi.org/10.18356/9789210014007

Terven, J. (2023). A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *Machine Learning and Knowledge Extraction*, *5*(4), 1680–1716. https://doi.org/10.3390/make5040083

The Global Initiative. (2023a). *Organized crime index Peru Profile*.

The Global Initiative. (2023b). *Global organized crime index*. https://ocindex.net/

The Mathworks Inc. (2022). *Tangente hiperbólica-tanh*. https://es.mathworks.com/help/matlab/ref/tanh.html

Thoma, M. (2017). *Analysis and Optimization of Convolutional Neural Network Architectures*. *August*. http://arxiv.org/abs/1707.09725

Tkaczyk, D., Szostek Paweł and Fedoryszak, M., Dendek, P. J., & Bolikowski, Ł. (2015). CERMINE: automatic extraction of structured metadata from scientific literature. *International Journal on Document Analysis and Recognition (IJDAR)*, *18*, 317–335.

Vinjit, B. M., Bhojak, M. K., Kumar, S., & Chalak, G. (2020). A review on handwritten character recognition methods and techniques. *2020 International Conference on Communication and Signal Processing (ICCSP)*, 1224–1228.

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience*, *2018*. https://doi.org/10.1155/2018/7068349

Vrbančič, G., & Podgorelec, V. (2020). Transfer learning with adaptive fine-tuning. *IEEE Access*, *8*, 196197–196211. https://doi.org/10.1109/ACCESS.2020.3034343

Wang, C.-Y., Liao, H.-Y. M., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W., & Yeh, I.-H. (2020). CSPNet: A new backbone that can enhance learning capability of CNN. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 390–391.

Xu, D., Zhang, S., Zhang, H., & Mandic, D. P. (2021). Convergence of the RMSProp deep learning method with penalty for nonconvex optimization. *Neural Networks*, *139*, 17–23.

Xu, Z., Yang, W., Meng, A., Lu, N., Huang, H., Ying, C., & Huang, L. (2018). Towards end-to-end license plate detection and recognition: A large dataset and baseline. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence*

and Lecture Notes in Bioinformatics), *11217 LNCS*, 261–277. https://doi.org/10.1007/978-3-030-01261-8_16

Yacouby, R., & Axman, D. (2020). *Probabilistic Extension of Precision, Recall, and F1 Score for More Thorough Evaluation of Classification Models*. 79–91. https://doi.org/10.18653/v1/2020.eval4nlp-1.9

Yuan, Y., Zou, W., Zhao, Y., Wang, X., Hu, X., & Komodakis, N. (2017). A Robust and Efficient Approach to License Plate Detection. *IEEE Transactions on Image Processing*, *26*(3), 1102–1114. https://doi.org/10.1109/TIP.2016.2631901

Zaidi, S. S. A., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M., & Lee, B. (2022). A survey of modern deep learning based object detection models. *Digital Signal Processing*, *126*, 103514.

Zhang, S., Chi, C., Yao, Y., Lei, Z., & Li, S. Z. (2020). *Bridging the Gap Between Anchor-Based and Anchor-Free Detection via Adaptive*. 9759–9768.

Zhang, Z. (2016). A gentle introduction to artificial neural networks. *Annals of Translational Medicine*, *4*(19).

Zhao, D., & Lorin, E. (2020). *Image Recognition by Convolutional Neural Networks*. Carleton University.

Zhao, X., Ma, H., Zhang, H., Tang, Y., & Fu, G. (2014). Metadata extraction and correction for large-scale traffic surveillance videos. *Proceedings - 2014 IEEE International Conference on Big Data, IEEE Big Data 2014*, 412–420. https://doi.org/10.1109/BigData.2014.7004258

Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019). Object Detection with Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(11), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865

Zhou, W., Li, H., Lu, Y., & Tian, Q. (2012). Principal visual word discovery for automatic license plate detection. *IEEE Transactions on Image Processing*, *21*(9), 4269–4279. https://doi.org/10.1109/TIP.2012.2199506

Anexos.

Anexo 1: Matriz de consistencia para la tesis: "Aplicación de algoritmos de aprendizaje profundo con análisis de metadata para identificación vehicular"

Problema General	***Objetivo General***	***Hipótesis General***	***Variables***	***Dimensiones***
¿De qué manera se relacionan la aplicación de algoritmos de aprendizaje profundo con análisis de metadata y la identificación vehicular?	*Implementar algoritmos de aprendizaje profundo con análisis de metadata capaz de permitir la identificación vehicular.*	*Con la aplicación de algoritmos de aprendizaje profundo con análisis de metadata se realizará la identificación vehicular*	***Variable Independiente:*** *X1. Algoritmos de aprendizaje profundo con análisis de metadata*	*D1. Arquitectura aprendizaje profundo basada en redes neuronales convolucionales* *D2. Algoritmo de optimización* *D3. Reconocimiento de caracteres* *D4. Metadata*
			Variable Dependiente: *Y1. Identificación vehicular*	*D1. Formato de numeración* *D2. Matricula* *D3. Color* *D4. Metadata Sunarp*

Problemas Específicos	***Objetivos Específicos***	***Hipótesis Específicas***
a. ¿Qué relación existe entre la identificación vehicular y las arquitecturas de aprendizaje profundo basadas en redes neuronales convolucionales? *b. ¿De qué manera la aplicación de algoritmos de optimización aplicados a redes neuronales convolucionales es eficaz para la identificación vehicular?* *c. ¿De qué modo influye el reconocimiento de caracteres en la identificación vehicular?* *d. ¿En qué medida se puede relacionar la evaluación de la metadata con la identificación vehicular?*	*a. Implementar arquitecturas de aprendizaje profundo basadas en redes neuronales convolucionales para identificación vehicular.* *b. Implementar algoritmos de optimización aplicados a redes neuronales convolucionales para identificación vehicular.* *c. Implementar algoritmos de reconocimiento de caracteres para identificación vehicular.* *d. Evaluar la influencia de la metadata y algoritmos para la identificación vehicular.*	*a. La implementación de arquitecturas de aprendizaje profundo basadas en redes neuronales convolucionales es eficaz en la identificación vehicular.* *b. La implementación de algoritmos de optimización aplicados a redes neuronales convolucionales es relevante para la identificación vehicular.* *c. La aplicación de un correcto algoritmo de reconocimiento de caracteres es significativa para la identificación vehicular.* *d. La evaluación de la metadata y algoritmos resulta eficaz para la identificación vehicular.*

Anexo2: Códigos de Python

```python
import cv2
import time
import numpy as np
import easyocr
from Funciones import *
from datetime import datetime
import os
from ultralytics import YOLO


# Constantes
CONFIDENCE_THRESHOLD = 0.85
NMS_THRESHOLD = 0.7
COLORS = [(0, 0, 255), (0, 0, 255), (0, 0, 255), (255, 0, 0)]
OUTPUT_SIZE = (1280, 720)
CROP_OUTPUT_SIZE = (650, 200)
FONT = cv2.FONT_HERSHEY_SIMPLEX
FONT_SCALE = 0.7
FONT_COLOR = (48, 66, 227)
colores = {
    "ROJO": [(0, 100, 100), (18, 255, 255)],
    "ROJO": [(160, 100, 100), (180, 255, 255)],

    "GUINDA": [(0, 70, 30), (10, 255, 150)],
    "GUINDA": [(170, 70, 30), (180, 255, 150)],

    "ROJO_PALIDO": [(0, 30, 200), (10, 255, 255)],
    "ROJO_GRANATE": [(0, 60, 20), (10, 255, 150)],

    "ROSADO_CLARO": [(140, 50, 200), (170, 255, 255)],
    "ROSADO_MEDIO": [(140, 50, 150), (170, 255, 255)],
    "ROSADO_OSCURO": [(140, 50, 100), (170, 255, 255)],

    "NARANJA": [(10, 100, 100), (25, 255, 240)],
    "NARANJA_OSCURO": [(0, 100, 40), (10, 255, 100)],

    "AMARILLO": [(20, 50, 50), (35, 255, 255)],
    "AZUL": [(100, 100, 100), (140, 255, 255)],
    "AZUL_CLARO": [(85, 100, 100), (115, 255, 255)],
    "AZUL_OSCURO": [(90, 50, 50), (130, 255, 120)],
    "AZUL_MARINO": [(100, 100, 40), (140, 255, 100)],
    "AZUL_REAL": [(85, 100, 100), (125, 255, 255)],
    "CELESTE": [(85, 80, 100), (110, 255, 255)],
    "INDIGO": [(100, 100, 100), (140, 255, 255)],

    "VIOLETA": [(130, 100, 100), (160, 255, 255)],
    "MAGENTA": [(140, 100, 100), (170, 255, 255)],
    "VINO": [(140, 50, 20), (170, 80, 73)],
    "LAVANDA": [(110, 40, 100), (140, 255, 255)],
    "LAVANDA_CLARO": [(110, 40, 150), (140, 160, 255)],
    "LAVANDA_OSCURO": [(110, 40, 80), (140, 160, 200)],

    "NEGRO": [(0, 0, 0), (179, 255, 50)],
    "BLANCO": [(0, 0, 200), (179, 30, 255)],
    "GRIS": [(0, 0, 100), (179, 5, 180)],
    "GRIS METALICO": [(0, 0, 120), (179, 20, 220)],
    "GRIS_CLARO": [(0, 0, 150), (179, 30, 200)],
    "GRIS_OSCURO": [(0, 0, 40), (179, 40, 100)],

    "MARRON": [(0, 20, 20), (30, 255, 200)],
    "BEIGE": [(20, 40, 150), (40, 255, 255)],
    "TOSTADO": [(0, 80, 100), (20, 255, 200)],
    "CHOCOLATE": [(0, 80, 40), (20, 255, 150)],
    "CAQUI": [(30, 40, 150), (50, 150, 255)],
    "CREMA": [(0, 40, 180), (40, 160, 255)],
    "MELOCOTON": [(0, 80, 200), (20, 255, 255)],

    "ORO": [(20, 100, 100), (40, 255, 255)],
    "PLATA": [(0, 0, 120), (179, 60, 180)],
    "BRONCE": [(10, 80, 100), (30, 255, 200)],
    "PERLA": [(0, 0, 180), (30, 30, 255)],
    "CORAL": [(0, 100, 100), (10, 255, 255)],
    "ORO_ROSA": [(0, 80, 160), (15, 255, 255)],
}
```

```python
# Configuraciones y parametros
config = {
    "input_size": (416, 416),
    "scale": 1 / 255,
    "swap_rb": True,
    'model_path': './best_small.pt',
    "video_path": "rtsp://admin:*****@******:554/Streaming/channels/101",
}


def main(config):
    # Inicializacion
    read = easyocr.Reader(['en'], gpu=True)
    model = YOLO(config['model_path'])
    cap = cv2.VideoCapture(config["video_path"])

    # Variables
    salida = [[] for _ in range(6)]
    valor_f = ""
    pos = 0
    tiempo_bbox = 2668269365.4799159
    best_conf = 0
    crop_color=None
    while cap.isOpened():
        # Procesamiento de imagen
        grabbed, frame = cap.read()
        if not grabbed:
            cap.release()
            cap = cv2.VideoCapture(config['video_path'])
            cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)

            continue
        if grabbed:
            start = time.time()

            # --------------------- Prediccion ---------------------
            results = model.predict(
                                    frame,
                                    conf=CONFIDENCE_THRESHOLD,
                                    iou=NMS_THRESHOLD,
                                    verbose=False,
                                    max_det=1,)

            #------------------------------------------------------

            frame_predicted = results[0].plot()
            boxes = results[0].boxes

            tiempo_frame = time.time()

            for box in boxes:
                f = True

                x, y, x_1, y_1 = box.xyxy[0].tolist()
                x, y, x_1, y_1 = int(x), int(y), int(x_1), int(y_1)

                w = x_1 - x
                h = y_1 - y

                if any(salida) and (pos - x) > 100:
                    salida = [[] for _ in range(6)]
                    f = False
                    break

                crop_img = frame[y:y+h, x:x + w]

                crop_color = frame[y-50:y+h, (x-500):x-200+ w]
                cv2.imshow("Color", crop_color)
```

```
                gran = get_grayscale(crop_img)
                gran = get_resize(gran, size=CROP_OUTPUT_SIZE)

                bl = cv2.GaussianBlur(gran, (7, 7), 1.5, borderType=cv2.BORDER_REFLECT)
                bil = remove_noise(bl)

                res = read.readtext(bil, allowlist='0123456789ABCDEFGHIJKLMNOPQRSTUWXYZV')
                #print(res)

                for n in range(len(res)):
                    text = res[-(n + 1)][1]

                    if license_complies_format(text):
                        text = format_license(text)
                        print("---------------")
                        print(text)
                        print("---------------")
                        for i in range(6) :
                            salida[i].append(text[i])
                        # Imprimimos el valor en las listas de salida
                        print("---------------")
                        for i in range(6):
                            print(salida[i])
                        print("---------------")
                        valor_f = ''.join([most_freq(s) for s in salida])

                        if res[-(n + 1)][2] > best_conf:
                            best_conf = res[-(n + 1)][2]
                            # reducir el tamaño de la imagena la mitad
                            best_crop = cv2.resize(gran, (0,0), fx=0.5, fy=0.5)

                        cv2.putText(frame_predicted, valor_f, (x, y+h+20), FONT, FONT_SCALE, FONT_COLOR, 2)

                pos = x
                tiempo_bbox = time.time()

            if (tiempo_frame - tiempo_bbox) > 0.5 and f:
                now = datetime.now()

                if crop_color is not None:
                    color = detectar_color(crop_color, colores)

                if valor_f != '':
                    with open("LOG_Prueba.txt", "a") as file:
                        data = [str(now.strftime('%Y-%m-%d %H:%M:%S')) + ", " + valor_f + "," + color + "\n"]
                        file.writelines(data)
                    print("Valor final", valor_f)

                    carpeta = './plates/'
                    cv2.imwrite(f'{carpeta}/{valor_f}.jpg', best_crop)
                    best_conf = 0

                f = False

            end = time.time()
            fps = 'FPS: %.2f ' % (1 / (end - start))
            cv2.putText(frame_predicted, fps, (0, 25), FONT, 1, (255, 0, 255), 2)
            cv2.imshow("YOLOv8 Detection", cv2.resize(frame_predicted, OUTPUT_SIZE))

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        else:
            break

    cap.release()
    cv2.destroyAllWindows()


if __name__ == "__main__":
    main(config)
```

```python
import string
import cv2

# get grayscale image
def get_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# resize the image
def get_resize(image, size):
    # return cv2.resize(image, None, fx=7, fy=7, interpolation=cv2.INTER_NEAREST_EXACT)
    return cv2.resize(image, size, interpolation=cv2.INTER_CUBIC)

def most_freq(list):
    if len(list) >= 1:
        num0 = max(set(list), key = list.count)
        return num0

# noise removal
def remove_noise(image):
   return cv2.bilateralFilter(image, 5, 69, 69)
    #return cv2.bilateralFilter(image, 5, 35, 35)


#Correcion de placas

# Mapping dictionaries for character conversion
dict_char_to_int = {'O': '0',
                    'I': '1',
                    'J': '3',
                    'A': '4',
                    'G': '6',
                    'S': '5'}

dict_int_to_char = {'0': 'O',
                    '1': 'I',
                    '3': 'B',
                    '4': 'A',
                    '6': 'G',
                    '5': 'S'}

def license_complies_format(text):
    if len(text) < 6:
        return False

    elif (text[0] in string.ascii_uppercase or text[0] in dict_int_to_char.keys()) and \
       (text[2] in string.ascii_uppercase or text[2] in dict_int_to_char.keys()) and \
       (text[3] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or text[3] in dict_char_to_int.keys()) and \
       (text[4] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or text[4] in dict_char_to_int.keys()) and \
       (text[5] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or text[5] in dict_char_to_int.keys()):
        return True
    else:
        return False
```

```python
def format_license(text):
    license_plate_ = ''
    mapping = {0: dict_int_to_char, 2: dict_int_to_char, 4: dict_int_to_char, 5: dict_char_to_int,
               3: dict_char_to_int, 4: dict_char_to_int}

    for j in [0, 1, 2, 3, 4, 5]:
        if j==1:
            license_plate_ += text[j]
        elif text[j] in mapping[j].keys():
            license_plate_ += mapping[j][text[j]]
        else:
            license_plate_ += text[j]

    return license_plate_


def detectar_color(frame, colores):
    if frame is not None:
        frame = cv2.resize(frame, (1000, 500))
        hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        height, width, _ = frame.shape
        cx = int(width / 2)
        cy = int(height / 2)
        pixel_center = hsv_frame[cy, cx]
        print(pixel_center)

        color = "No Definido"
        for color_name, (lower_range, upper_range) in colores.items():
            if (lower_range[0] <= pixel_center[0] <= upper_range[0] and
               lower_range[1] <= pixel_center[1] <= upper_range[1] and
               lower_range[2] <= pixel_center[2] <= upper_range[2]):
                color = color_name
                break

        pixel_center_bgr = frame[cy, cx]
        b, g, r = int(pixel_center_bgr[0]), int(pixel_center_bgr[1]), int(pixel_center_bgr[2])

        # Agrega el texto al frame
        cv2.rectangle(frame, (cx - 220, 10), (cx + 200, 120), (255, 255, 255), -1)
        cv2.putText(frame, color, (cx - 200, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (b, g, r), 5)
        frame = cv2.resize(frame, (1280, 720))
        # Muestra el frame editado
        cv2.imshow("Frame Editado", frame)
    return color
```