

Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación



T E S I S

**Análisis Comparativo de Técnicas de
Preprocesamiento de Imágenes y Datos
LiDAR para la Optimización del
Entrenamiento de Redes Neuronales en
Sistemas para la Conducción Autónoma**

Presenta: *Jorge Osorno Viveros*

Para obtener el grado de: Ingeniería en Ciencias de la
Computación

Director: *Dra. Beatriz Beltrán Martínez*

Codirector: *Dr. David Eduardo Pinto Avedaño*

Puebla, Pue, Marzo 2026

Dedicatoria

A mi hermano Jesus, a mis padres y a mi novia Are, por siempre apoyarme durante cada paso que tomo para alcanzar nuevos horizontes... Les amo mucho.

Agradecimientos

Al laboratorio de Ingeniería del Lenguaje y el Conocimiento por el vehículo, los recursos y el apoyo de los investigadores para lograr este proyecto. Al Doctor David Pinto por abrirme las puertas al laboratorio e integrarme a su equipo. A la Doctora Beatriz Beltrán por su asesoría y apoyo durante este proceso. Al Doctor Rigoberto Jimenez por enseñarme sobre robótica y redes convolucionales. Y al Doctor Abraham Sánchez por todo su apoyo en mi aprendizaje del Machine Learning y Deep Learning.

Índice

Resumen	5
Introducción	5
Capítulo 1: Estado del arte	9
1.1 Primera ola de la Inteligencia Artificial	10
1.1.1 Función de activación y su importancia	12
1.2 Segunda ola de la Inteligencia Artificial	13
1.3 Tercera ola de la Inteligencia Artificial	14
Capítulo 2: Marco Teórico	16
2.1 Redes Neuronales Convolucionales	16
2.1.1 Capa convolucional (Convolutional layer)	16
2.1.2 Capa de agrupación (Pooling layer)	17
2.1.3 Capa de aplanamiento (Flatten layer)	17
2.1.4 Capa totalmente conectada (Fully connected layer)	17
2.2 Vision Transformer	18
2.2.1 Embedding posicional	19
2.2.2 Encoder Transformer	19
2.3 ¿Qué es la conducción autónoma?	20
2.3.1 Niveles de conducción autónoma	21
2.3.1.1 Nivel 0: Sin automatización en la conducción	22
2.3.1.2 Nivel 1: Asistencia de conducción.	22
2.3.1.3 Nivel 2: Automatización parcial.	23
2.3.1.4 Nivel 3: Automatización Condicional.	24
2.3.1.5 Nivel 4: Alta automatización.	24
2.3.1.6 Nivel 5: Total automatización.	25
2.3.2 LiDAR	26
2.3.3 ¿Qué es el preprocesamiento de imágenes?	27
2.3.4 Técnicas de preprocesamiento de imágenes.	28
2.3.4.1 Técnicas de corrección radiométrica.	28
2.3.4.1.1 Métodos de dominio espacial	28
2.3.4.1.2 Métodos de dominio de frecuencia:	35
2.3.5 Buscadores de bordes de convolución	36
2.3.6 Técnicas de preprocesamiento de imágenes en la conducción autónoma	46
Capítulo 3: Diseño	49
3.1 Metodología	50
3.1.1 Infraestructura	51
Capítulo 4: Implementación, pruebas y resultados	57
4.1 Recolección de datos	57
4.2 Análisis y limpieza de los datos	60
4.3 Integración de los datos	61
4.4 Preprocesamiento	69

4.5 Entrenamiento	77
4.5.1 Entrenamiento CNN	78
4.5.2 Entrenamiento ViT	81
4.6 Comparación	84
Capítulo 5: Conclusiones	90
5.1 Propuesta de arquitectura	91
5.2 Trabajo futuro	91
Referencias	93

Resumen

El presente trabajo tiene por objetivo la aplicación de distintas técnicas de preprocesamiento de imágenes y datos LIDAR para la optimización del entrenamiento de redes neuronales en sistemas de conducción autónoma. El objetivo principal consiste en entrenar distintas redes de *Deep Learning* (DL) con datos preprocesados con distintas técnicas en imágenes, para evaluar mejoras o reducción en el rendimiento en la conducción autónoma de un vehículo tipo Ackerman.

Se espera que este proyecto ayude a entender un poco más sobre cómo funcionan las redes DL así como conocer qué características influyen en su desempeño global por medio de la aplicación de algoritmos para el entrenamiento.

Introducción

La Inteligencia Artificial (IA) tiene su nacimiento en la *Dartmouth Conference* en 1956 y desde entonces han aparecido diversas definiciones según el autor y el enfoque al que se haga referencia, algunas definiciones que encontramos son las siguientes:

- La inteligencia artificial es “la ciencia e ingeniería de hacer máquinas inteligentes” (McCarthy, 1955, como se citó en *Human-Centered Artificial Intelligence* - Stanford University, s.f.)
- “Any artificial system that performs tasks under varying and unpredictable circumstances without significant human oversight, or that can learn from experience and improve performance when exposed to data sets.” (National Aeronautics and Space Administration, 2024)

Con base en las definiciones anteriores podemos llegar a una definición propia de que la inteligencia artificial se refiere a la rama del conocimiento cuyo objeto de estudio es crear computadoras capaces de aprender a realizar tareas que requieren esfuerzo mental de manera similar a como lo hacen los seres vivos.

Dentro del campo de la inteligencia artificial podemos observar que se han ido desarrollando nuevos subcampos según el tipo de algoritmos que se usan y el alcance que tienen para desempeñar tareas. Un nivel más profundo de la inteligencia artificial se refiere al Machine Learning (ML) el cual se define como:

- “Machine Learning es el campo de estudio que le da a las computadoras la habilidad de aprender sin ser programadas explícitamente”(Brown, 2021)

Una definición más moderna fue propuesta por Tom Mitchell en 1997:

- “Machine Learning es el campo que se interesa en cómo construir programas de computadora que mejoran automáticamente con la experiencia” (Mitchell, 1997).

Para que los algoritmos de ML funcionen de manera correcta, dependen de manera significativa de la elección de la representación de los datos o características que son aplicadas (Bengio et al., 2012). Para dar solución y poder elegir las mejores características de los problemas se desarrolló el subcampo de la representación del conocimiento (RL por sus siglas en inglés). Siendo el conjunto de técnicas que permiten aprender representaciones directamente de los datos. En este subcampo se busca transformar los datos brutos en una representacional adecuada para una tarea a menudo a través de múltiples capas que logran abstracción para obtener información relevante.

El siguiente subcampo de la IA y el ML que es el Deep Learning (DL) o Aprendizaje Profundo. Ian Goodfellow, Yoshua Bengio y Aaron Courville definen el aprendizaje profundo como un tipo de aprendizaje automático el cual permite a las computadoras aprender conceptos complejos mediante la construcción de conceptos más simples, organizados jerárquicamente por capas (Goodfellow et al., 2016). Su principal característica es el uso de redes neuronales profundas en las cuales cada capa se especializa en una tarea específica. De esta manera se pueden obtener representaciones más complejas del problema abordado, por medio de la jerarquización de características o extracción de patrones profundos sin la

necesidad de programarlos explícitamente. Dentro de los algoritmos del DL se encuentran las *Recurrent Neural Networks* (RNN), *Convolutional Neural Networks* (CNN), *Graph Neural Network* (GNN) por mencionar algunas (Amini, 2025). Una de las mejoras de estos algoritmos a otros del campo de la IA, es su capacidad de trabajar con datos no estructurados tales como imágenes, audio, video o texto; ya sea de manera autónoma o en conjunto logrando resultados más complejos, como:

- Detección y reconocimiento de voz por medio de RNN's
- Generar texto como los language models que en un principio usaron RNN's
- Detectar características de imágenes por medio de las CNN's
- Generar una descripción de un texto por medio de una imagen combinando RNN's y CNN's

En el presente trabajo se hizo uso específicamente de la arquitectura de redes neuronales convolucionales (CNN por sus siglas en inglés), las cuales fueron concebidas con el objetivo de procesar imágenes para poder dotar de un modo de procesamiento a la rama de la visión computacional, permitiendo de esta manera trabajar con cualquier dispositivo que nos permite capturar fotografías y video para posteriormente extraer información de estos.

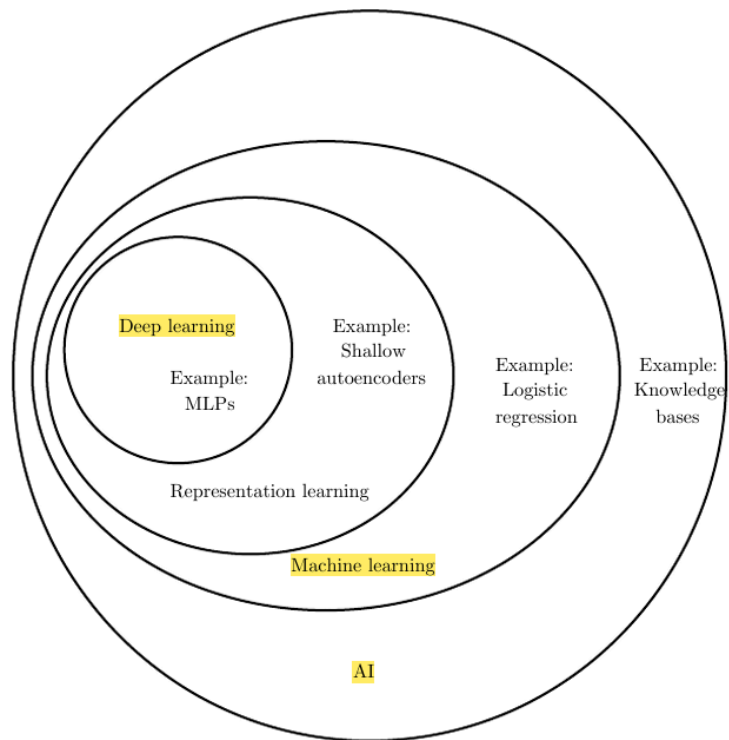


Figura 1: Subcampos de la inteligencia artificial mostrados mediante un diagrama de Venn. Cada diagrama representa una tecnología en base a sus principios y maneras de resolución. Tomado de (deep learning book, Ian Goodfellow et al.2015 pág. 24).

Capítulo 1: Estado del arte

El subcampo del *Deep Learning* como parte de la Inteligencia Artificial pertenece a una de las áreas que más rápido se han desarrollado en las últimas décadas. Y aunque tiene sus inicios en los pensamientos filosóficos de si una máquina puede pensar, con el paso de las décadas ha evolucionado de distintas maneras, pasando por distintas corrientes las cuales se han enfocado en dotar de capacidades a las computadoras para resolver problemas cada vez más complejos y abstractos.

Con sus fundaciones en la idea de crear máquinas que puedan resolver problemas como los humanos, resultó ser que estas son buenas para resolver problemas de tipo numérico dejando a los seres humanos muy atrás. Sin embargo, para tareas que en un principio son sencillas para los seres humanos, como reconocer objetos, voces o incluso olores, resultó no ser tan sencillo; abriendo paso al área de investigación de algoritmos que pudieran imitar mejor estas funciones del ser humano. Por ello, los primeros algoritmos fueron diseñados con inspiración del cerebro biológico (ya fuese el humano o el de algún otro animal).

Desde el primer avance en 1943 con el modelo de neurona artificial de McCulloch y Pitts cuyo modelo lineal podía reconocer dos categorías diferentes (positivo o negativo) (Springer, 2019) en donde aún no se acuñaba el término tal cual de IA, para después ir asentándose en el ámbito científico con el artículo publicado por Alan Turing “*Computing Machinery and Intelligence*” a mediados del siglo XX (Turing, 1950). Hasta su nacimiento oficial en 1956 en la acuñada “*Dartmouth Summer Research Project on Artificial Intelligence*” en Hanover, New Hampshire, EE. UU. realizada por John McCarthy.

Esta área se ha ido desarrollando en tendencias con distintos nombres que reflejan la manera en la que se abordaban los problemas no solo de manera técnica sino filosófica.

1.1 Primera ola de la Inteligencia Artificial

En las décadas de 1940-1960 esta área se conoce como *Cybernetics*, donde su principal enfoque fue lograr el control y la comunicación en animales y máquinas, por medio de mecanismos de retroalimentación (Heylighen & Joslyn, 2001).

Un hito importante en esta época fue la creación del perceptrón, el cual fue presentado en el paper "*The perceptron: A probabilistic model for information storage and organization in the brain*" (Rosenblatt, 1958), donde responde a la pregunta de cómo la información del mundo físico es medida, recordada para luego influenciar el comportamiento humano. En su trabajo propuso el modelo de la neurona artificial, la cual aprendía de ejemplos modificando los pesos (sinapsis) por medio de ejemplos para poder clasificar problemas.

La neurona artificial se compone de las señales de entrada de un vector X cuyos valores se encargan de traer la información del exterior para poder procesarla en la neurona. Cada una de las señales es multiplicada por un peso sináptico (w), el cual es generado para modificar la información de entrada a lo que se esperaría de salida.

Acto seguido lo que se realiza es que todos estos pesos se suman para tener un único valor agregando un sesgo (bias), el cual en la imagen es representado con la " b ", aunque también se ve presentado en la literatura como w_0 . A esta parte del proceso se le llama suma de pesos y reduce la dimensionalidad de nuestro vector a un escalar único, el cual posteriormente será llevado a la función de activación, la cual tiene por objetivo dar no linealidad a nuestros datos y darnos una salida (output).

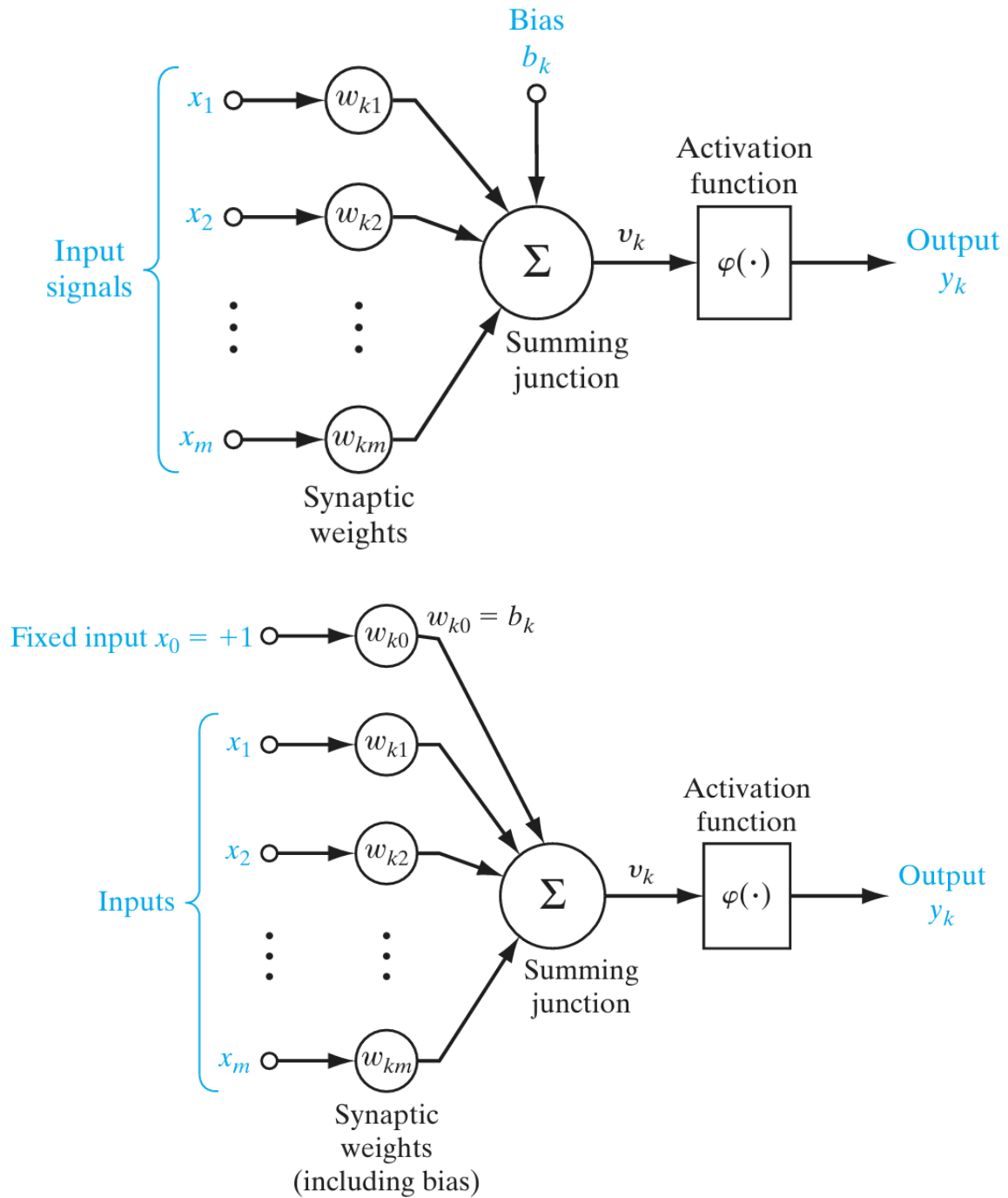


Figura 2: Imágenes de representación del perceptrón. Se observa la arquitectura que tiene una neurona artificial así como los componentes de esta. pesos (w), el sesgo se representa como w_{k0} , función de activación. Tomado de (Haykin, n.d., pp. 41-43)

1.1.1 Función de activación y su importancia

La función de activación ejerce como umbral permitiendo decidir si la señal de la neurona se transmite no basada en la entrada que se le asigna. Esta forma de asignar no linealidad no parece tener ningún sentido desde el punto de vista de una neurona, pero al apilarse en una red neuronal, nos permiten desbloquear la habilidad de aprender y modelar relaciones complejas de datos.

De manera algebraica se puede entender como la capacidad de las neuronas de poder modelar datos que tengan funciones con exponentes mayor a uno. Lo cual nos permite modelar ecuaciones más complejas como lo son las cuadráticas, cúbicas, etc.

De manera gráfica se puede visualizar como un cuadrante del plano cartesiano con datos linealmente separables, cuya división puede trazarse con una línea recta. Por otro lado, cuando nuestras clases son complejas y los datos pueden verse como en la imagen de la derecha, donde los datos no pueden ser delimitados con una línea recta. Para ello, la función de activación nos permite modelar este tipo de datos. De esta manera se dota a las redes neuronales de la capacidad de modelar funciones complejas o con varias dimensiones.

Es de hecho esta capacidad de modelar datos no lineales lo que le permite a áreas como el ML el asumir que las clases de datos son linealmente independientes en cualquier caso. Lo que en la actualidad ha sido aprovechada con algoritmos de ML como Support Vector Machine.

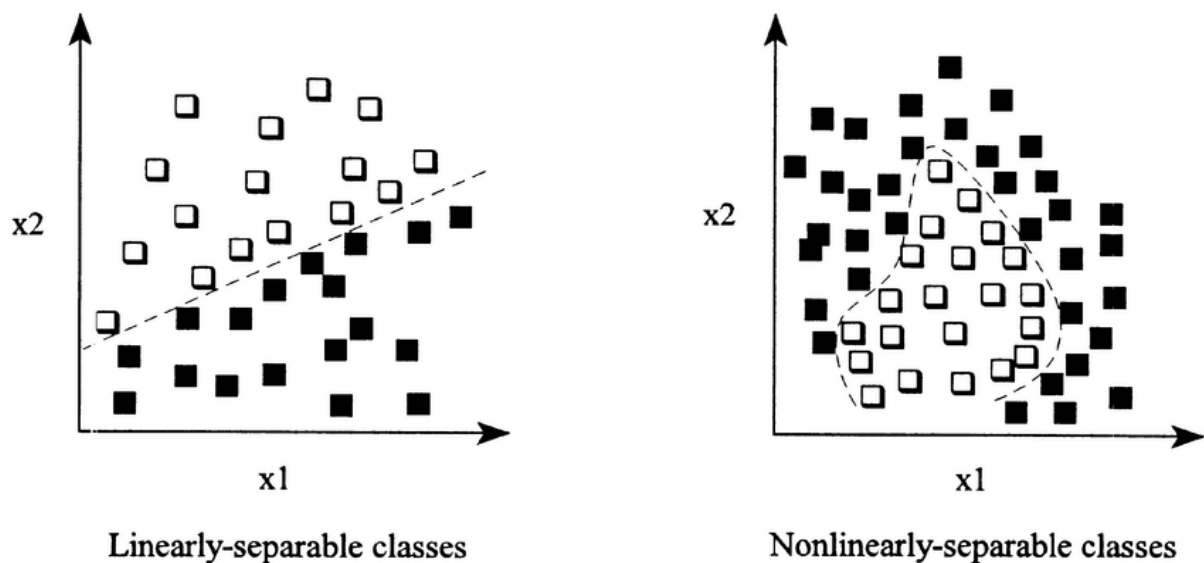


Figura 3: Representación de clases linealmente separables (lado izquierdo) y clases que no son linealmente separables (lado derecho). Tomado de (Basheer, Imad & Hajmeer, M.N. 2001. Artificial Neural Networks: Fundamentals, Computing, Design, and Application. Journal of Microbiological Methods. 43. 3-31. 10.1016/S0167-7012(00)00201-3.)

1.2 Segunda ola de la Inteligencia Artificial

Después en las décadas de los 80's - 90's la rama de la IA se le pasó a conocer como connectionism (conexionismo), el cual fue un movimiento en las ciencias cognitivas, la cual trataba de explicar las habilidades intelectuales usando Artificial Neural Networks (ARN).

Los conexionistas hicieron grandes avances (Buckner & Garson, 1997) demostrando el poder de las redes neuronales artificiales (ANN por sus siglas en inglés) para lograr tareas cognitivas. Algunos de los esfuerzos más relevantes fueron en el trabajo de Sejnowski y Rosenberg en 1987 cuando desarrollaron una red que podía leer inglés llamada NETtalk.

Otro modelo de la época del connectionism bastante importante fue en 1986 con la red para predecir el pasado de los verbos en inglés entrenada por Rumelhart y McClelland. La parte más relevante de este trabajo fue que los verbos más usados en inglés son verbos irregulares ("is" / "was", "come" / "came", "go" / "went") a pesar de que la gran mayoría de verbos son regulares teniendo la terminación "ed". La red

primeramente fue entrenada con verbos irregulares y después con un grupo de datos de 460 verbos de mayoría regular. La red aprendió el pasado de estos verbos en aproximadamente 200 rounds de entrenamiento.

Sin embargo, Pinker y Price (1988) señalaron unos años más tarde que el modelo hacía un trabajo pobre en generalizar algunos verbos más modernos, lo cual se vio como una señal de error básico en los modelos conexionistas. Donde quizá son buenos para hacer asociaciones y encontrar patrones, pero tuvieron limitaciones importantes para dominar reglas generales.

A pesar de que el debate continuó por años, los modelos conexionistas supusieron un nuevo paradigma para entender cómo la información puede ser representada por el cerebro. Aunque la evidencia no respalda que sea la manera en la que actúan los cerebros biológicos .

1.3 Tercera ola de la Inteligencia Artificial

A partir de mediados de los 2000 dentro del campo de la IA nace una nueva rama, la cual se denomina *Deep Learning* (DL). La cual ha sido la tendencia que se ha mantenido hasta la actualidad con el desarrollo de algoritmos de redes profundas tales como las *Convolutional Neural Network* (CNN's) o Redes Convolucionales, *Recurrent Neural Network* (RNNs) o Redes Recurrentes, *Reinforcement Learning Networks* o Redes de Aprendizaje por Refuerzo, por mencionar algunos.

El DL moderno va más allá de copiar el funcionamiento cerebral de algún ser vivo, puesto que distintos descubrimientos en distintas áreas como la neurociencia han encontrado hallazgos que han cambiado la manera en la que entendemos la inteligencia y el aprendizaje.

Por ejemplo:

En el 2000 en un artículo publicado por Von Melchner et al . Se descubrió que los hurones pueden aprender a “ver” usando la región cerebral auditiva si sus cerebros son recableados para enviar las señales visuales a esa área (Goodfellow et al., 2016). Aquel descubrimiento replantea la manera en la que se abordaba el DL,

puesto que parece ser que el cerebro, a pesar de tener áreas específicas, puede usar un mismo algoritmo para aprender tareas distintas.

Esto ha llevado a una interacción más cercana entre las distintas áreas del DL buscando mejores algoritmos que solucionan de manera holística distintos procesos. El DL moderno, aunque ha hecho avances significativos, aún tiene distintos problemas que solucionar para llegar a resolver problemas de manera eficiente en distintas áreas como lo hace el cerebro de los seres vivos. Esto ha dado apertura a la especialización del campo de la IA con nuevas subramas tales como la Generative AI (IA generativa).

Capítulo 2: Marco Teórico

2.1 Redes Neuronales Convolucionales

Las Convolutional Neural Networks (CNN's) son una arquitectura de red neuronal perteneciente al campo del DL donde cada una de las capas de la red CNN aplica una tarea específica a manera de filtro para poder abstraer características de imágenes y de esta manera poder entrenar modelos que puedan tomar decisiones en base a información visual obtenida por cámaras o sensores de video (Lecun et al., 2015, pp.436-444).

Las CNN 's son el estándar hoy en día para áreas de visión computacional y procesamiento de imágenes y son ampliamente usadas en el ámbito científico y empresarial. Tienen una arquitectura basada en las siguientes características:

- Capa de entrada
- La capa convolucional (Convolutional Layer)
- La capa de agrupación (Pooling Layer)
- La capa de aplazamiento (Flatten Layer)
- La capa totalmente conectada (Fully-Connected Layer)
- Capa de salida

2.1.1 Capa convolucional (Convolutional layer)

Esta capa consiste en un conjunto de filtros que pueden ser aprendidos para comenzar a extraer información de las imágenes. Por ejemplo, un filtro (kernel) de 7x7x3 donde los primeros dos parámetros se refieren al tamaño del kernel en las dimensiones ancho y alto, mientras que el último parámetro (3) hace referencia a la profundidad.

La profundidad de una imagen se determina por los canales que la componen, así que si se está usando una imagen con el espacio de color RGB, podemos hablar de un filtro de 3 en la dimensión de profundidad (Red, Blue, Green). Así mismo aplica

para los espacios de colores como HSV (Hue, Saturation, Value). Sin embargo, en espacios de color con otras composiciones como CMYK (Cian, Magenta, Yellow, Key) habría 4 capas de profundidad. Tal que el filtro sería 7x7x4.

Esto es particularmente importante ya que la manera en la que una red neuronal sea diseñada será influenciada por los instrumentos y los formatos bajo los que sea operada.

2.1.2 Capa de agrupación (Pooling layer)

Las imágenes en el fondo son matrices conformadas por píxeles, los cuales son representados por valores numéricos de sus espacios de colores. De esta manera una imagen RGB tiene 3 valores numéricos que van del 0 al 255 para representar los colores que observamos.

Computacionalmente hablando, tiene una dimensionalidad alta debido a la cantidad de valores numéricos con los que operamos. Para evitar la alta dimensionalidad y debido a que es impráctico conectar todas las neuronas con todas las del volumen anterior, se conecta cada neurona únicamente a una región local del volumen total.

2.1.3 Capa de aplanamiento (Flatten layer)

La capa de aplanamiento lo que hace es convertir la información de las capas de convolución y pooling en un vector y después se conecta a la capa totalmente conectada.

2.1.4 Capa totalmente conectada (Fully connected layer)

La capa totalmente conectada simplemente es una capa que se conecta a todos los valores de la capa anterior. Entre la capa de aplanamiento y la capa totalmente conectada buscan reducir la dimensionalidad de la información para poder tomar una decisión de la clasificación o el valor al que se asemeja la información obtenida por las capas anteriores. Esta capa puede después conectarse con una función softmax cuya función es la de convertir nuestros valores en una distribución de

probabilidad para poder reducir la clasificación a un valor porcentual (%) y de esta manera dar una respuesta.

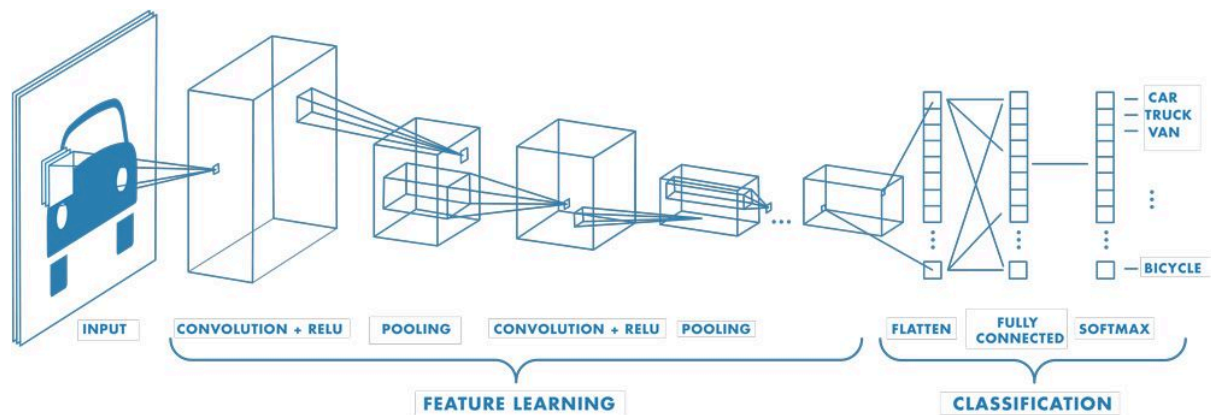


Figura 4: Arquitectura de una red neuronal convolucional (CNN). Tomado de (*What Is a Convolutional Neural Network?* - MATLAB & Simulink, <https://www.mathworks.com/discovery/convolutional-neural-network.html>.)

2.2 Vision Transformer

Los *Vision Transformers* (ViT) son una arquitectura de *Deep Learning* propuesta originalmente para extender el uso de la arquitectura *Transformer*, la cual había probado ser exitosa en el campo del Procesamiento del Lenguaje Natural (NLP) al campo de la visión computacional. Esta arquitectura fue introducida por Alexey Dosovitskiy en el paper “*An image is worth 16X16 words: Transformers for image recognition at scale*” (Dosovitskiy et al., 2021).

Los ViT modelan las imágenes como secuencias de parches (patches) y utilizan mecanismos de *self-attention* para capturar relaciones globales dentro de la imagen. Esto contrasta con las CNN, las cuales dependen de las operaciones convolucionales locales para extraer características del espacio. La motivación detrás de la arquitectura ViT es debido a que las CNN's tienden a imponer sesgos inductivos como:

- Localidad espacial (sujeta al tamaño de los kernels)
- Invariancia a traslaciones
- Jerarquía espacial implícita.

Esto aunque no explícitamente perjudicial si impiden al modelo el capturar relaciones globales de un alcance mayor, especialmente en imágenes donde la información no se encuentra en regiones contiguas.

Los transformers no operan sobre los píxeles individuales ni utilizan kernels convolucionales como lo hacen las CNNs. En lugar de ello, una imagen puede descomponerse por sus características en:

- *Height (H)*
- *Width (W)*
- *Channels (C)*

Tal que una imagen se representa como: $H \times W \times C$.

De esta manera los ViT se dividen en parches no superpuestos de tamaño $P \times P$. Luego de ello, cada parche es aplanado y proyectado (embebido) linealmente en un espacio de mayor dimensión.

A este proceso se le conoce como (*embedding*). Siendo una imagen representada como una secuencia de vectores. Y de esta forma y por su proceso asimilando al proceso de tokenización en NLP, pero aplicado a las imágenes.

2.2.1 Embedding posicional

Para conservar el orden de la información de la imagen, se agrega un *embedding* posicional a cada parche para conservar la información espacial. De esta manera un modelo puede distinguir entre diferentes ubicaciones dentro de un plano visual.

2.2.2 Encoder Transformer

Una vez generada la secuencia de parches con *embeddings* posicionales, estos son procesados por un *Transformer Encoder*, el cual está compuesto por múltiples bloques idénticos que contienen:

- Mecanismo de autoatención (*Multi-Head Self-Attention*): Similar al *Transformer* usado en NLP, el mecanismo de *self-attention* permite a cada parche de la imagen ponderar la importancia relativa contra los demás para capturar dependencias globales, relaciones espaciales no locales y contexto visual completo.
- *Red Feed Forward* (MLP): Una vez pasado el mecanismo de *self-attention*, cada token se pasa por una red neuronal (MLP) totalmente conectada para introducir no linealidad y pulir las representaciones aprendidas.
- Normalización: Se normalizan los resultados para estabilizar el entrenamiento y facilitar la propagación del gradiente.

Por medio de este proceso las redes ViT generan una salida común, la cual puede ser usada para clasificación o regresión.

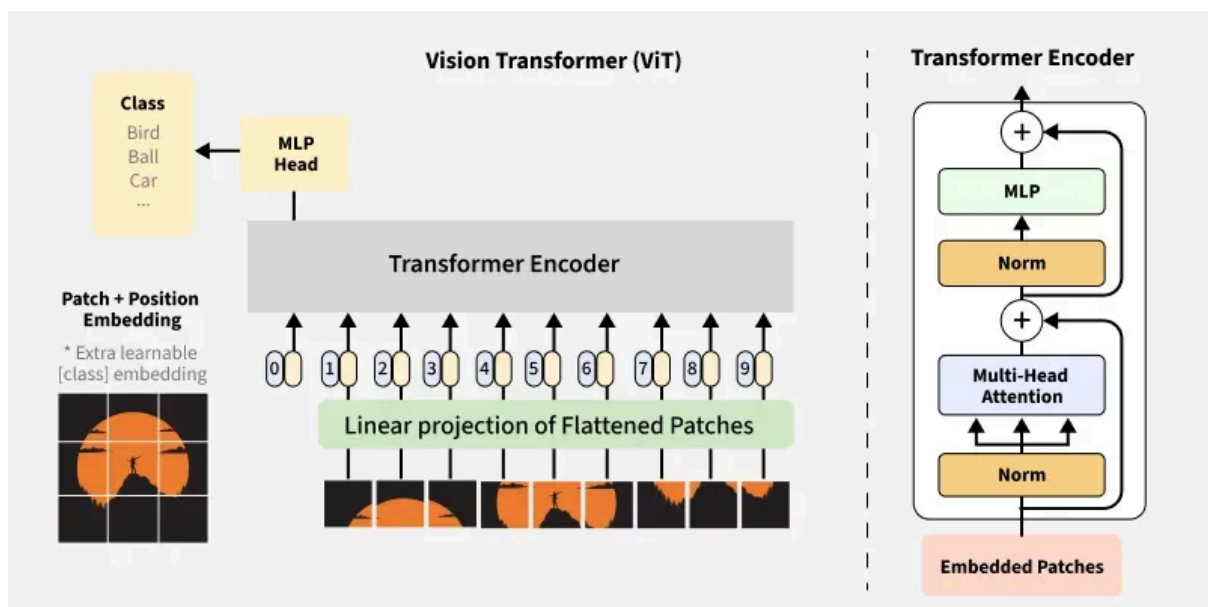


Figura 5: Arquitectura de una red tipo visual transformer (ViT). Tomado de (<https://www.geeksforgeeks.org/deep-learning/vision-transformer-vit-architecture>).

2.3 ¿Qué es la conducción autónoma?

La conducción autónoma se refiere a vehículos que pueden operar sin la intervención de un humano, utilizando inteligencia artificial para controlar los

movimientos y tomar decisiones basadas en datos recolectados en tiempo real por medio de sensores e información geográfica. (“AI-generated definition based on: Innovations in Artificial Intelligence and Human-Computer Interaction in the Digital Era, 2023, Sciencist Direct:

<https://www.sciencedirect.com/topics/computer-science/autonomous-driving#definition>”)

También conocido como *self-driving cars* o *automated driving cars*. Son términos que se usan de manera intercambiada para definir el concepto de vehículos que se conducen de manera autónoma (sin la intervención de personas).

En la enciclopedia Britannica (Cole, R. 2025) se definen seis fases de automatización para los automóviles según sus características.

2.3.1 Niveles de conducción autónoma

Los lineamientos establecidos en la J 3016-2018 por la *Society of Automotive Engineers* son los que dan guía a los niveles de conducción autónoma. En cuanto a sistemas vehiculares, se hace una diferencia entre las tecnologías de asistencia en la conducción y las tecnologías de automatización de la conducción de la siguiente manera:

- Tecnologías de asistencia en la conducción: Es la tecnología de asistencia a los humanos en el proceso de conducción, aún requiere al conductor hacer todas las decisiones y operar el vehículo (niveles 1 y 2)
- Tecnologías de automatización de la conducción: Tecnología de inteligencia artificial para que el vehículo intervenga, tome decisiones y controle operaciones del vehículo (niveles 3-5)

Los niveles fueron establecidos en función del grado de automatización que existe en el vehículo para la conducción y determinan el nivel de inteligencia de un vehículo evaluando sus capacidades en un rango de 0-5 donde 0 representa nada de automatización y 5 representa automatización total. Estos estándares fueron

establecidos con la finalidad de mantener la consistencia entre la ingeniería, los medios de comunicación y las áreas legales.

Para esta clasificación se hicieron clasificaciones de 3 elementos que participan en la conducción:

- El conductor
- El sistema de automatización
- Otros sistemas de soporte

Los niveles fueron establecidos en función de la labor empleada en cada uno de los elementos durante la conducción.

2.3.1.1 Nivel 0: Sin automatización en la conducción

Aquí no existe ninguna autonomía. Es el auto con el que la gran mayoría de las personas ha consumido en el cual depende 100% de las decisiones humanas, siendo igualmente responsable.



Figura 5: Vehículo tradicional sin automatización. Tomado de (<https://www.google.com>).

2.3.1.2 Nivel 1: Asistencia de conducción.

Aquí el vehículo tiene algún sistema de ayuda durante la conducción, como sistemas de velocidad adaptativos. Aquí el conductor sigue siendo el responsable de las decisiones y el control del vehículo en general. Por ejemplo, la imagen de la

figura 6 muestra el Honda Civic 2018 con crucero adaptativo el cual puede mantener una velocidad constante o ajustar automáticamente la marcha para conservar una distancia segura con el vehículo delantero. También, puede frenar, detenerse y reanudar la marcha en el tráfico.



Figura 6: Vehículo 2018 Honda Civic con control crucero adaptativo (adaptive cruise control). Tomado de (<https://www.google.com>).

2.3.1.3 Nivel 2: Automatización parcial.

El vehículo puede combinar 2 o más tareas automatizadas, como dirigir y acelerar a la vez. Sin embargo, el conductor debe estar alerta todo el tiempo para intervenir en caso de ser necesario. Por ejemplo, el Tesla Model S 2018 cuenta con hardware Autopilot 2.5, que utiliza cámaras, radares para mantenerse en el carril o cambiar con supervisión del piloto.



Figura 7: Vehículo Tesla Model S 2018 con sistema de piloto automático. Tomado de (<https://www.google.com>).

2.3.1.4 Nivel 3: Automatización Condicional.

En este nivel los vehículos tienen una automatización limitada bajo ciertas condiciones. El vehículo es capaz de tomar ciertas decisiones usando sensores como LiDAR. El conductor debe estar ahí en caso de que el vehículo no vaya como se espera, aunque el vehículo es capaz de tomar el control bajo condiciones específicas durante una ruta. Un ejemplo es el Audi A8 2019 con AI y conducción híbrida donde el vehículo puede tomar el control total de la dirección, aceleración y frenado. El conductor puede soltar el volante y realizar otras actividades, aunque debe estar listo para retomar el control si el sistema lo solicita.



Figura 8: Vehículo Audi A8 2019 con AI y conducción híbrida. Tomado de (<https://www.google.com>).

2.3.1.5 Nivel 4: Alta automatización.

Los vehículos tienen tecnología para automatización total en la mayoría de las condiciones. Bajo ciertas circunstancias, la supervisión del conductor humano es necesaria, tal como conducir donde hay poca visibilidad. Estos se conducen gracias a sistemas de geolocalización, radares, cámaras y detectores. Sin embargo, todavía se puede regular si es necesario. Estos vehículos aún se encuentran en desarrollo y son empresas como Google con Waymo o Tesla quienes más han buscado avances en este campo.



Figura 9: Vehículo Waymo proyecto Google self-driving car.

Tomado de (<https://waymo.com/blog/2016/12/say-hello-to-waymo-whats-next-for>).

2.3.1.6 Nivel 5: Total automatización.

Estos vehículos tienen tecnología para automatización total donde no es requerida la intervención de los humanos. Los vehículos son capaces de operar bajo cualquier condición. Debido a que estos vehículos no requieren intervención humana, no necesitan un volante o pedales, por lo que es un modelo totalmente distinto.

Actualmente no hay vehículos de nivel 5 ya que aún no se han encontrado métodos para que los vehículos puedan operar en condiciones adversas.

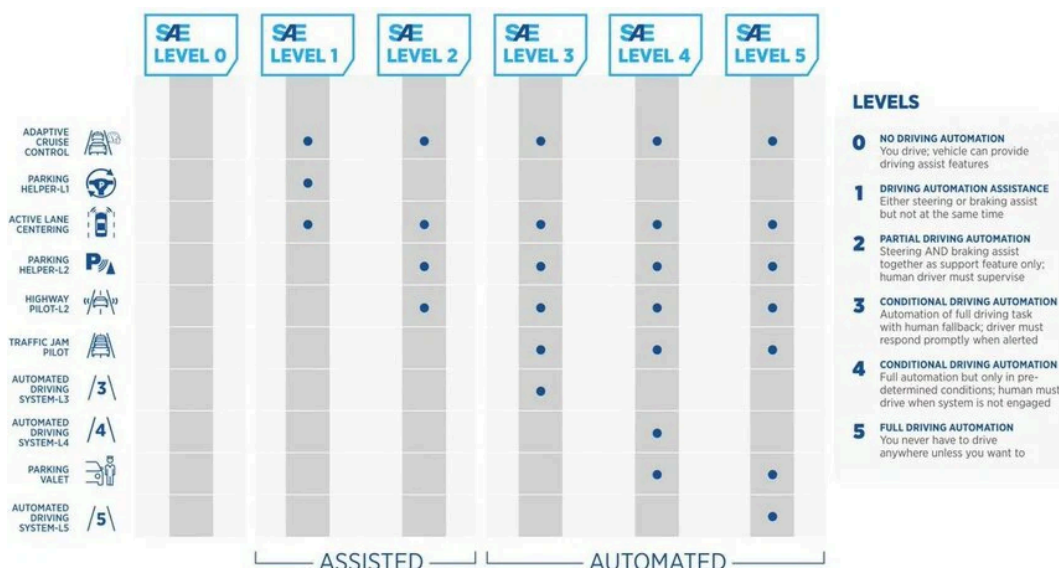


Figura 10: Explicación de los 6 niveles de automatización en conducción. Tomada de: (<https://blog.rgbsi.com/6-levels-of-driving-automation>, 2025).

2.3.2 LiDAR

El *Light Detection and Ranging* (LiDAR) es una tecnología que emite luz láser pulsada hacia el entorno, la cual rebota en los objetos del ambiente alrededor y vuelve al sensor LiDAR (IBM, 2025). El sensor así mide el tiempo que tarda en volver cada pulso y con esa información calcula la distancia a la que se encuentra un objeto. Esto es posible gracias a que el láser tiene una velocidad constante que es la de la luz, siendo de esta manera muy fácil de calcular la distancia por medio de una regla de tres.

Para modelar la información obtenida, los datos se someten a varias etapas de procesamiento para transformar la información recibida por el LiDAR en un mapa 3D. Primero, se comprueba que sean correctos y completos y se limpia para eliminar ruidos anómalos. A continuación, se pueden identificar y clasificar algorítmicamente las características de la superficie del terreno, como edificios, bancos y marquesinas. Para simplificar el cálculo, se reduce la resolución para eliminar datos redundantes y reducir el tamaño de la información. Finalmente, los datos se convierten al formato de archivo LAS (o LASer) estándar del sector que se utiliza para intercambiar datos 3D x, y, z.

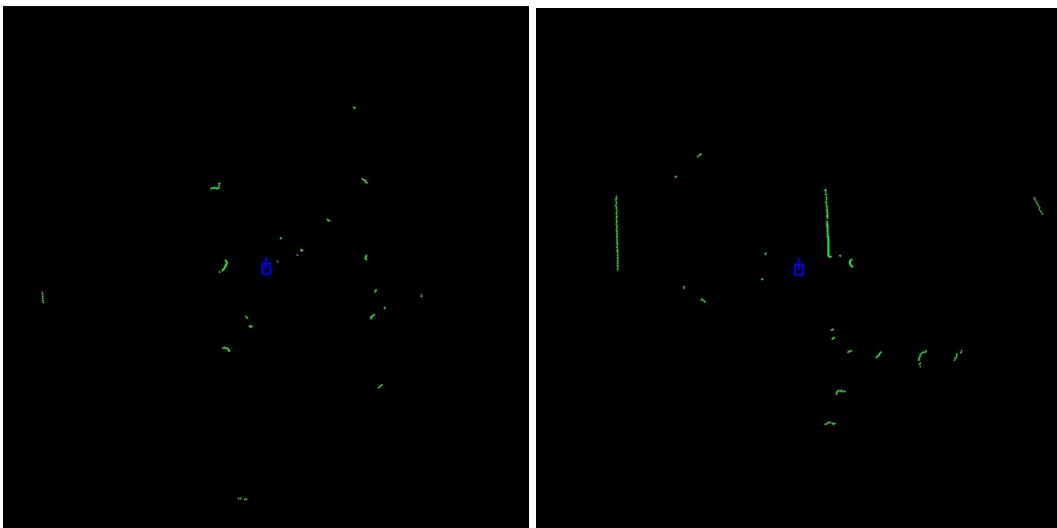


Figura 11: Mapa generado con los datos de LiDAR. Imágenes recolectadas con el vehículo tipo Akerman del laboratorio del lenguaje y el conocimiento (LKE) de la Benemérita Universidad Autónoma de Puebla.

2.3.3 ¿Qué es el preprocesamiento de imágenes?

El preprocesamiento es el conjunto de operaciones realizadas con imágenes al más bajo nivel de abstracción. El objetivo del preprocesamiento es mejorar la calidad de los datos de las imágenes para su futuro consumo por un software (Sonka et al., n.d., pp. 56–111). Otra definición es:

“El preprocesamiento de imágenes son una serie de técnicas empleadas para mejorar la calidad y extraer información relevante de las imágenes digitales antes que sean analizadas y procesadas por un algoritmo de aprendizaje de visión computacional.(Microblink).”

Así el preprocesamiento de imágenes consiste en modificar a nivel de la estructura de la imagen misma características que quizá no fueron bien capturadas durante su toma debido a problemas con el hardware o las condiciones, con la finalidad de mejorar las imágenes y que a la hora de ser utilizadas en un programa computacional este pueda aprovechar sus características de la mejor manera para tomar mejores decisiones.

En el área de IA la calidad de los datos es de suma relevancia en muchas áreas como la medicina se ha demostrado que la calidad de las características de las imágenes influye de manera significativa en el entrenamiento de una *Artificial Neural Network* (ANN) y el cómo responde en base a distintos datasets con conjuntos de datos de diferentes calidades.

Como se menciona en *Impact of quality, type and volume of data used by deep learning models in the analysis of medical images* (Luca et al., 2022), *Preprocessing Technique - an overview* (Science Direct, n.d.) y *Impact of Data Quality on CNN-Based Sewer Defect Detection* (Jang, 2025), la calidad de los datos utilizados en sus entrenamientos afecta de manera directa la calidad de las respuestas de los modelos. Por ello, a continuación se explican algunas de las técnicas de preprocesamiento de imágenes.

2.3.4 Técnicas de preprocesamiento de imágenes.

Las principales técnicas de preprocesamiento de imágenes son requeridas previamente al análisis principal de los datos y la extracción de información. Estas modificaciones son generalmente agrupadas como correcciones radiométricas y geométricas.

Las correcciones radiométricas son aquellas que modifican los niveles digitales (ND por sus siglas en inglés) de la imagen, con el propósito de hacer que estos emitan un caso ideal de captura de imagen en condiciones perfectas. Estos incluyen problemas causados por el funcionamiento de los sensores.

Por otro lado, las correcciones geométricas son correcciones que se realizan principalmente en imágenes tomadas por satélites para corregir la geometría de la imagen, ya que estas están distorsionadas por la distancia a la que se toman.

2.3.4.1 Técnicas de corrección radiométrica.

Estas se dividen en dos métodos: métodos de dominio espacial y métodos de dominio de frecuencia

2.3.4.1.1 Métodos de dominio espacial

Estos métodos operan directamente en los píxeles de la imagen. Esto significa que el valor de un píxel con coordenadas (x,y) de la imagen T es el resultado de realizar alguna operación en los píxeles vecinos de (x,y) en la imagen ingresada F .

Algunas de las manipulaciones a nivel de dominio espacial son: transformación de espacio de color, ecualización del histograma, formulación discreta, suavizado de imágenes y realce de imagen.

2.3.4.1.1.1 Transformación de espacio de color

Un modelo o espacio de color es la especificación de un sistema de coordenadas tridimensionales y de un subespacio de este sistema en el que cada color queda representado por un único punto. (Molinero Diez, n.d.)

La manera en la que se define un color depende como lo define su espacio. Componiéndose por la abstracción de características que componen el color mismo. Esto nos permite normalizar un color y poder definirlo dentro de un conjunto de características. Aunque existen muchos espacios de color, los más utilizados para el procesamiento de imágenes son : RGB, YIQ, CMY, YCbCr y HSI.

2.3.4.1.1.2 Características del color

Las características principales de los colores son: brillo, tono y saturación.

El brillo es el umbral de 0% (negro total) - 100% (blanco) de luminosidad o penumbra que existe.

El tono es el valor reflejado o transmitido a través de los objetos. Este, a diferencia de los anteriores, se mide como un ángulo en grados, por lo que su rango va de 0% a 360%. Este es el que es comúnmente definido como el color en sí mismo, ya que identifica los tonos como rojo, naranja, amarillo, azul, etc.

La saturación o cromatismo se refiere a la pureza de la cantidad de luz blanca mezclada con el tono, es decir, es la fuerza o pureza del color. Esta es representada como la cantidad de blanco que existe con respecto al tono y se mide de igual manera con un umbral de 0% (gris) - 100% (máxima saturación).

☐ RGB

En este modelo del color sus componentes espectrales Red, Blue, Green son los que encierran todos los colores. Este modelo está basado en el sistema de coordenadas cartesianas de 3 dimensiones donde cada color representa una dimensión en un rango de 0 a 255 y

es mediante la representación de las coordenadas (x,y,z) como se puede obtener una gama de colores. Por ejemplo:

0/0/0 Representa el origen del plano y representa el color negro. Por otro lado, la coordenada 255/255/255 representa el color blanco siendo la diagonal del mapa cartesiano.

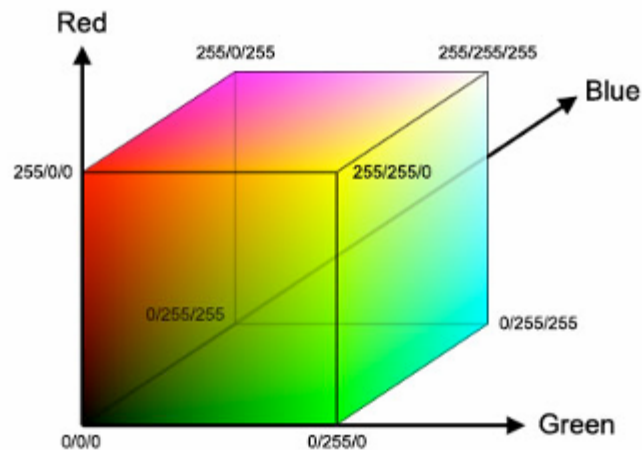


Figura 12: Espacio de color RGB una representación gráfica. Tomado de: (<https://biblus.us.es/bibing/proyectos/abreproy/11875/fichero/Proyecto+Fin+de+Carrera%252F3.Espacios+de+color.pdf>)

☐ CMY

Representados por los valores Cian Magenta y Amarillo (Y por su nombre en inglés) son los colores primarios de los pigmentos y secundarios de la luz. Estos colores son denominados como sustractivos ya que se utilizan como filtros para sustraer colores de la luz blanca. En cuanto a cómo se representa, es igual en el sistema coordenado que el RGB, pero con una inversión del negro y el blanco.

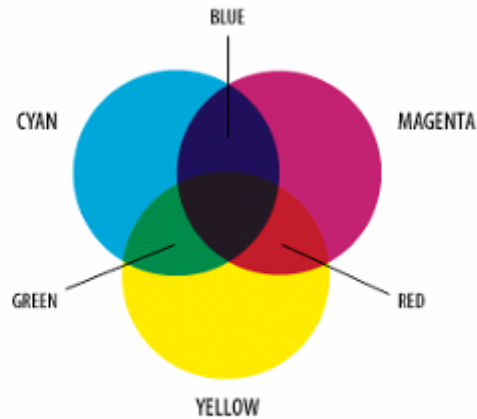


Figura 13: Espacio de color CMY una representación gráfica. Tomado de: (<https://biblus.us.es/bibing/proyectos/abreproy/11875/fichero/Proyecto+Fin+de+Carrera%252F3.Espacios+de+color.pdf>)

□ YCbCr

Este espacio de color es una codificación no lineal del espacio RGB donde Y es la luminancia, Cb y Cr indican el tono del color: donde Cb ubica el color en una escala entre azul y amarillo y Cr ubica entre el rojo y el verde. Para transformar un espacio de RGB a YCbCr se puede hacer uso de la siguiente transformación matricial.

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{pmatrix} 65.481 & 128.553 & 24.966 \\ -39.797 & -74.203 & 112 \\ 112 & -93.786 & -18.214 \end{pmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

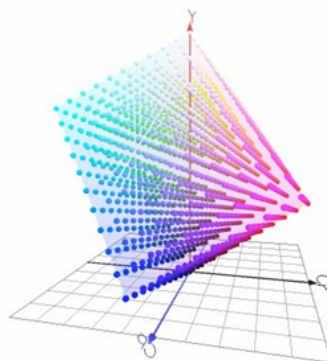


Figura 14: Espacio de color YCbCr una representación gráfica. Tomado de: (<https://biblus.us.es/bibing/proyectos/abreproy/11875/fichero/Proyecto+Fin+de+Carrera%252F3.Espacios+de+color.pdf>)

□ HSI

El espacio de color HSV (Hue, Saturation, Value) es un modelo que representa los colores de manera muy similar a como lo percibimos los seres humanos. HSV separa los colores en tres componentes principales:

- **Hue (Tono):** Representa el color base.
- **Saturation (Saturación):** Representa la intensidad del color
- **Intensity (Intensidad):** Representa el brillo

El espacio HSV es de los más usados en el procesamiento de imágenes debido a que separa la información (i) de la información cromática contenida en la imagen. Además los canales H y S se encuentran relacionados con la manera en la que los seres humanos percibimos los colores. Haciendo que HSI sea el espacio de color ideal para el desarrollo de algoritmos para el procesamiento de imágenes.

En la figura 16 podemos ver una representación del espacio HSV, en la imagen de la izquierda podemos ver que a diferencia de otros espacios como el RGB el espacio HSV se representa como un cono de colores donde el color se representa como un ángulo entre 0° y 360° . En el eje vertical que representa el valor (V) mide el brillo con el que se percibe un color, llendo desde el color negro hasta el máximo brillo del color. La saturación (S) nos da el valor de la intensidad del color, yendo desde el gris como baja saturación hasta un color de alta saturación en el eje más externo.

Como se mencionó previamente separar la información de esta manera tiene beneficios a la hora de trabajar con imágenes, ya que con estos componentes, se pueden extraer las características inherentes del color de manera que se pueden manipular individualmente para mejores resultados, ya sea trabajando con el

color, la saturación o el brillo. Pero su mayor importancia es que nos permite normalizar la escala de grises de manera más fiel sin perder calidad o detalles dentro de las imágenes.

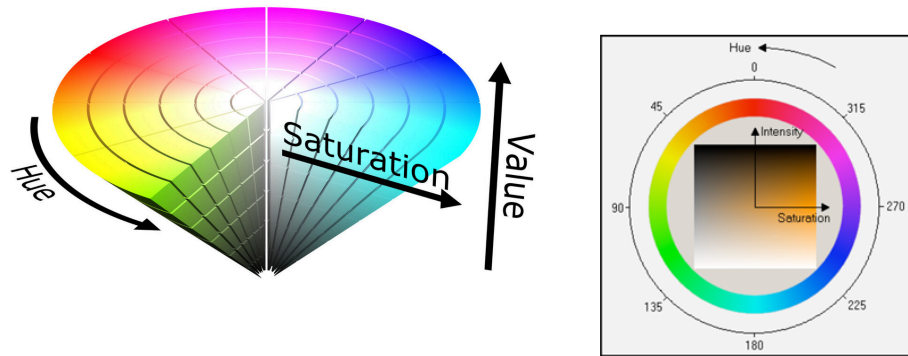


Figura 15: Espacio de color HSI, una representación gráfica. Tomado de: (<https://biblus.us.es/bibing/proyectos/abreproy/11875/fichero/Proyecto+Fin+de+Carrera%252F3.Espacios+de+color.pdf>)

2.3.4.1.1.3 Ecualización del histograma

La ecualización del histograma es una técnica común para mejorar el contraste de una imagen, al redistribuir los valores de intensidad que ocurren con mayor frecuencia. Esta herramienta es especialmente útil cuando las imágenes tienen un contraste puro debido al bajo nivel de contraste dinámico en los valores de los píxeles.

La ecualización del histograma busca una transformación de escala de grises que correa una imagen con un histograma uniforme o cercano a uniforme.

Contraste dinámico: es una medida que evalúa la variación de un píxel de negro a blanco (o entre grises) en un período de tiempo (Visual LED, n.d.).

2.3.4.1.1.4 Suavizado de imagen

El principal objetivo del suavizado de imagen (image smoothing) es reducir los efectos de ruido por la cámara, valores atípicos, pérdida de valores de píxeles, etc.

Las imágenes se pueden filtrar con filtros de tipo low-pass (LPF), high-pass (HPF), etc. Los LPF ayudan a reducir el ruido mientras que los HPF ayudan a encontrar bordes en las imágenes.

Algunas técnicas de LPF son:

- ☐ Promediando: Toma el promedio de todos los píxeles en un kernel (matriz de $N \times N$ donde N es impar) y reemplaza el valor central del kernel

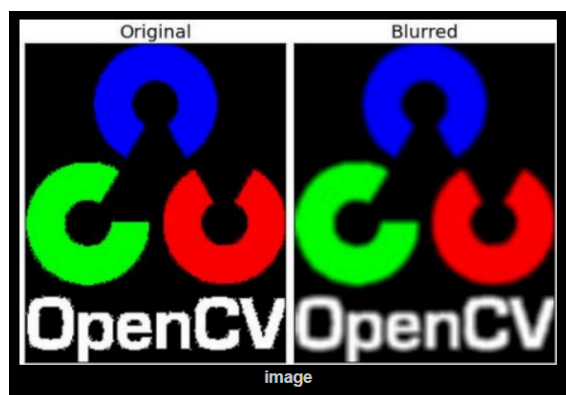


Figura 16: Representación de Kernel OpenCV, imagen original (izquierda), imagen difuminada (derecha). Tomado de: (https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html).

- ☐ Suavizado Gaussiano: Este es muy útil para remover el ruido gaussiano, el cual tiene una distribución normal (campana de Gauss).

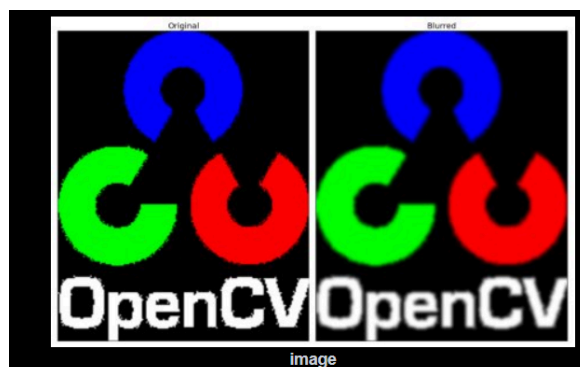


Figura 17: Suavizado gaussiano OpenCV, imagen original (izquierda), imagen gaussiana (derecha). Tomado de: (https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html).

- ☐ Suavizado de Media: Toma la media de los valores del kernel y reemplaza el valor al centro del kernel con esta media. Este filtro es altamente efectivo en imágenes que tienen ruido del tipo sal y pimienta (estática).

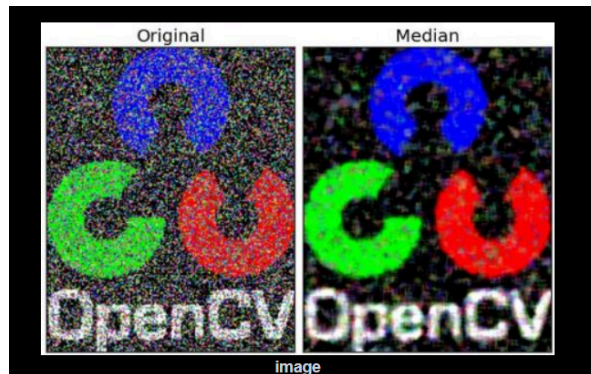


Figura 18: Suavizado de media OpenCV, imagen original (izquierda), imagen suavizada (derecha). Tomado de: (https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html)

- ☐ Filtro bilateral: Es un filtro altamente efectivo para remover ruido mientras mantiene un alto contraste con los bordes.

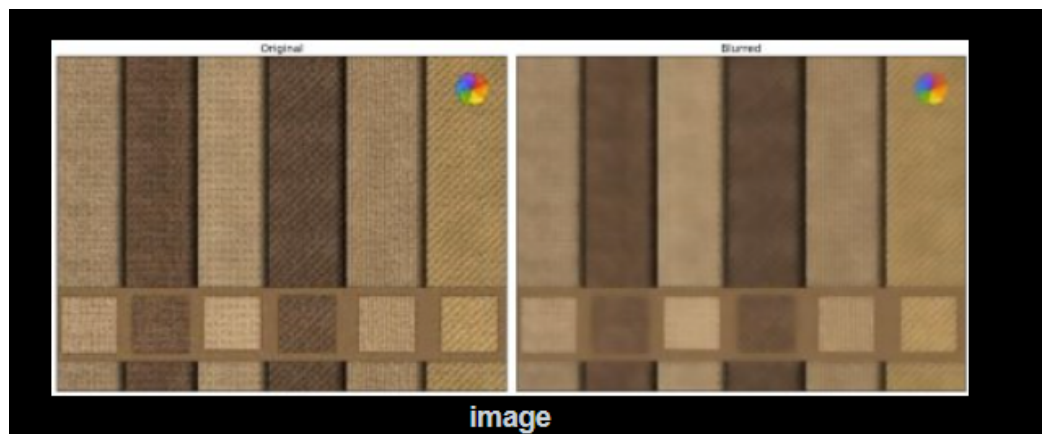


Figura 18: Filtro bilateral OpenCV, imagen original (izquierda), imagen suavizada (derecha). Tomado de: (https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html).

2.3.4.1.2 Métodos de dominio de frecuencia:

En los métodos de dominio de frecuencia, el procesamiento de la imagen no se hace directamente sobre los valores de píxel, sino sobre su representación en frecuencia, obtenida mediante una transformada de Fourier, generalmente usando

una Transformada Discreta de Fourier (DFT por sus siglas en ingles) o la Transformada Rapida de Fourier (FFT por sus siglas en ingles).

- Se transforma la imagen al dominio de frecuencia (por ejemplo, con FFT).
- Se modifica la representación de frecuencia aplicando un filtro o función de transferencia.
- Se vuelve a transformar al dominio espacial mediante la transformada inversa.

Este proceso aprovecha que la multiplicación en el dominio de frecuencia equivale a la convolución en el dominio espacial, lo que puede simplificar ciertas operaciones complejas (Universidad de Sevilla, n.d.).



Figura 19: Transformación en el dominio de la frecuencia. Tomado de: (<https://asignatura.us.es/imagendigital/Tema3-Frecuencia.pdf>).

2.3.5 Buscadores de bordes de convolución

Los algoritmos de detección de bordes tienen el objetivo de identificar los bordes más significativos dentro de una imagen. Estos bordes le permiten a las redes neuronales convolucionales el poder distinguir de mejor manera la composición de las imágenes y contener imágenes segmentadas de mejor función.

Los modelos de bordes son construidos para describir los diferentes tipos de bordes que pueden suceder en una imagen. Los modelos básicos son STEP RAMP y ROOF.

Un borde de tipo STEP representa un cambio abrupto en intensidad, cambiando de manera rápida de intensidad en un paso.



Figura 20: Representación del buscador de bordes de convolución, borde tipo STEP. Tomado de: (<https://blog.roboflow.com/edge-detection/>).

Un borde tipo RAMP describe una transición gradual en la intensidad dentro de cierta distancia.

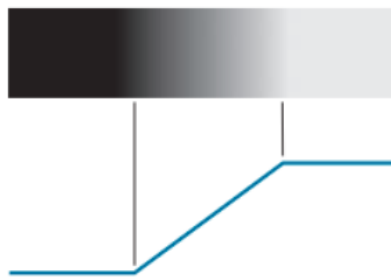


Figura 21: Representación del buscador de bordes de convolución, borde tipo RAMP. Tomado de: (<https://blog.roboflow.com/edge-detection/>).

Finalmente, un borde tipo ROOF representa un pico de intensidad donde va de un máximo y después decrece.

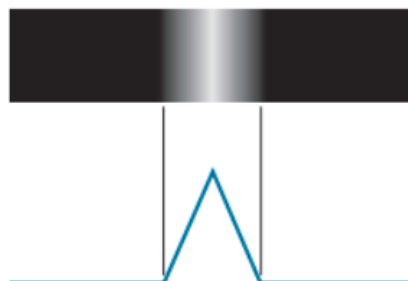


Figura 22: Representación del buscador de bordes de convolución, borde tipo ROOF. Tomado de: (<https://blog.roboflow.com/edge-detection/>).

Dentro de la gama de detectores de imágenes más relevantes tenemos:

- *Detección de bordes Sobel (Sobel Edge Detection)*

El detector de bordes Sobel es una técnica ampliamente utilizada en el procesamiento de imágenes y visión computacional para detectar bordes en una imagen. Es un método que se basa en una convolución de gradiente con kernels específicos para calcular la magnitud del gradiente y la dirección de cada píxel en la imagen. A continuación se explica cómo funciona este detector.

El operador Sobel utiliza dos kernels convolucionales de 3x3 (filtros). El primero es para detectar cambios en el eje x (bordes horizontales). El segundo kernel busca detectar cambios en el eje y (bordes verticales). De esta manera estos dos filtros son usados para computar el gradiente de la intensidad en cada punto de la imagen, lo que ayuda a detectar bordes. Estos son los kernels que utiliza el filtro Sobel:

□ Kernel Horizontal (G_x):

Este kernel es usado para detectar bordes horizontales al enfatizar el gradiente en el eje x.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Figura 23: Matriz del kernel horizontal Sobel. Tomado de: (<https://docs.opencv.org/>).

El kernel G_x enfatiza los cambios de intensidad en los píxeles en la dirección horizontal. Los valores positivos (+1 y +2) en el eje correcto serán remarcados resultando en áreas brillantes, mientras que los

valores negativos (-1 y -2) serán desmarcados dando áreas más oscuras de esta manera detectando bordes horizontales.

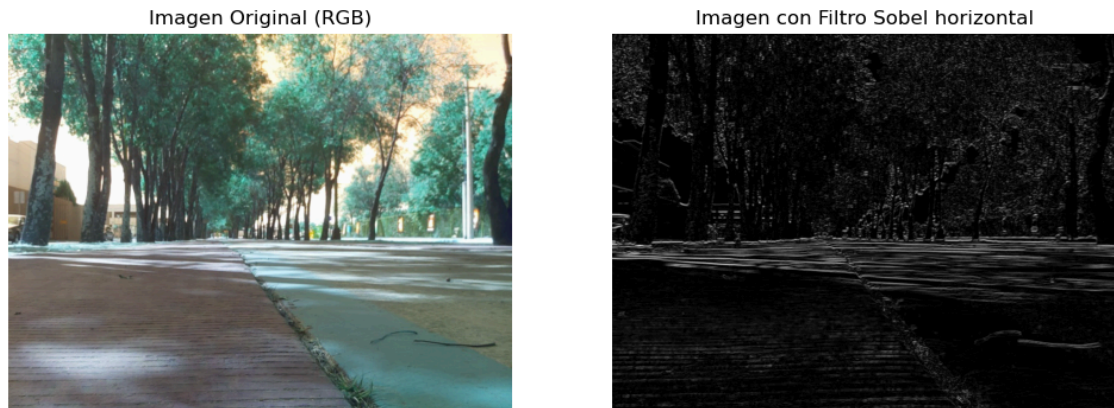


Figura 24: Aplicación del kernel horizontal Sobel.

□ Vertical Kernel (G_y):

Este kernel es usado para detectar bordes verticales al enfatizar el gradiente en el eje y.

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Figura 25: Matriz del kernel vertical Sobel. Tomado de: (<https://docs.opencv.org/>).

El kernel G_y enfatiza la intensidad en dirección vertical. De manera similar con los valores positivos (+1 and +2) al fondo serán marcados como áreas brillantes, mientras que los valores negativos (-1 and -2) arriba serán marcados como áreas negras, de esta manera detectando bordes verticales.

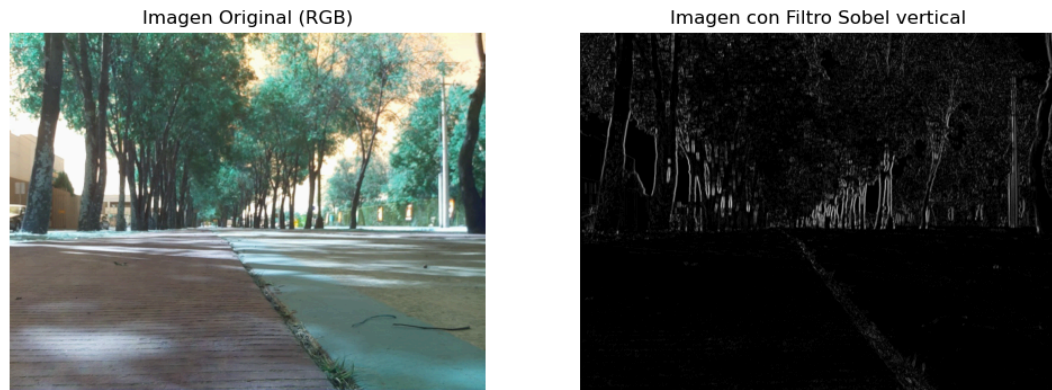


Figura 26: Aplicación del kernel vertical Sobel.

- *Detector de Bordes Canny (Canny Edge Detection)*

El detector de bordes canny es un algoritmo que detecta un amplio rango de bordes en las imágenes. Fue desarrollado por John F. Canny en 1986 y es conocido por su buen desempeño para detectar bordes. El algoritmo funciona siguiendo una serie de pasos para reducir ruido, detectar bordes y de esta manera mejorar la exactitud para detectar bordes. A continuación se explica cómo funciona el algoritmo Canny:

1. *Reducción de ruido con Gaussian Blurring:* El primer paso en el detector Canny es el alisamiento de la imagen por medio de un filtro Gaussiano. Esto ayuda a reducir el ruido y detalles no deseados en una imagen. El filtro Gaussiano se aplica a la imagen para aplicar una convolución y con ello obtener un resultado más liso.



Figura 27: Reducción de ruido Gaussiano. Tomado de: (https://es.wikipedia.org/wiki/Archivo:Halftone,_Gaussian_Blur.jpg).

2. *Cálculo del Gradiente:* Después de la reducción se aplica el operador Sobel para calcular la intensidad del gradiente y se calcula en las direcciones de los ejes x , y (G_x y G_y). De esta manera, la magnitud del gradiente es computada usando los siguientes gradientes.

$$Edge_Gradient (G) = \sqrt{G_x^2 + G_y^2}$$
$$Angle (\theta) = \tan^{-1} \left(\frac{G_y}{G_x} \right)$$

Figura 28: Encontrando el gradiente de la imagen. Tomado de: (https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html?ref=blog.roboflow.com).

3. *Non-Maximum Suppression:* Para pulir los bordes y eliminar anomalías durante la detección de bordes, se aplica una supresión de non-maximum. Este paso lo que hace es mantener los máximos locales en la dirección del gradiente. Con esto la idea es eliminar cualquier píxel que no sea considerado un borde.

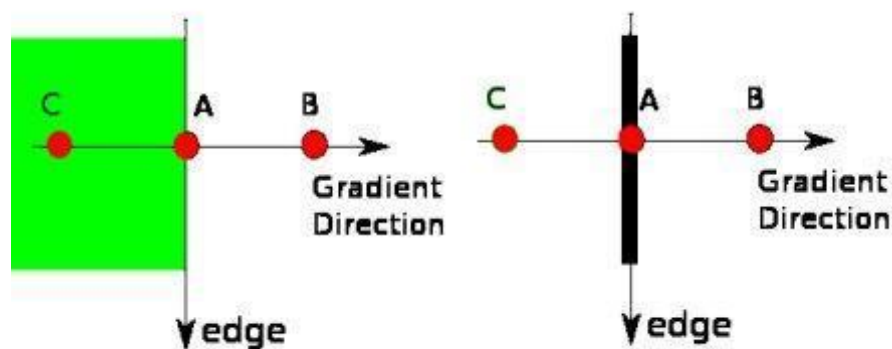


Figura 29: Supresión del valor no máximo. Tomado de: (https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html?ref=blog.roboflow.com)

Lo que observamos de la imagen anterior es que el punto A se encuentra localizado en el borde de la dirección vertical. Por lo tanto, la dirección del gradiente es perpendicular al borde. Los puntos B y C se encuentran en la dirección de la detección del gradiente (perpendicular al eje vertical) y por lo tanto el punto A es comparado con los puntos B y C para determinar si es que representa un máximo local. En caso de serlo, el punto A será parte de los valores iluminados para marcar bordes, en caso contrario será eliminado, haciendo su valor 0.

4. *Double Thresholding (Doble umbral)*: Después de la eliminación de los valores no máximos se avanza al siguiente paso que es el doble umbral. Aquí, los bordes se clasifican en “fuertes”, “débiles” y “sin borde”. Los bordes fuertes son aquellos cuyo valor está por encima del umbral en el límite superior del gradiente, mientras que los bordes que son débiles son aquellos cuyos valores se encuentran entre los límites superior e inferior del gradiente. Finalmente, los no bordes son todos aquellos que no alcanzan el límite inferior. Dado un gradiente de magnitud M y dos umbrales T_{high} y T_{low} , la clasificación puede expresarse matemáticamente como:

☐ Bordes Fuertes:

$$E_{strong}(i, j) = \begin{cases} 1 & \text{if } M(i, j) \geq T_{high} \\ 0 & \text{otherwise} \end{cases}$$

☐ Bordes débiles:

$$E_{weak}(i, j) = \begin{cases} 1 & \text{if } T_{low} \leq M(i, j) < T_{high} \\ 0 & \text{otherwise} \end{cases}$$

□ Sin bordes:

$$E_{\text{non-edge}}(i, j) = \begin{cases} 1 & \text{if } M(i, j) < T_{\text{low}} \\ 0 & \text{otherwise} \end{cases}$$

5. *Edge Tracking by Hysteresis (Seguimiento de bordes por histéresis)*: El paso final implica transponer la imagen para determinar cuáles bordes débiles están conectados con bordes fuertes. Únicamente los bordes débiles conectados a bordes fuertes se mantienen, ya que son considerados bordes reales. De esta manera se asegura de que el ruido y las variaciones pequeñas son ignorados, resultando en una detección de bordes más limpia.

- *Detector de bordes Laplaciano (Laplacian Edge Detection)*

La detección de bordes Laplaciana es una técnica de procesamiento de imágenes usada para resaltar áreas que tienen un cambio de intensidad rápido, las cuales normalmente se asocian a bordes dentro de las imágenes. A diferencia de las técnicas que se basan en el gradiente, la detección Laplaciana se basa en la segunda derivada de la intensidad de la imagen.

Para ello se hace uso del operador laplaciano, el cual se encarga de calcular la segunda derivada de la intensidad de la imagen. Podemos representar la segunda derivada de una imagen $f(x, y)$ como:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Esto puede ser implementado usando una operación de convolución con un kernel laplaciano de 3x3 como se muestra en la siguiente imagen:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figura 30: Matriz del kernel detector de bordes Laplaciano. Tomado de: (<https://docs.opencv.org/>).

- *Detección de bordes Prewitt (Prewitt Edge Detection)*

El detector de bordes Prewitt es una técnica usada para detectar bordes en imágenes digitales. Funciona de manera similar al detector Sobel y Canny por medio del cálculo de la magnitud del gradiente de la intensidad de la imagen usando la convolución con kernels de Prewitt. Los gradientes son usados para identificar cambios significativos en la intensidad que usualmente corresponden a bordes.

El detector Prewitt usa dos kernels, uno para detectar bordes en la dirección horizontal y el otro en dirección vertical. Al igual que otras técnicas, el detector Prewitt aplica convolución con los kernels.

☐ Kernel Horizontal (Gx):

$$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Figura 31: Matriz del kernel horizontal Prewitt. Tomado de: (<https://docs.opencv.org/>).

☐ Kernel Vertical (Gy):

$$Gy = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Figura 32: Matriz del kernel vertical Prewitt. Tomado de: (<https://docs.opencv.org/>).

- *Detección de bordes Prewitt (Prewitt Edge Detection)*

El detector de bordes de Robert es una técnica simple para la detección de bordes en las imágenes digitales. Funciona computando la magnitud del gradiente en la intensidad de una imagen, usando convolución con kernels de Robert. Este kernel es efectivo para detectar bordes delgados y prominentes. Esta técnica fue introducida en 1963 y fue uno de los primeros detectores de bordes. Este detector utiliza dos kernels (horizontal y vertical) y se aplican en la imagen usando convolución.

□ Kernel Horizontal de Robert (G_x):

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Figura 33: Matriz del kernel horizontal de Robert. Tomado de: (<https://docs.opencv.org/>).

□ Kernel Vertical de Robert (G_y):

$$G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Figura 34: Matriz del kernel vertical Robert. Tomado de: (<https://docs.opencv.org/>).

- *Detector de Bordos Scharr*

El detector Scharr es otro método para la detección de bordes en imágenes digitales. Este detector de bordes es mejor que el operador de Sobel. Al igual que Sobel o Canny, el operador consiste en dos kernels de convolución de 3x3, uno para aproximar el gradiente horizontal y otro para el gradiente

vertical. Estos kernels computan el gradiente en cada píxel y luego resaltan las áreas que tienen un cambio de intensidad rápido, lo cual se traduce en bordes.

El kernel horizontal (G_x) está diseñado para aproximar la tasa de cambio de intensidad en la dirección horizontal. Por otro lado, el kernel vertical (G_y) aproxima la tasa de cambio en la intensidad en dirección vertical de la imagen. Los kernels son definidos como:

□ Kernel de gradiente Horizontal (G_x):

$$G_x = \begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}$$

Figura 35: Matriz del kernel gradiente horizontal del detector de bordes Scharr. Tomado de: (<https://docs.opencv.org/>).

□ Kernel de gradiente Vertical (G_y):

$$G_y = \begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

Figura 36: Matriz del kernel gradiente vertical del detector de bordes Scharr. Tomado de: (<https://docs.opencv.org/>).

2.3.6 Técnicas de preprocesamiento de imágenes en la conducción autónoma

Las imágenes son tipos de datos no estructurados en los que las ciencias de la computación se han visto involucradas con la finalidad de obtener información más

precisa de estas, que pueda servir para mejorar la toma de decisiones de los algoritmos.

Afortunadamente ha probado ser de utilidad para la sociedad ya que hoy en día existen muchos problemas los cuales aún no se han resuelto y que aún siguen en búsqueda de soluciones, como la conducción bajo condiciones difíciles las cuales pueden ser, conducción con poca visibilidad como lluvia, nieve, neblina o la oscuridad misma.

Dependiendo del tipo de problema que se requiera resolver será el tipo de metodología de trabajo por el que uno pueda optar. A continuación veremos algunas aproximaciones usadas.

En el artículo de Viswanathan y Hussein (Viswanathan & Hussein, 2017, pp. 35-49) se realizaron estudios en el campo de la detección de vehículos autónomos. Características como detección de líneas de tráfico, detección de radares de límite de velocidad fueron estudiadas por varios estudios alrededor del mundo (Fernandes et al., 2021). La manera en la que los sistemas eran configurados era seleccionando un área de interés (*Region Of Interest*) y descartando información que no se quisiera, acto seguido se procedía con la detección de bordes Canny y transformaciones de Hough. Esta aproximación logró que se detectaran señales de tráfico buscando ciertas figuras en la imagen y luego buscando el color en la imagen si se encontraba.

Xue Li (International Congress on Image and Signal Processing, 2015) y su equipo trabajaron en mejorar la eficiencia de los algoritmos de detección de bordes. Su mejora, sin embargo, radicó más en el uso de la arquitectura de la red neuronal que en el uso convencional de la detección de bordes Canny. La red tipo spike es una red inspirada en simular cómo funcionan las redes neuronales humanas y con ellas una vez obtenidos los bordes se pasó mediante una transformación de Hough con su método Xue Li fue capaz de identificar una mayor cantidad de objetos con una mejor eficiencia.

Otro enfoque propuesto por *Guha en Application of Image Processing Techniques for Autonomous Cars* (Fernandes et al., 2020) se enfoca en detectar otros objetos durante el camino como señales para detenerse. Este fue implementado usando cascadas HARR. Por medio del uso de múltiples imágenes de ejemplos positivos y negativos fueron capaces de clasificar estos objetos y reaccionar de manera adecuada. También fueron capaces de predecir la distancia de un objeto al ajustar y calibrar la cámara y considerando la altura contra el piso.

Shah, Adhikari y Maheta (Fernandes et al., 2020) trataron de resolver este problema implementando detección de bordes obteniendo resultados similares. Pero usaron señales de la calle como flechas de derecha e izquierda. Esto nos demuestra que distintas aproximaciones al problema nos dan soluciones bastante similares.

Capítulo 3: Diseño

El diseño del proyecto de tesis se desarrolló con los siguientes objetivos en mente.

Objetivo General:

Desarrollar un análisis comparativo de técnicas de preprocesamiento de imágenes y datos LiDAR para conocer si el preprocesamiento de imágenes tiene una influencia en el desempeño del entrenamiento de redes neuronales en sistemas de conducción autónoma para su posterior optimización. Esto con el fin de mejorar la precisión y eficiencia de los vehículos autónomos en entornos complejos.

Objetivos Específicos:

1. Evaluar diferentes técnicas de preprocesamiento de imágenes y datos LiDAR utilizados en la conducción autónoma, identificando sus fortalezas y limitaciones en términos de precisión, velocidad de procesamiento y capacidad de integración.
2. Desarrollar un conjunto de pruebas y métricas para comparar el rendimiento de las redes neuronales entrenadas con datos preprocesados utilizando técnicas seleccionadas, tanto en simulaciones como en vehículos autónomos a pequeña escala desarrollados en el laboratorio.
3. Proponer una técnica de preprocesamiento optimizada que combine las mejores prácticas identificadas durante el análisis comparativo, orientada a mejorar la percepción del entorno y la toma de decisiones de los vehículos autónomos en diversos escenarios.

Todo esto con la finalidad de responder a la siguiente pregunta:

¿Aplicar preprocesamiento de imágenes afecta el desempeño de una red neuronal artificial para su capacidad de respuesta en vehículos autónomos?

3.1 Metodología

Recolección de datos

En esta etapa del proyecto, se conducirá el vehículo del laboratorio de ingeniería del lenguaje y el conocimiento (LKE) con el fin de obtener los datos que representan la materia prima para el trabajo de entrenamiento de algoritmos de conducción autónoma. Cabe mencionar que para la validez de los datos estos deben ser variados y en una gran cantidad. Para asegurar la variedad de datos recolectados, estos deben ser obtenidos de distintos ambientes, los cuales representen los ambientes en los cuales se desempeñará el vehículo. Por otro lado, para asegurar la cantidad de información necesaria se requerirá la recolección de datos por varios periodos de tiempo hasta conseguir al menos 50,000 instancias. La razón de este número de instancias es para que el algoritmo ViT pueda obtener y generalizar información de manera correcta, de lo contrario, se verá limitado para un análisis.

Análisis y limpieza de los datos

En esta etapa, la información recolectada deberá someterse a un análisis de qué es información útil. Así mismo realizar la limpieza de datos corruptos, incompletos o que no aportan utilidad al experimento. Además, se somete a correcciones en caso de ser necesario o su eliminación en caso de aportar consistencia al proyecto.

Integración de los datos

En esta etapa se procederá a integrar toda la información recolectada en un solo dataset, el cual servirá para desarrollar los subsets de entrenamiento y validación que serán introducidos en los algoritmos de aprendizaje automático. Aquí se analizará la integridad del dataset, lo cual significa que no exista duplicidad de información entre los datasets. Así mismo, que para cada registro dentro del dataset debe corresponder una imagen y un dato LiDAR respectivamente.

Preprocesamiento

Después de haber sido integrada la información. Se aplicará el preprocesamiento de imágenes. Con ello se modificará el tamaño y la composición de la imagen para la comparación.

Entrenamiento

Se definirán los algoritmos a aplicar y sobre ello como precisión, estabilidad del modelo. Se realizará el entrenamiento de los mismos obteniendo sus métricas principales de desempeño para análisis.

Evaluación y comparación de modelos

- Evaluación de Desempeño: Todos los modelos entrenados serán probados en condiciones similares usando los mismos conjuntos de validación y prueba. Se evaluará el desempeño de cada modelo.
- Análisis Comparativo: Los resultados obtenidos de los diferentes modelos serán comparados para determinar cuál técnica de preprocesamiento ofrece el mejor rendimiento en términos de precisión, velocidad de procesamiento y eficiencia general.

3.1.1 Infraestructura

Para el desarrollo del proyecto se contó con los siguientes componentes:

Hardware

- Computador personal Acer Nitro 5 con el procesador Intel Core i7 de 12^a generación
- NVIDIA Geforce RTX 4050
- Raspberry Pi 5
- Coche autónomo tipo Ackerman con sistema LiDAR (LD06) y cámara

Software

- Tensor Flow
- Python
- Bibliotecas Opencv

El presente proyecto se basa en el proyecto de tesis desarrollado por Sergio Delgadillo Marín, quien desarrolló un vehículo móvil no holomónico con dirección tipo Ackermann.

“Un robot móvil no holonómico es un tipo de robot que tiene limitaciones en su capacidad de movimiento debido a restricciones cinemáticas, es decir, no puede moverse libremente en cualquier dirección. En particular, el robot Ackermann es una de las configuraciones más conocidas, utilizando un sistema de dirección con ruedas orientables que permiten cambiar su dirección de movimiento sin cambiar su orientación . Sus limitaciones para moverse de forma lateral lo convierten en un desafío para la navegación autónoma en entornos complejos.” (Delgadillo, 2023).



Figura 37: Vehículo autónomo no holomónico estilo Akermann. Fuente propia.

El vehículo se conforma por una computadora de placa única (SBC por sus siglas en inglés) Raspberry Pi versión 5 con 8 GB de RAM

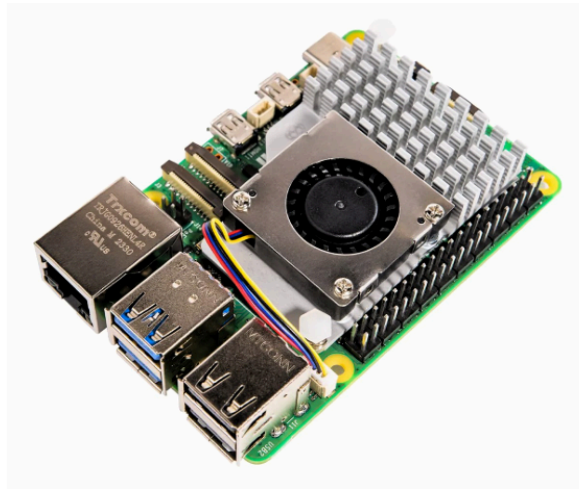


Figura 38: Raspberry Pi versión 5. Tomado de (Delgadillo, 2023).

Un chasis con configuración de dirección Ackermann elaborado con fibra de vidrio. Al mismo tiempo cuenta con 2 motores que operan a 12 voltios de corriente continua (VCC). El robot cuenta con una suspensión delantera basculante que le permite reducir las vibraciones transmitidas a los sensores, ocasionadas por las irregularidades en el terreno que el robot pudiera encontrar.



Figura 39: Chasis. Tomado de (Delgadillo, 2023).

El controlador de motores seleccionado es el doble puente H TB6612FNG, para regular la velocidad y el sentido del giro de manera independiente.

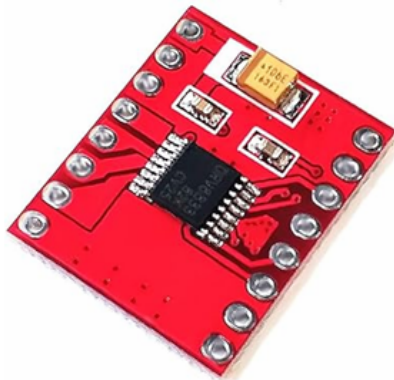


Figura 40: Puente H. Tomado de (Delgadillo, 2023).

El microcontrolador es el ESP32 DEVKIT V1 de 30 pines con conexión micro USB. Dispone de salidas digitales y PWM compatibles tanto con el controlador de motores como con el servomotor de la dirección. Además, el ESP32 cuenta con entradas digitales compatibles con los codificadores ópticos, las cuales incorporan interrupciones de hardware.



Figura 41: ESP32. Tomado de (Delgadillo, 2023).

Cuenta con baterías de polímero de litio (LiPo por sus siglas en inglés) de 11.1V y 3.3Ah conectadas en paralelo.

Cuenta con un sensor LiDAR modelo RP LIDAR C1, diseñado principalmente para aplicaciones en interiores, como robots aspiradores o proyectos educativos. Este sensor tiene un rango de detección máximo de 6 metros para superficies de color blanco con un 70% de reflectividad y de 12 metros para superficies de color negro con un 10% de reflectividad.

Finalmente cuenta con una cámara Raspberry Pi Rev 1.3, la cual cuenta con una resolución de 5 megapíxeles. Esto nos da una definición de imagen configurada de 800 * 600 píxeles (pág. 63 Reporte Sergio Delgadillo)

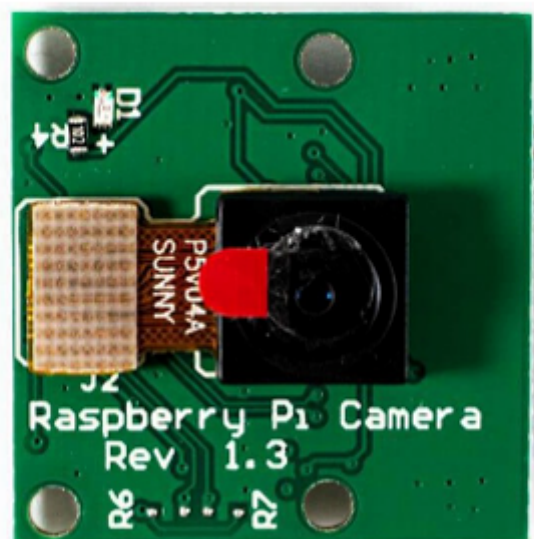


Figura 42: Cámara. Tomado de (Delgadillo, 2023).

La comunicación se realiza por medio de WiFi para la transmisión remota y en tiempo real del robot.

Finalmente cuenta con la versión de ROS 2 jazzy para el sistema de comunicación a nivel sistema operativo. Robot Operating System (ROS) es un conjunto de bibliotecas y herramientas para construir aplicaciones aplicadas a la robótica (Open

Robotics, 2022). ROS 2 tiene por objetivo el adaptarse a las necesidades de la comunidad dedicada a la robótica, ofreciendo algoritmos de última generación, herramientas de desarrollo y diversidad que permitan la implementación de proyectos de robótica de manera abierta para apoyar el desarrollo de robots que pueden comunicarse de manera estandarizada. Cada versión de ROS 2 se enlaza con una distribución de Ubuntu específica siendo Jazzy para Ubuntu 24 el cual es el sistema operativo con el que opera el vehículo autónomo.

Capítulo 4: Implementación, pruebas y resultados

4.1 Recolección de datos

Se inició con la recolección de imágenes para el proyecto. Para cumplir con las características previstas en el diseño del experimento. Se recolectaron datos en CU, Parque ecológico e interiores con la finalidad de obtener un dataset variado. Esto se logró consiguiendo más de 50 000 instancias. A continuación se muestran algunos ejemplos de las imágenes que integran el dataset.





Figura 43: Imágenes capturadas en Ciudad Universitaria de la Benemérita Universidad Autónoma de Puebla. Fuente propia.

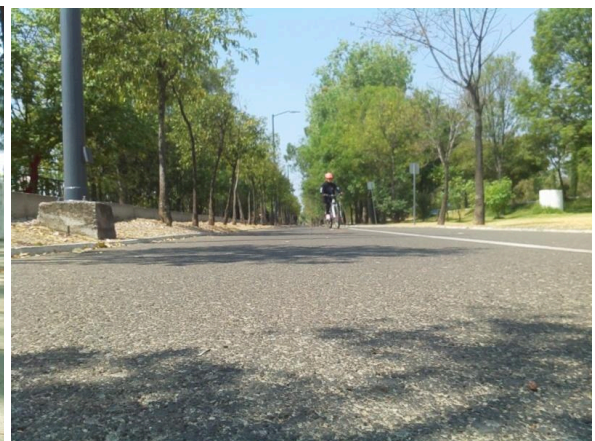
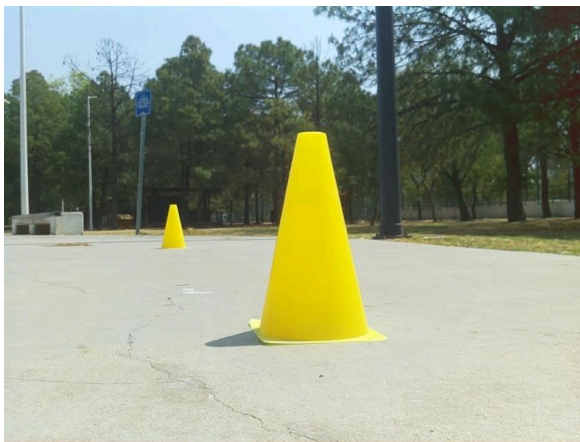


Figura 44: Imágenes capturadas en el parque ecológico Puebla, México. Fuente propia.

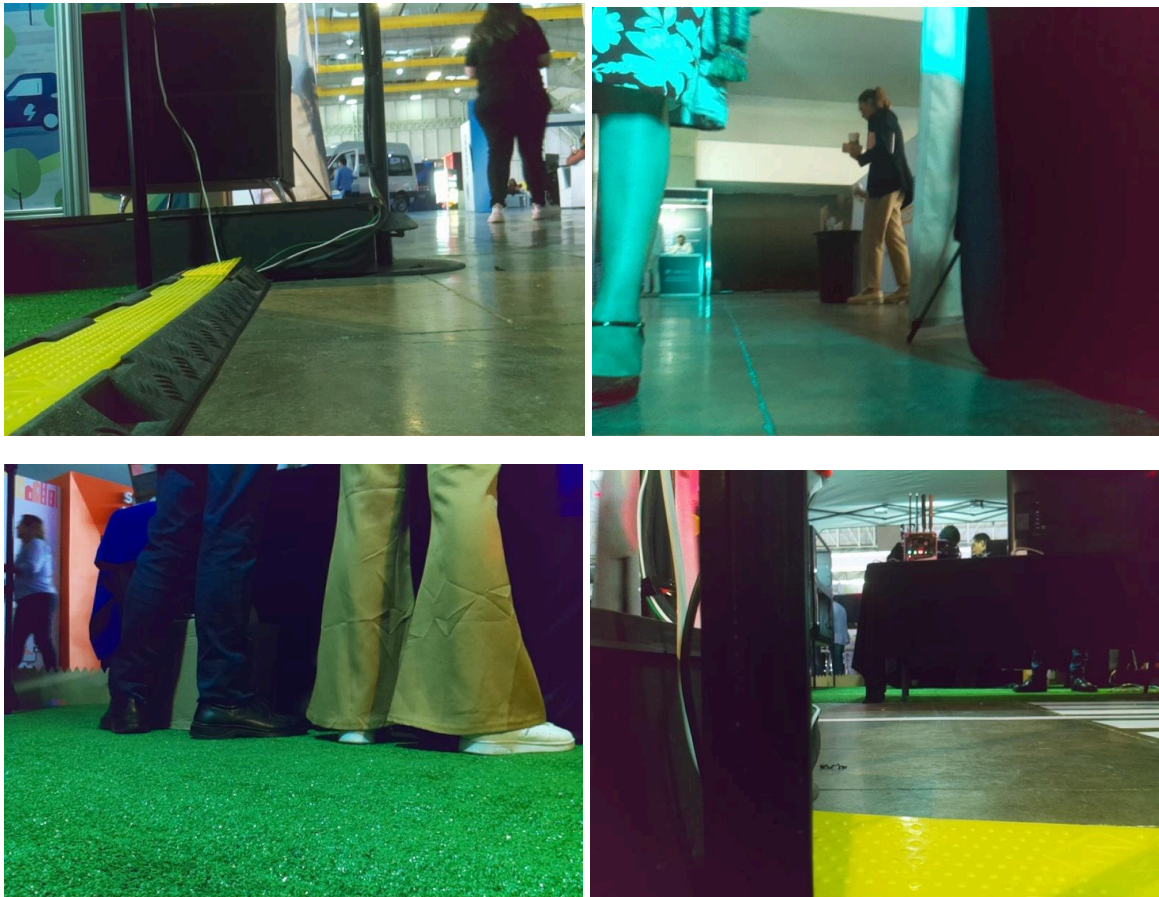
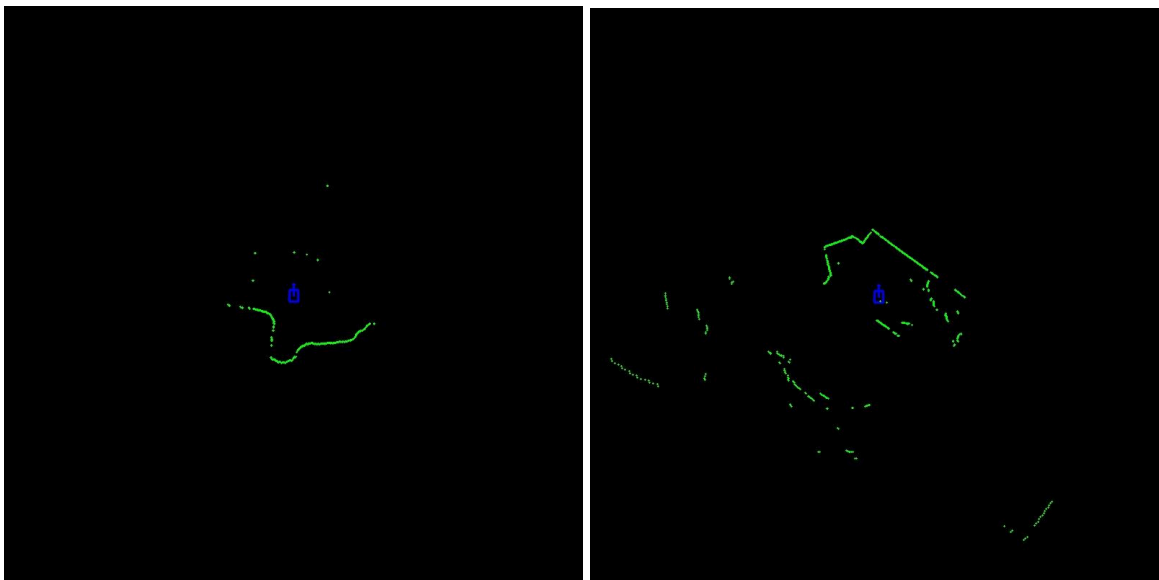


Figura 45: Imágenes capturadas en interiores para entrenamiento. Fuente propia.



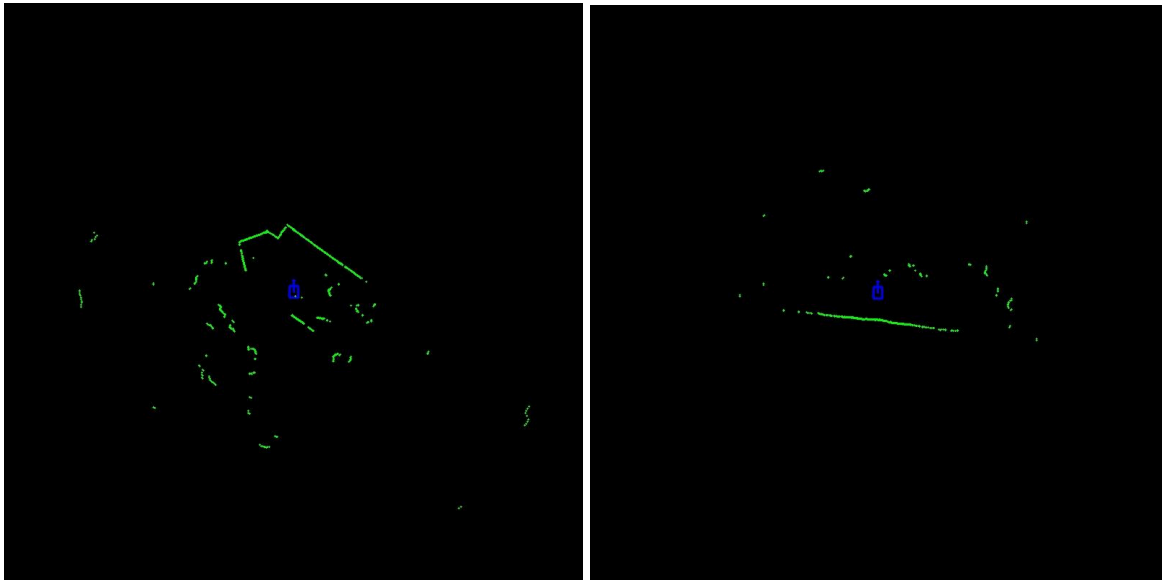


Figura 46: Imágenes LiDAR para entrenamiento. Fuente propia.

4.2 Análisis y limpieza de los datos

Durante el proceso de recolección se iban purificando los datos. La manera de proceder a esto fue la siguiente:

El robot al iniciarse y apagarse corta los procesos de golpe, por ello mismo, algunos archivos al inicio del dataset se corrompen y se eliminan al extraerlos. A continuación se analizan a grandes rasgos las imágenes capturadas para buscar imágenes corruptas o que no aporten información. Debido a que la captura de los datasets es un proceso llevado a cabo de manera personal, la mayoría de las imágenes son capturadas y los ambientes a los que se expone son elegidos deliberadamente para que aporten información. Reduciendo los problemas de limpieza.

Por ahora no nos preocupamos por el archivo csv debido a que una vez integrado en un solo dataset, un algoritmo buscará la integridad de los datos y eliminará los registros que no cumplan los requisitos. Esto no significó que no existieran imágenes que debieran ser eliminadas. A continuación algunos ejemplos:



Figura 47: Imágenes cuya toma afectan la captura de información o que podrían dar como resultado alucinaciones, también imágenes que simplemente no existían en el registro csv debido al encendido o apagado del vehículo por lo que no tenían información que aportar al entrenamiento. Fuente propia.

4.3 Integración de los datos

Para lograr la integración de los datasets, había que ser cuidadosos de mantener la información de los csv y al mismo tiempo que pudiera estar ligada a los datos recolectados. Para esto se desarrolló un script el cual lee una carpeta raíz (DIR_ROOT) y con ello comenzamos a trabajar para agregar todos nuestros datos recolectados para unificarlos en un dataset global.

El algoritmo construye un dataset multimodal unificado a partir de los registros generados durante la recolección de datos. Ya que los registros están compuestos por datos estructurados (CSV) y datos no estructurados (imágenes y LiDAR) se realiza una indexación de todas las imágenes y datos LiDAR. Posteriormente se identifican y unifican todos los archivos CSV correspondientes a los registros del

sistema para la construcción del dataset global.

Después, se construyen las rutas relativas para su uso durante el entrenamiento de las redes neuronales. Una vez terminado el paso anterior se procede a copiar las imágenes a las nuevas rutas creadas para que se consoliden en un solo directorio y después se guarda el archivo CSV.

Pseudocódigo 1: Integración de los datasets en un solo csv global.

Algorithm 1: Construcción del Dataset Multimodal

Input: Directorio raíz con subdirectorios de imágenes RGB, imágenes LiDAR y archivos CSV

Output: Dataset unificado en CSV y directorios consolidados de imágenes RGB y LiDAR

1 **Procedimiento Principal;**

2 Definir el directorio raíz de trabajo;

3 Definir el directorio de entrada de registros (**data_logs**);

4 Definir el directorio de salida del dataset consolidado;

5 Crear los directorios de salida para imágenes RGB;

6 Crear los directorios de salida para imágenes LiDAR;

7 **Lllamar a Subproceso 1: Indexación de Imágenes;**

8 **Lllamar a Subproceso 2: Lectura y Unificación de Archivos CSV;**

9 **Lllamar a Subproceso 3: Generación de Rutas Relativas;**

10 **Lllamar a Subproceso 4: Copia Controlada de Archivos;**

11 **Lllamar a Subproceso 5: Almacenamiento del Dataset Final;**

Algorithm 2: Subproceso 1: Indexación de Imágenes

Input: Directorio de registros

Output: Diccionarios de imágenes RGB y LiDAR

1 Inicializar un diccionario vacío para imágenes RGB;

2 Inicializar un diccionario vacío para imágenes LiDAR;

3 **foreach** *archivo con extensión .jpg en el directorio de registros* **do**

4 Obtener el nombre del archivo;

5 **if** *nombre inicia con image_* **then**

6 Asociar el nombre con su ruta completa en el índice RGB;

7 **else if** *nombre inicia con lidar_image_* **then**

8 Asociar el nombre con su ruta completa en el índice LiDAR;

Algorithm 3: Subproceso 2: Lectura y Unificación de Archivos CSV

Input: Archivos CSV en el directorio de registros

Output: Dataset unificado de todas las estructuras tabulares

1 Localizar todos los archivos CSV que cumplan patrón **data_log_*.csv**;

2 **if** *no se encuentra ningún archivo CSV* **then**

3 Finalizar el algoritmo con error;

4 Inicializar lista vacía de estructuras tabulares;

5 **foreach** *archivo CSV encontrado* **do**

6 Leer el archivo CSV;

7 Almacenar la estructura en la lista;

8 Unificar todas las estructuras en un único dataset;

Algorithm 4: Subproceso 3: Generación de Rutas Relativas

Input: Dataset unificado

Output: Rutas relativas de imágenes RGB y LiDAR

```
1 foreach registro del dataset do  
2   Extraer nombre de archivo RGB;  
3   Construir ruta relativa de salida para RGB;  
4   Extraer nombre de archivo LiDAR;  
5   Construir ruta relativa de salida para LiDAR;
```

Algorithm 5: Subproceso 4: Copia Controlada de Archivos

Input: Dataset unificado, índices de imágenes

Output: Archivos copiados al directorio de salida, contadores de faltantes

```
1 Inicializar contador de imágenes RGB faltantes;  
2 Inicializar contador de imágenes LiDAR faltantes;  
3 foreach registro del dataset do  
4   Imágenes RGB;  
5   Obtener nombre del archivo RGB;  
6   if archivo existe en índice RGB then  
7     if archivo no existe en directorio de salida then  
8       Copiar archivo RGB;  
9   else  
10    Incrementar contador de imágenes RGB faltantes;  
11  Imágenes LiDAR;  
12  Obtener nombre del archivo LiDAR;  
13  if archivo existe en índice LiDAR then  
14    if archivo no existe en directorio de salida then  
15      Copiar archivo LiDAR;  
16  else  
17    Incrementar contador de imágenes LiDAR faltantes;
```

Algorithm 6: Subproceso 5: Almacenamiento del Dataset Final

Input: Dataset unificado

Output: CSV final y confirmaciones

```
1 Guardar dataset unificado en CSV sin eliminar información previa;  
2 Mostrar mensajes de confirmación y rutas finales;
```

Esto nos deja con un dataset de +61 000 instancias

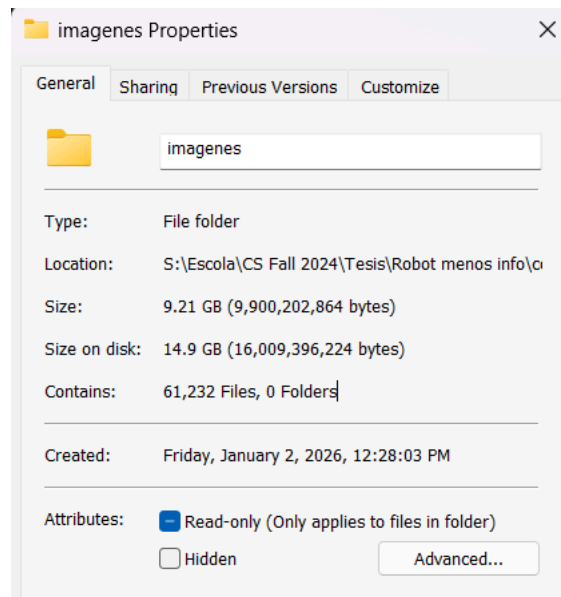


Figura 48: Número de imágenes (instancias) recolectadas.

Sin embargo, como se mencionó durante el diseño del experimento, se necesita que el dataset sea íntegro y que no existan valores duplicados. Así mismo, si durante el proceso hubo información que se perdió o que no existía, se necesita verificar la integridad del dataset completo. Para ello se desarrolló un código que verifica que no existen registros duplicados y en caso de existir eliminarlos.

Tomando el archivo del algoritmo anterior se busca el registro que corresponde al nombre de la imagen asociados para generar una clave de pertenencia. De esta manera se recorre todo el dataset de manera secuencial y en caso de encontrar un registro que no tenga una asociación a una imagen y la clave no haya sido observada previamente son agregados al dataset final. De esta manera se eliminan datos que hayan sido duplicados evitando así redundancia. Al terminar se muestran las métricas de cuantos archivos fueron eliminados para de esta manera conocer el grado de consistencia del dataset generado.

Pseudocódigo 2: Código deduplicate.py lo que hace es buscar que exista una única instancia de cada registro y elimina instancias duplicadas, creando el archivo global_dedup.csv

Algorithm 7: Deduplicación del Dataset Multimodal

Input: Dataset global en formato CSV con rutas a imágenes RGB y LiDAR

Output: Dataset depurado sin registros duplicados en un nuevo CSV

1 Procedimiento Principal;

2 Definir el directorio raíz del proyecto;

3 Definir la ruta del dataset global de entrada;

4 Definir la ruta del archivo CSV depurado;

5 Leer el CSV de entrada y almacenarlo como estructura tabular;

6 Llamar a Subproceso 1: Definición de la Clave de Muestreo;

7 Llamar a Subproceso 2: Eliminación de Registros Duplicados;

8 Llamar a Subproceso 3: Construcción del Dataset Depurado;

9 Llamar a Subproceso 4: Verificación de Integridad;

Algorithm 8: Subproceso 1: Definición de la Clave de Muestreo

Input: Registro del dataset

Output: Clave única del registro

1 Definir función que genere la clave de un registro usando:

- Nombre del archivo de imagen RGB

- Nombre del archivo de imagen LiDAR

Algorithm 9: Subproceso 2: Eliminación de Registros Duplicados

Input: Dataset global

Output: Lista de registros únicos

1 Inicializar un conjunto vacío para claves únicas;

2 Inicializar una lista vacía para registros no duplicados;

3 foreach *registro en el dataset original* **do**

4 | Generar clave del registro;

5 | **if** *clave no pertenece al conjunto de claves únicas* **then**

6 | | Agregar clave al conjunto;

7 | | Almacenar registro en lista de registros válidos;

Algorithm 10: Subproceso 3: Construcción del Dataset Depurado

Input: Lista de registros únicos

Output: Dataset depurado

1 Crear nueva estructura tabular a partir de registros únicos;

2 Reindexar el dataset resultante para numeración consecutiva;

3 Almacenar dataset depurado en un nuevo archivo CSV;

Algorithm 11: Subproceso 4: Verificación de Integridad

Input: Dataset original y dataset depurado

Output: Métricas de validación

1 Calcular número de registros originales;

2 Calcular número de registros finales;

3 Determinar número de duplicados eliminados;

4 Mostrar métricas y ruta del archivo generado;

Después de ello ejecutamos el código clean_dataset.py el cual busca confirmar que el dataset es correcto y crear un nuevo archivo csv de nombre global_clean.csv.

El objetivo de este algoritmo es realizar una limpieza final del dataset verificando la existencia de una dupla (imagen, LiDAR) para cada registro en el CSV. Aunque anteriormente se había eliminado la duplicidad de los registros, existe otro problema inherente del diseño del vehículo empleado para la recolección de información.

Al encender el vehículo el sistema ROS 2 lanza en automático los nodos programados para comenzar con la captura de la información, sin embargo, no todos los sistemas están cargados en el mismo momento creando una disparidad entre la cámara y los sensores de conducción. Esto se ve reflejado teniendo archivos corruptos al inicio del vehículo. Lo mismo sucede al apagarlo, debido a que no da tiempo de cerrar los programas hace que los últimos archivos aunque existan estén corruptos. Debido a que los datos son la parte más importante en un proyecto de IA el mantener la integridad de los mismos es crucial para poder asegurar su correcta aplicación e interpretabilidad de los resultados.

Para solucionar esto el algoritmo evalúa la existencia de los archivos en las carpetas de imagen y LiDAR correspondiendo al dato recolectado en el dataset. De esta manera en caso de que los archivos se hayan eliminado, movido o perdido, se puede tener un dataset que contenga información que pueda ser procesada por los algoritmos de DL así garantizando portabilidad y coherencia estructural. Así se guarda un nuevo dataset limpio de los datos que son completos.

Pseudocódigo 3: clean_dataset.py Crea un nuevo dataset de nombre global_clean.csv con una relación de 1 a 1 en el dataset buscando que exista para cada registro una imagen y una imagen LiDAR para asegurar la integridad del dataset.

Algorithm 12: Limpieza del Dataset mediante Verificación de Archivos

Input: Dataset depurado sin duplicados en CSV, con rutas relativas a imágenes RGB y LiDAR

Output: Dataset limpio con registros válidos cuyos archivos existen físicamente

1 Procedimiento Principal;

2 Definir el directorio raíz del proyecto;

3 Definir el directorio base del dataset consolidado;

4 Definir la ruta del archivo CSV de entrada (dataset sin duplicados);

5 Leer el archivo CSV de entrada y almacenarlo como estructura tabular;

6 Llamar a Subproceso 1: Verificación de Existencia de Archivos;

7 Llamar a Subproceso 2: Filtrado del Dataset;

8 Llamar a Subproceso 3: Almacenamiento y Reporte;

Algorithm 13: Subproceso 1: Verificación de Existencia de Archivos	
Input:	Dataset depurado
Output:	Máscara lógica de registros válidos
1	Definir función que, dada una ruta relativa, verifique si el archivo existe en el directorio del dataset;
2	foreach <i>registro en el dataset</i> do
3	Evaluar existencia de archivo RGB;
4	Evaluar existencia de archivo LiDAR;
5	Construir máscara lógica indicando si ambos archivos existen;
Algorithm 14: Subproceso 2: Filtrado del Dataset	
Input:	Dataset depurado, máscara lógica de registros válidos
Output:	Dataset filtrado
1	Seleccionar registros que cumplen condición de validez;
2	Reindexar dataset resultante para numeración consecutiva;
Algorithm 15: Subproceso 3: Almacenamiento y Reporte	
Input:	Dataset filtrado
Output:	CSV limpio y métricas informativas
1	Definir ruta del archivo CSV de salida (dataset limpio);
2	Almacenar dataset limpio en formato CSV;
3	Calcular número de registros originales;
4	Calcular número de registros finales válidos;
5	Mostrar mensajes con métricas y ruta del archivo generado;

```

Filas finales      : 61658
Duplicados quitados: 3607
Archivo generado  : S:\Escola\CS Fall 2024\Tesis\Robot menos info\copias de datos\project_root\robot\global_dedup.csv
python -m clean_dataset
Open file in editor (ctrl + click)
S:\Escola\CS Fall 2024\Tesis\Robot menos info\copias de datos\project root\robot\global_clean.csv
Filas originales: 61658
Filas limpias: 61232
PS S:\Escola\CS Fall 2024\Tesis\Robot menos info\copias de datos\project_root>

```

Figura 49: Ejecución de los códigos deduplicate y clean_dataset.

Finalmente, ejecutamos el código `validate_dataset.py` para asegurarnos de que nuestro dataset es íntegro y existe una sola instancia de `dato:imagen:imagen_lidar`. Este algoritmo tiene por función únicamente verificar que el trabajo previamente realizado por los algoritmos anteriores realmente se haya aplicado de manera correcta, para así evitar problemas de incompletitud de datos.

Este script recorre cada registro del dataset limpio y busca cualquier inconsistencia que no se haya realizado correctamente en los anteriores. Busca que exista la misma cantidad de registros en el CSV como imágenes y archivos LiDAR así como su consistencia en nombres para su correcto uso en el entrenamiento. En caso contrario marca cuales son las inconsistencias para que se apliquen los scripts

anteriores y se pueda volver a verificar. El porqué de este algoritmo es debido a que mientras más información se va recolectando se puede ir perdiendo el flujo y con esto se busca el que independientemente de la persona que llegase a trabajar en el proyecto en el futuro, pueda integrar y formar una idea de cuán completo es el dataset.

Pseudocódigo 4: código validate_dataset.py

Algorithm 16: Verificación de Integridad del Dataset Multimodal

Input: Dataset limpio en CSV con rutas relativas a imágenes RGB y LiDAR

Output: Reporte de integridad indicando presencia o ausencia de archivos faltantes

1 **Procedimiento Principal;**

2 Definir el directorio raíz del proyecto;

3 Definir el directorio base del dataset consolidado;

4 Definir la ruta del archivo CSV del dataset limpio;

5 Leer el CSV y almacenarlo como estructura tabular;

6 **Llamar a Subproceso 1: Inicialización de Estructuras de Validación;**

7 **Llamar a Subproceso 2: Verificación de Integridad por Registro;**

8 **Llamar a Subproceso 3: Reporte de Resultados;**

Algorithm 17: Subproceso 1: Inicialización de Estructuras de Validación

Input: Dataset limpio

Output: Listas de índices de registros faltantes

1 Inicializar lista vacía para índices de registros con imágenes RGB faltantes;

2 Inicializar lista vacía para índices de registros con imágenes LiDAR faltantes;

Algorithm 18: Subproceso 2: Verificación de Integridad por Registro

Input: Dataset limpio

Output: Listas actualizadas de registros con archivos faltantes

1 **foreach** registro en el dataset **do**

2 Construir ruta absoluta del archivo de imagen RGB;

3 Construir ruta absoluta del archivo de imagen LiDAR;

4 **if** archivo RGB no existe **then**

5 Almacenar índice del registro en lista de RGB faltantes;

6 **if** archivo LiDAR no existe **then**

7 Almacenar índice del registro en lista de LiDAR faltantes;

Algorithm 19: Subproceso 3: Reporte de Resultados

Input: Listas de registros con archivos faltantes

Output: Mensaje de integridad del dataset

1 Calcular número total de registros evaluados;

2 Calcular número de registros con imágenes RGB faltantes;

3 Calcular número de registros con imágenes LiDAR faltantes;

4 **if** existe al menos un registro con archivos faltantes **then**

5 Indicar que el dataset no está completamente íntegro;

6 **else**

7 Indicar que el dataset está íntegro y listo para entrenamiento;

```
PS S:\Escola\CS Fall 2024\Tesis\Robot menos info\copias de datos\project_root> python -m validate_dataset
===== VALIDACIÓN DATASET =====
Total filas: 61232
RGB faltantes: 0
LiDAR faltantes: 0
Dataset íntegro y listo para entrenamiento
PS S:\Escola\CS Fall 2024\Tesis\Robot menos info\copias de datos\project_root> []
```

Figura 50: Ejecución del código `validate_dataset.py` para asegurar la integridad de los datos.

Con estos pasos se terminó la fase de integración del dataset y se procedió al siguiente paso.

4.4 Preprocesamiento

En este trabajo las técnicas de preprocesamiento utilizadas fueron únicamente de carácter radiométrico; debido a la naturaleza del proyecto, no se usaron imágenes satelitales o a distancias que requieren una corrección de tipo geométrico.

Específicamente se usaron las siguientes técnicas de preprocesamiento en:

- Transformación de espacio de colores de BGR que es el formato original que captura la cámara a RGB y HSV.
- Conversión a escala de grises
- Reescalado de imagen a 128x128
- Eliminación de ruido con suavizado gaussiano
- Aplicación de filtro de detección de bordes con filtro Sobel.

Otras de las opciones que se manejaron fueron la transformada de Hough o el filtro de Prewitt. La razón de no aplicar estos fue para Prewitt, que es un filtro antiguo y no se usa actualmente, filtros más potentes como Canny o Sobel son más efectivos. Por el lado de la transformada de Hough no se aplica debido a que opera buscando reglas específicas predefinidas por el usuario como líneas de distribución específicas. Dado que queremos obtener una generalización y el vehículo se operó en distintas áreas, no existen relaciones visibles más allá de la existencia de un carril o no. Por ello se optó por aplicar filtros que fueran globales.

A continuación se muestran los resultados de las transformaciones aplicadas.

Imagen original



Características

Property	Value
Image ID	
Dimensions	800 x 600
Width	800 pixels
Height	600 pixels
Horizontal resolution	96 dpi
Vertical resolution	96 dpi
Bit depth	24
Compression	
Resolution unit	
Color representation	
Compressed bits/pixel	

Imagen RGB



Características

Image

Image ID

Dimensions 128 x 128

Width 128 pixels

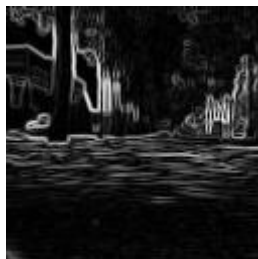
Height 128 pixels

Horizontal resolution 96 dpi

Vertical resolution 96 dpi

Bit depth 24

Imagen Sobel



Características

Image

Image ID

Dimensions 128 x 128

Width 128 pixels

Height 128 pixels

Horizontal resolution 96 dpi

Vertical resolution 96 dpi

Bit depth 24

Imagen HSV espectro de grises



Características

Image

Image ID

Dimensions	128 x 128
Width	128 pixels
Height	128 pixels
Horizontal resolution	96 dpi
Vertical resolution	96 dpi
Bit depth	24

Para llegar a estos resultados, lo que se realizó fue lo siguiente. Dentro del espacio de trabajo se creó una carpeta (preprocess) para realizar cambios y modificaciones a las imágenes como se muestra a continuación.

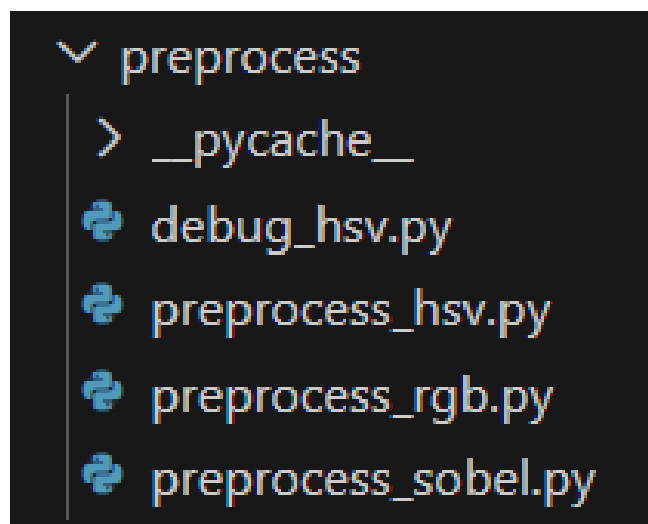


Figura 51: Directorio de códigos para preprocesamiento de imágenes.

Se realizaron 3 con la intención de poder debuggear cuando se requería. Sin embargo, se reconoce que todo podría llevarse a cabo en un único script.

RGB

- Se carga el dataset de imágenes.
- Se modifica la imagen del formato BGR a RGB.
- Se aplica un rescalado de la imagen a 128x128.
- Se normaliza el conjunto de imágenes (se divide entre 255 para tener un valor de píxel [0,1]).
- Se guarda la imagen en una nueva para su posterior consumo.

Pseudoódigo 5: código preprocess_rgb.py

Algorithm 20: Preprocesamiento de Imágenes RGB

Input: Dataset limpio en CSV con rutas a imágenes RGB, directorio de imágenes originales

Output: Imágenes RGB preprocesadas, normalizadas y redimensionadas

- 1 **Procedimiento Principal;**
 - 2 Definir directorio raíz del proyecto;
 - 3 Definir ruta del CSV del dataset limpio;
 - 4 Definir directorio de imágenes RGB originales;
 - 5 Definir directorio de salida para imágenes preprocesadas;
 - 6 Definir tamaño objetivo de las imágenes;
 - 7 Crear directorio de salida si no existe;
 - 8 **Llamar a Subproceso 1: Carga del Dataset;**
 - 9 **Llamar a Subproceso 2: Preprocesamiento de Imágenes;**
 - 10 **Llamar a Subproceso 3: Reporte del Proceso;**
-

Algorithm 21: Subproceso 1: Carga del Dataset

Input: CSV del dataset limpio

Output: Estructura tabular y contadores inicializados

- 1 Leer CSV y almacenar contenido en estructura tabular;
 - 2 Inicializar contador de imágenes procesadas;
 - 3 Inicializar contador de imágenes omitidas;
 - 4 Inicializar contador de imágenes faltantes o con error;
-

Algorithm 22: Subproceso 2: Preprocesamiento de Imágenes

Input: Rutas de imágenes RGB

Output: Imágenes procesadas y almacenadas

- 1 **foreach** *ruta de imagen RGB en el dataset* **do**
- 2 Extraer nombre del archivo de imagen;
- 3 Construir ruta absoluta de archivo fuente;
- 4 Construir ruta absoluta de archivo destino;
- 5 **if** *archivo preprocesado ya existe* **then**
- 6 Incrementar contador de imágenes omitidas;
- 7 Continuar con siguiente iteración;
- 8 **if** *archivo fuente no existe* **then**
- 9 Incrementar contador de imágenes faltantes;
- 10 Continuar con siguiente iteración;
- 11 Leer imagen desde disco;
- 12 **if** *lectura falla* **then**
- 13 Incrementar contador de imágenes faltantes;
- 14 Continuar con siguiente iteración;

15	Convertir imagen de BGR a RGB;
16	Redimensionar imagen al tamaño objetivo;
17	Normalizar valores de píxeles al rango [0, 1];
18	Reescalar valores de píxeles al rango [0, 255];
19	Convertir imagen a tipo entero sin signo;
20	Guardar imagen preprocesada en directorio de salida;
21	Incrementar contador de imágenes procesadas;

Algorithm 23: Subproceso 3: Reporte del Proceso

Input: Contadores de procesamiento

Output: Resumen y finalización

1	Mostrar número de imágenes procesadas;
2	Mostrar número de imágenes omitidas;
3	Mostrar número de imágenes faltantes o con error;
4	Indicar finalización exitosa del preprocesamiento;

Sobel

- Se carga el dataset de imágenes.
- Se convierte la imagen del formato BGR a escala de grises.
- Se aplica un rescalado de la imagen a 128x128.
- Se calcula el gradiente de la imagen mediante el operador Sobel en las direcciones horizontal y vertical.
- Se obtiene la magnitud del gradiente combinando ambas direcciones.
- Se normaliza la imagen resultante al rango de valores [0, 255].
- Se replica la imagen en tres canales para mantener compatibilidad con modelos CNN.
- Se guarda la imagen procesada para su posterior consumo.

Pseudocódigo 6: código preprocess_sobel.py

Algorithm 24: Preprocesamiento de Imágenes mediante Filtro Sobel

Input: Dataset limpio en CSV con rutas a imágenes, directorio de imágenes originales

Output: Imágenes preprocesadas mediante operador Sobel

1	Procedimiento Principal;
2	Definir directorio raíz del proyecto;
3	Definir ruta del CSV del dataset limpio;
4	Definir directorio de imágenes originales;
5	Definir directorio de salida para imágenes Sobel;
6	Definir tamaño objetivo de las imágenes;
7	Crear directorio de salida si no existe;
8	Llamar a Subproceso 1: Carga del Dataset;
9	Llamar a Subproceso 2: Preprocesamiento de Imágenes Sobel;
10	Llamar a Subproceso 3: Reporte del Proceso;

Algorithm 25: Subproceso 1: Carga del Dataset

Input: CSV del dataset limpio

Output: Estructura tabular y contadores inicializados

- 1 Leer CSV y almacenar contenido en estructura tabular;
 - 2 Inicializar contador de imágenes procesadas;
 - 3 Inicializar contador de imágenes omitidas;
 - 4 Inicializar contador de imágenes faltantes o con error;
-

Algorithm 26: Subproceso 2: Preprocesamiento de Imágenes Sobel

Input: Rutas de imágenes

Output: Imágenes procesadas con operador Sobel

- 1 **foreach** *ruta de imagen en el dataset* **do**
 - 2 Extraer nombre del archivo de imagen;
 - 3 Construir ruta absoluta del archivo fuente;
 - 4 Construir ruta absoluta del archivo destino;
 - 5 **if** *archivo de salida ya existe* **then**
 - 6 Incrementar contador de imágenes omitidas;
 - 7 Continuar con siguiente iteración;
 - 8 **if** *archivo fuente no existe* **then**
 - 9 Incrementar contador de imágenes faltantes;
 - 10 Continuar con siguiente iteración;
 - 11 Leer imagen desde disco;
 - 12 **if** *lectura falla* **then**
 - 13 Incrementar contador de imágenes faltantes;
 - 14 Continuar con siguiente iteración;
 - 15 Convertir imagen de BGR a escala de grises;
 - 16 Redimensionar imagen al tamaño objetivo;
 - 17 Aplicar operador Sobel en dirección horizontal;
 - 18 Aplicar operador Sobel en dirección vertical;
 - 19 Calcular magnitud del gradiente;
 - 20 Normalizar valores al rango [0,255];
 - 21 Convertir imagen a tipo entero sin signo;
 - 22 Replicar imagen a tres canales;
 - 23 Guardar imagen procesada en directorio de salida;
 - 24 Incrementar contador de imágenes procesadas;
-

Algorithm 27: Subproceso 3: Reporte del Proceso

Input: Contadores de procesamiento

Output: Resumen del preprocesamiento

- 1 Mostrar número de imágenes procesadas;
 - 2 Mostrar número de imágenes omitidas;
 - 3 Mostrar número de imágenes faltantes o con error;
 - 4 Indicar finalización del preprocesamiento Sobel;
-

HSV

- Se carga el dataset de imágenes.
- Se convierte la imagen del formato BGR a RGB.
- Se aplica un rescalado de la imagen a 128x128.
- Se transforma la imagen del espacio de color RGB al espacio de color HSV.
- Se separan los canales de tono (H), saturación (S) y valor (V).

- Se aplica ecualización adaptativa del histograma (CLAHE) sobre el canal valor (V) para mejorar el contraste.
- Se reconstruye una imagen de tres canales a partir del canal procesado.

Pseudocódigo 7: código preprocess_hsv.py

Algorithm 28: Preprocesamiento de Imágenes en el Espacio de Color HSV

Input: Dataset limpio en CSV con rutas a imágenes, directorio de imágenes originales

Output: Imágenes preprocesadas en espacio HSV con realce de contraste

```

1 Procedimiento Principal;
2 Definir directorio raíz del proyecto;
3 Definir ruta del CSV del dataset limpio;
4 Definir directorio de imágenes originales;
5 Definir directorio de salida para imágenes HSV;
6 Definir tamaño objetivo de las imágenes;
7 Crear directorio de salida si no existe;
8 Llamar a Subproceso 1: Carga del Dataset;
9 Llamar a Subproceso 2: Preprocesamiento de Imágenes HSV;
10 Llamar a Subproceso 3: Reporte del Proceso;

```

Algorithm 29: Subproceso 1: Carga del Dataset

Input: CSV del dataset limpio

Output: Estructura tabular y contadores inicializados

```

1 Leer CSV y almacenar contenido en estructura tabular;
2 Inicializar contador de imágenes procesadas;
3 Inicializar contador de imágenes omitidas;
4 Inicializar contador de imágenes faltantes o con error;

```

Algorithm 30: Subproceso 2: Preprocesamiento de Imágenes HSV

Input: Rutas de imágenes

Output: Imágenes procesadas en espacio HSV

```

1 foreach ruta de imagen en el dataset do
2   Extraer nombre del archivo de imagen;
3   Construir ruta absoluta del archivo fuente;
4   Construir ruta absoluta del archivo destino;
5   if archivo de salida ya existe then
6     Incrementar contador de imágenes omitidas;
7     Continuar con siguiente iteración;
8   if archivo fuente no existe then
9     Incrementar contador de imágenes faltantes;
10    Continuar con siguiente iteración;
11  Leer imagen desde disco;
12  if lectura falla then
13    Incrementar contador de imágenes faltantes;
14    Continuar con siguiente iteración;
15  Convertir imagen de BGR a RGB;
16  Redimensionar imagen al tamaño objetivo;
17  Convertir imagen de RGB a HSV;
18  Separar canales H, S y V;
19  Aplicar CLAHE sobre el canal V;
20  Reconstruir imagen de tres canales replicando el canal V procesado;
21  Guardar imagen procesada en directorio de salida;
22  Incrementar contador de imágenes procesadas;

```

Algorithm 31: Subproceso 3: Reporte del Proceso

Input: Contadores de procesamiento

Output: Resumen del preprocesamiento

- 1 Mostrar número de imágenes procesadas;
 - 2 Mostrar número de imágenes omitidas;
 - 3 Mostrar número de imágenes faltantes o con error;
 - 4 Indicar finalización del preprocesamiento HSV;
-

El objetivo principal de esta etapa fue estandarizar las características visuales de las imágenes con el fin de hacerlas compatibles con los requerimientos de entrada de modelos de DL. Cabe mencionar que esto crea nuevos datasets no sobrescribe el anterior por lo que terminamos con 3 datasets preprocesados.

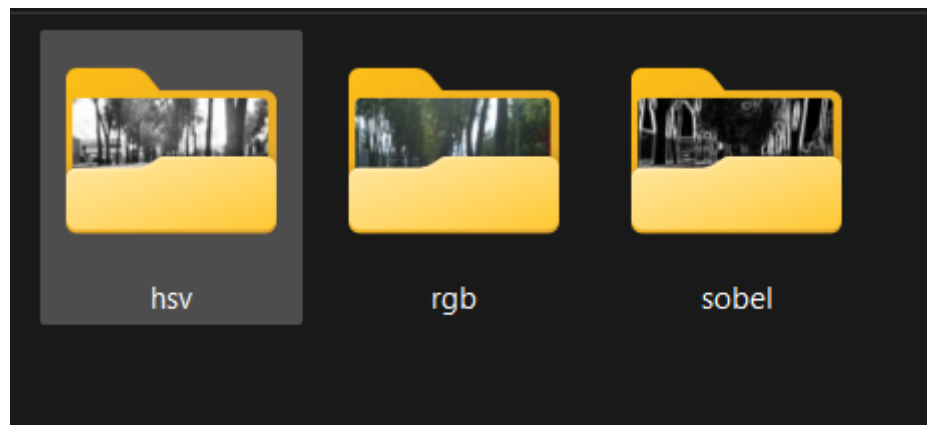


Figura 52: Directorios de imágenes preprocesadas.

4.5 Entrenamiento

Ya con los datos listos para entrenamiento, lo que sigue es elegir las arquitecturas de las redes a entrenar. Como se mencionó previamente, para la comparación se entrenaron dos redes neuronales que aplican distintas filosofías para extraer características y luego inferir.

Primero una red neuronal convolucional la cual es el estándar en los trabajos hoy en día y que por su desempeño se ha ido mejorando. Después se entrenó una red neuronal de tipo Visual Transformer (ViT), la cual representa un nuevo paradigma en la comunidad científica y cuyas posibilidades aún no se han explorado tanto debido a su reciente aparición en 2021. De esta manera se entrenaron seis redes

neuronales en total (3 CNN y 3 ViT), lo cual nos da un buen inicio para trabajar sobre los datos.

Para el entrenamiento se dividió el dataset en entrenamiento y validación en un ratio de 80:20 como se indica en el paper “preprocesamiento de bases de datos de imágenes para mejorar el rendimiento de redes neuronales convolucionales” (López Saca, n.d.) ya que entre los ratios 70:30 y 80:20 se obtienen los mejores resultados.

4.5.1 Entrenamiento CNN

Para el entrenamiento de los modelos convolucionales se tomó como referencia la arquitectura base propuesta por Delgadillo en el desarrollo de un vehículo autónomo, con el objetivo de mantener coherencia metodológica y facilitar la comparación de resultados. A partir de dicha arquitectura se realizaron diversas modificaciones orientadas a mejorar la capacidad de extracción de características visuales.

La red neuronal convolucional propuesta consta de tres capas convolucionales, en las cuales se incrementa progresivamente el número de filtros aplicados a las imágenes de entrada. Cada capa convolucional utiliza un tamaño de kernel de 3×3 , un desplazamiento (*stride*) de 2 y *padding* del tipo “same”, lo que permite reducir progresivamente la dimensionalidad espacial de las imágenes mientras se preserva la información relevante. Estas capas se aplican sobre los tres canales de entrada de las imágenes; en el caso del dataset HSV, dicha configuración se mantiene por compatibilidad estructural.

Posteriormente, cada capa convolucional es seguida por una capa de normalización por lotes (Batch Normalization), con el fin de estabilizar el entrenamiento y acelerar la convergencia del modelo, y por una función de activación no lineal del tipo ReLU. Para la agregación de las características extraídas, se emplea una capa de Global Average Pooling, lo cual permite reducir el número de parámetros y mitigar el sobreajuste.

La parte final del modelo está compuesta por una capa densa intermedia con 64 neuronas y activación ReLU. Finalmente, la capa de salida contiene dos neuronas sin función de activación, correspondientes a la predicción de la velocidad lineal y la velocidad angular del vehículo.

El entrenamiento del modelo se realiza utilizando el optimizador Adam, debido a su buen desempeño en tareas de regresión y visión por computadora, y una función de pérdida basada en el error cuadrático medio (Mean Squared Error, MSE). Como métricas de evaluación se emplean el MSE y el error absoluto medio (Mean Absolute Error, MAE). El dataset se divide en conjuntos de entrenamiento y validación siguiendo una proporción de 80:20, consistente con trabajos previos en la literatura.

Durante el proceso de entrenamiento se almacenan tanto el modelo final como las métricas y gráficas generadas, garantizando la trazabilidad y reproducibilidad de los experimentos realizados.

Pseudocódigo 8: Diseño de la red neuronal convolucional.

Algorithm 32: Entrenamiento de una CNN para Predicción de Velocidad
Input: Dataset final en CSV con rutas a imágenes procesadas y comandos de velocidad, imágenes preprocesadas, hiperparámetros de entrenamiento
Output: Modelo CNN entrenado, métricas y gráficas, modelo exportado en formato SavedModel
1 Procedimiento Principal;
2 Leer argumentos de ejecución: modo de preprocesamiento, número de épocas, tamaño de lote, tasa de aprendizaje;
3 Definir directorio raíz del proyecto;
4 Definir ruta del CSV del dataset final;
5 Definir directorio de imágenes procesadas según el modo;
6 Definir directorios de salida para modelos y gráficas;
7 Llamar a Subproceso 1: Configuración del Dataset;
8 Llamar a Subproceso 2: Construcción del Dataset en Memoria;
9 Llamar a Subproceso 3: Definición de la Arquitectura CNN;
10 Llamar a Subproceso 4: Entrenamiento del Modelo;
11 Llamar a Subproceso 5: Almacenamiento de Resultados;
Algorithm 33: Subproceso 1: Configuración del Dataset
Input: CSV del dataset final
Output: Conjunto de entrenamiento y validación
1 Definir tamaño de entrada de las imágenes;
2 Definir número de salidas del modelo (velocidad lineal y angular);
3 Leer CSV del dataset final;
4 Dividir dataset en conjunto de entrenamiento y conjunto de validación;

Algorithm 34: Subproceso 2: Construcción del Dataset en Memoria

Input: Conjunto de entrenamiento y validación

Output: Listas de imágenes de entrada y valores objetivo

```
1 Inicializar listas para imágenes de entrada y valores objetivo;
2 foreach registro en conjunto de entrenamiento do
3   Construir ruta de la imagen procesada;
4   if imagen no existe then
5     Omitir registro;
6   Leer imagen desde disco;
7   Redimensionar imagen al tamaño definido;
8   Normalizar valores de píxeles al rango [0, 1];
9   Almacenar imagen en lista de entradas;
10  Extraer comando de velocidad y almacenar como salida;
11 Repetir el proceso para conjunto de validación;
```

Algorithm 35: Subproceso 3: Definición de la Arquitectura CNN

Input: Listas de imágenes y valores objetivo

Output: Modelo compilado

```
1 Definir red neuronal convolucional secuencial con:
  • Capas convolucionales con incremento progresivo de filtros

  • Normalización por lotes

  • Funciones de activación no lineales

  • Capa de Global Average Pooling

  • Capas densas finales para regresión
```

Compilar modelo con:

- Optimizador Adam
 - Función de pérdida: error cuadrático medio
 - Métrica: error absoluto medio
-

Algorithm 36: Subproceso 4: Entrenamiento del Modelo

Input: Modelo compilado, conjuntos de entrenamiento y validación, hiperparámetros

Output: Modelo entrenado, historial de entrenamiento

```
1 Entrenar modelo con:
  • Conjunto de entrenamiento

  • Conjunto de validación

  • Número de épocas definido

  • Tamaño de lote especificado
```

Almacenar historial de entrenamiento;

Algorithm 37: Subproceso 5: Almacenamiento de Resultados

Input: Modelo entrenado, historial de entrenamiento

Output: Métricas, gráficas y modelo exportado

```
1 Guardar métricas del entrenamiento en disco;
2 Generar y almacenar gráficas de desempeño;
3 Exportar modelo en formato SavedModel;
4 Mostrar mensajes informativos indicando finalización del proceso;
```

Para la CNN se observó que tanto para RGB, Sobel y HSV encontró:

- 102,530 parámetros totales
- 102,082 parámetros entrenables
- 448 parámetros no entrenables

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	896
batch_normalization (BatchNormalization)	(None, 64, 64, 32)	128
re_lu (ReLU)	(None, 64, 64, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
re_lu_1 (ReLU)	(None, 32, 32, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
re_lu_2 (ReLU)	(None, 16, 16, 128)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 2)	130
Total params: 102,530 (400.51 KB)		
Trainable params: 102,082 (398.76 KB)		
Non-trainable params: 448 (1.75 KB)		

4.5.2 Entrenamiento ViT

En cuanto a la red visual transformer se optó por diseñar una en base a la información ofrecida en el paper *An image is Worth 16x16 Words* (Dosovitskiy et al., 2021) y de recursos digitales vistos en *Youtube* (DeepFindr, 20023), adaptándola a

las características del problema abordado y al tamaño del conjunto de datos disponible. El objetivo principal de esta implementación es evaluar el desempeño de un modelo basado en atención frente a una red convolucional tradicional bajo las mismas condiciones experimentales.

El algoritmo describe el flujo completo del entrenamiento del modelo, iniciando con la configuración de los hiper parámetros y la selección del tipo de preprocesamiento aplicado a las imágenes de entrada (RGB, Sobel o HSV). Esta elección permite analizar el impacto del tipo de representación visual sobre el desempeño del modelo *Transformer*.

En primer lugar, el dataset es cargado desde el archivo CSV y dividido en conjuntos de entrenamiento y validación utilizando una proporción de 80:20. Posteriormente, las imágenes preprocesadas son cargadas en memoria, redimensionadas y normalizadas, mientras que los valores objetivo corresponden a los comandos de velocidad lineal y angular asociados a cada imagen. Este procedimiento es equivalente al utilizado en el entrenamiento de la CNN, garantizando condiciones comparables entre ambos enfoques para lograr una comparación. La razón principal para mantenerlo en el algoritmo si ya se realizó previamente fue en caso de que no se aplicase un preprocesamiento anteriormente se pueda usar para pruebas de data sets individuales.

Para la arquitectura del modelo, se optó por dividir las imágenes de entrada en parches de tamaño fijo de 16x16 debido a que permite mantener la complejidad del Transformer manejable y siendo el estándar propuesto por Dosovitskiy, los cuales son proyectados a un espacio de *embeddings* de dimensión 128 para codificar la información visual mientras se capturan variaciones espaciales y de textura manteniendo la coherencia con la imagen. Esta decisión permite transformar la imagen en una secuencia de representaciones, facilitando su procesamiento mediante capas *Transformer*. La inclusión de *embeddings* posicionales responde a la necesidad de preservar la información espacial de los parches dentro de la imagen original.

El modelo está compuesto por múltiples capas codificadoras *Transformer*, cada una de ellas formada por mecanismos de atención *multi-head* y redes *feed-forward*. El número de capas (4), la dimensión de los embeddings y la cantidad de cabezas de atención (4) fueron seleccionados buscando un equilibrio entre capacidad de representación y complejidad computacional, considerando el tamaño del dataset y los recursos disponibles.

Una vez procesada la secuencia de embeddings, se aplica una operación de Global Average Pooling con el fin de obtener una representación global de la imagen. Esta elección reduce el número de parámetros y evita el uso de tokens adicionales, simplificando la arquitectura sin perder información relevante. Finalmente, se utilizan capas densas para realizar la regresión continua de las dos variables de salida correspondientes a la velocidad lineal y la velocidad angular.

El entrenamiento del modelo se lleva a cabo utilizando el optimizador Adam y una función de pérdida basada en el error cuadrático medio (MSE), manteniendo consistencia con el entrenamiento de la red convolucional. Durante el proceso se registran las métricas de desempeño y se generan las gráficas correspondientes, las cuales son almacenadas junto con el modelo entrenado para asegurar la trazabilidad y reproducibilidad de los experimentos realizados.

Pseudocódigo 9: Visual Transformer diseño.

Algorithm 38: Entrenamiento de un Visual Transformer
Input: Dataset final en CSV, imágenes preprocesadas, parámetros de entrenamiento
Output: Modelo Visual Transformer entrenado y métricas asociadas
1 Procedimiento Principal;
2 Definir argumentos de entrada del programa;
3 Definir directorio raíz del proyecto;
4 Definir ruta del CSV del dataset final;
5 Definir directorio de imágenes preprocesadas según el modo;
6 Definir directorio de almacenamiento del modelo;
7 Definir directorio de almacenamiento de gráficas;
8 Crear directorios de salida si no existen;
9 Definir tamaño de imagen y número de salidas del modelo;
10 Llamar a Subproceso 1: Carga y Preparación del Dataset;
11 Llamar a Subproceso 2: Construcción del Modelo Visual Transformer;
12 Llamar a Subproceso 3: Entrenamiento del Modelo;
13 Llamar a Subproceso 4: Almacenamiento de Resultados;

Algorithm 39: Subproceso 1: Carga y Preparación del Dataset

Input: CSV del dataset final, directorio de imágenes

Output: Datos de entrenamiento y validación

```
1 Leer CSV del dataset final;
2 Dividir dataset en entrenamiento y validación;
3 foreach subconjunto de datos do
4   foreach muestra del dataset do
5     Cargar imagen desde disco;
6     Normalizar valores de píxeles;
7     Extraer comando de velocidad lineal y angular;
8     Almacenar imagen y etiqueta;
```

Algorithm 40: Subproceso 2: Construcción del Modelo Visual Transformer

Input: Dimensiones de entrada y tasa de aprendizaje

Output: Modelo Visual Transformer compilado

```
1 Definir tamaño de parches;
2 Definir dimensión de embeddings;
3 Definir número de cabezas de atención;
4 Definir dimensión de red feed-forward;
5 Definir número de capas Transformer;
6 Dividir imagen en parches;
7 Proyectar parches a espacio de embeddings;
8 Agregar embeddings posicionales;
9 for número de capas Transformer do
10   Aplicar atención multi-cabeza;
11   Aplicar red feed-forward;
12   Aplicar normalización y dropout;
13 Aplicar pooling global;
14 Agregar capas densas finales;
15 Compilar modelo con optimizador Adam y pérdida MSE;
```

Algorithm 41: Subproceso 3: Entrenamiento del Modelo

Input: Modelo, datos de entrenamiento y validación

Output: Historial de entrenamiento

```
1 Entrenar modelo con datos de entrenamiento;
2 Validar modelo con datos de validación;
3 Almacenar historial de entrenamiento;
```

Algorithm 42: Subproceso 4: Almacenamiento de Resultados

Input: Modelo entrenado e historial

Output: Modelo exportado y métricas almacenadas

```
1 Guardar métricas de entrenamiento;
2 Generar y guardar gráficas de desempeño;
3 Exportar modelo entrenado en formato SavedModel;
4 Indicar finalización del entrenamiento;
```

4.6 Comparación

Finalmente se comparan los resultados obtenidos. Las métricas obtenidas medidas principalmente son de orden numérico debido a que el entrenamiento es un problema de regresión. Todos los datasets se entrenaron por 20 épocas (épocas) en ambas arquitecturas para comparación equitativa.

Se evaluaron con Mean Absolute Error (Error absoluto Medio o MAE por sus siglas en inglés) que mide en promedio cuánto se equivoca el modelo y *loss* (pérdida) a través del Mean Square Error (Error Cuadrático Medio o MSE por sus siglas en inglés) mide el promedio del cuadrado de los errores. A continuación se presentan los resultados de MAE para ambas arquitecturas:

CNN

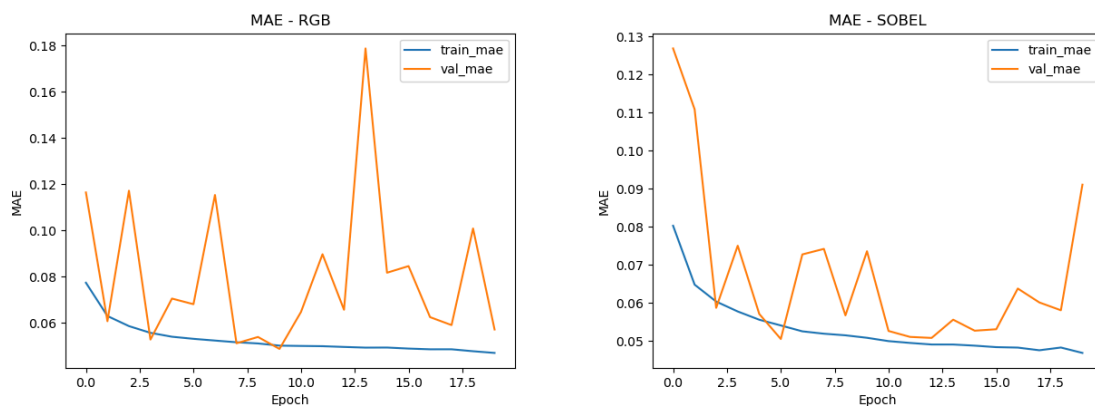
- hsv: 0.056
- rgb: 0.057
- sobel: 0.091

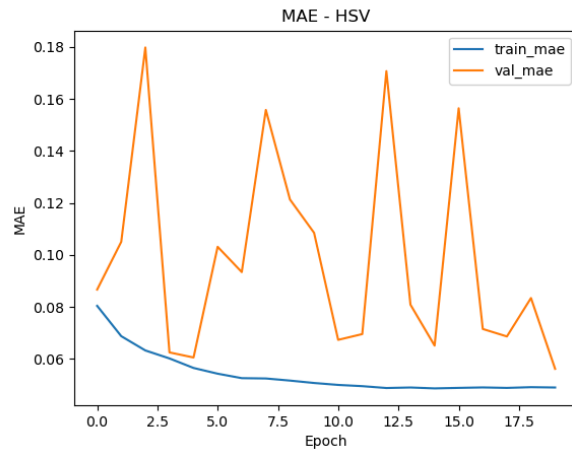
ViT

- vit_hsv: 0.157
- vit_rgb: 0.156
- vit_sobel: 0.158

En la CNN los entrenamientos con las técnicas de preprocesamiento HSV y RGB funcionan muy similar, con ligera ventaja para HSV. Mientras tanto las imágenes con filtro Sobel tienen un desempeño peor, con más ruido en la validación. Esto indica que los bordes simples no contienen suficiente información para la CNN, que depende de texturas y gradientes internos de la imagen.

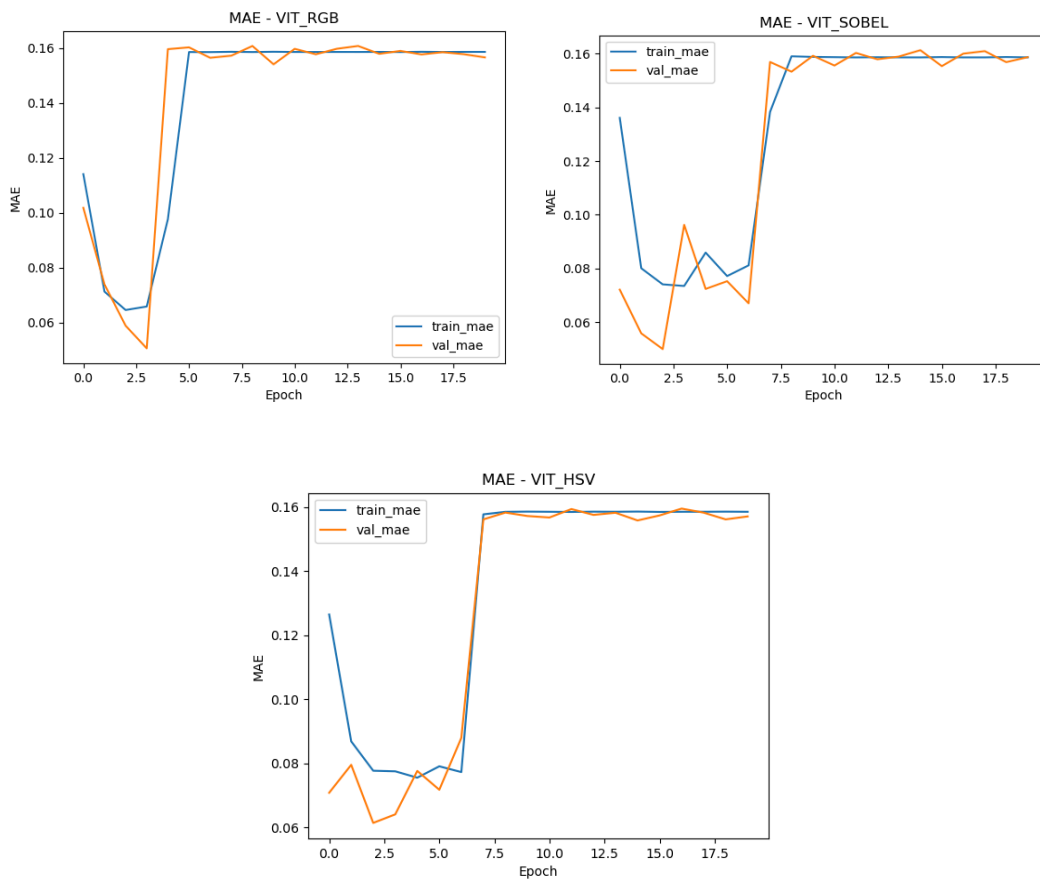
MAE de los modelos CNN





Para la arquitectura ViT el entrenamiento no mejora con HSV frente a RGB de manera significativa (un valor de milésima menos). Observamos un MAE más alto comparado con CNN (0.156 - 0.158 vs 0.056 - 0.057), lo que sugiere que con el tamaño del dataset y número de capas seleccionado, la ViT aún no logra generalizar tan bien como la CNN.

MAE de los modelos ViT



Aquí podemos ver que la arquitectura CNN es más robusta para generalizar que ViT. Siendo ViT más sensible a la cantidad de datos y al tamaño del *embedding*. Las diferencias de preprocesamiento casi desaparecen, probablemente porque la atención global permite a la red capturar información relevante incluso si hay menos textura como es el caso del filtro Sobel.

En cuanto a la pérdida (loss) observamos lo siguiente:

CNN

- hsv: 0.015
- rgb: 0.0157
- sobel: 0.026

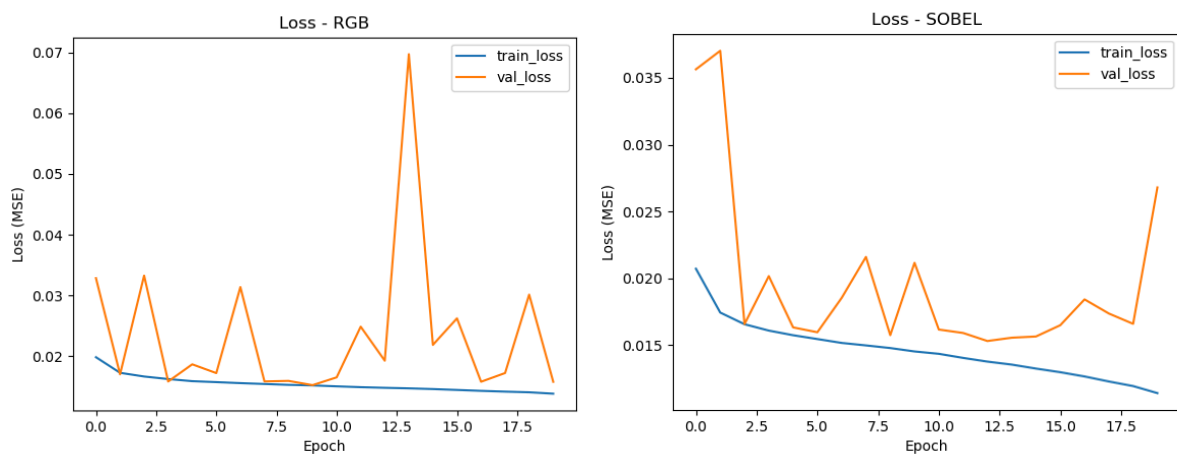
ViT

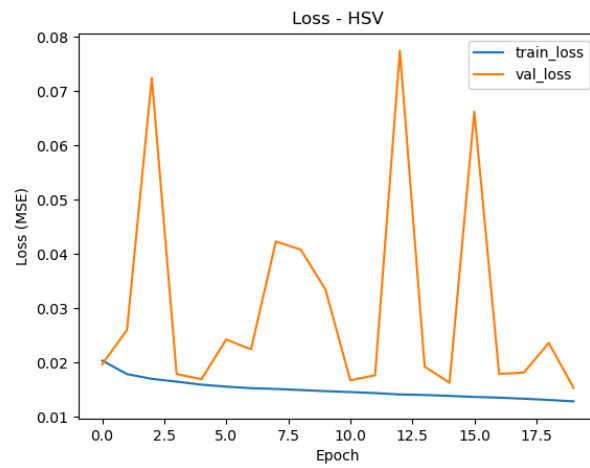
- hsv: 0.044
- rgb: 0.044
- sobel: 0.044

En el caso de la arquitectura CNN, los valores de pérdida confirman el comportamiento observado con el MAE, donde el preprocesamiento HSV y RGB presentan un error significativamente menor en comparación con Sobel.

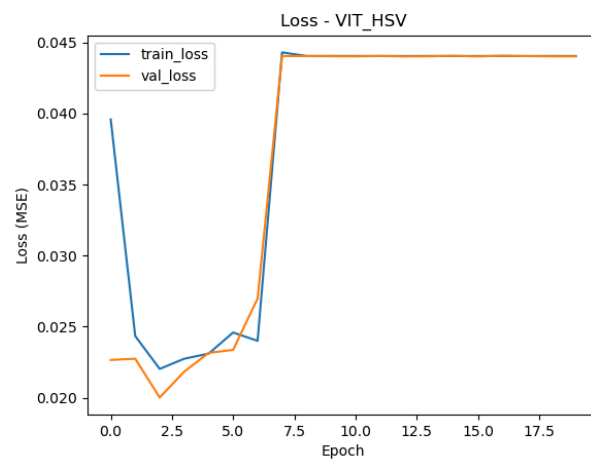
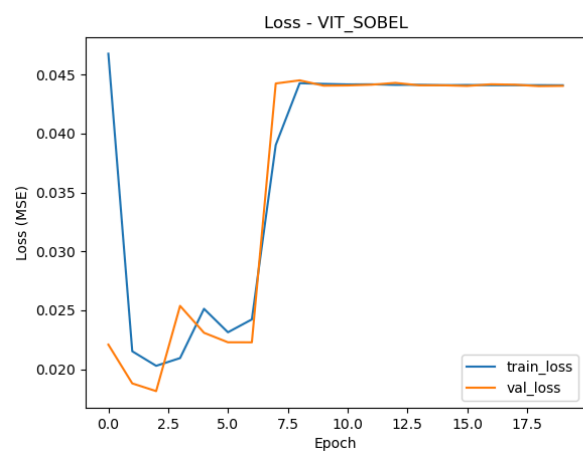
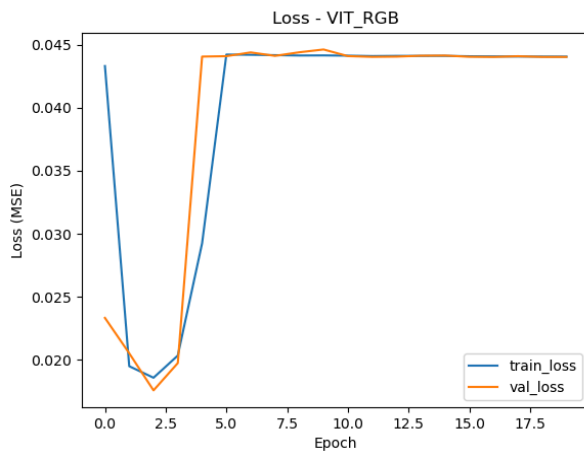
Para la arquitectura ViT, la pérdida de validación se mantiene prácticamente constante con una diferencia mínima independientemente del tipo de preprocesamiento aplicado.

MSE = Loss CNN





MSE = Loss ViT



Esto sugiere que el modelo Visual Transformer no logra beneficiarse de las diferencias en la representación visual de los datos y presenta un error más alto en comparación con la CNN. Este comportamiento puede atribuirse a la mayor complejidad del modelo y a su sensibilidad al tamaño del dataset disponible, lo que limita su capacidad de generalización en este escenario experimental.

Capítulo 5: Conclusiones

Se puede observar que durante estos experimentos el desempeño de las arquitecturas sí varían según el tipo de dataset. También podemos observar que el desempeño que según la arquitectura tiene ciertos patrones específicos, ya que la CNN tuvo un mejor desempeño en para reducir el MSE. Por otro lado, la arquitectura ViT tiene un patrón similar y overfitting en los tres casos.

La arquitectura CNN el patrón se marca con RGB y HSV vs Sobel donde en las últimas épocas tuvo un repunte en cuanto a desempeño. Quizá el correr más pruebas con más épocas ayude a desarrollar mejores conclusiones de si es solo una variación temporal o es un patrón. Sin embargo, con la información presentada podemos extraer conclusiones.

En cuanto al MAE nuevamente la arquitectura CNN tuvo un mejor desempeño que la red ViT, en general obteniendo resultados sin overfitting. En cuanto a la arquitectura ViT los resultados en cuanto al desempeño también marca un patrón. Parece existir una relación entre el preprocesamiento cuando es en el campo RGB vs Sobel y HSV (escala de grises). Ya que RGB obtuvo overfitting antes que Sobel y HSV específicamente el doble de rápido (epoch 2.5 vs 5).

Esto sugiere que el modelo Visual Transformer no logra beneficiarse de las diferencias en la representación visual de los datos y presenta un error más alto en comparación con la CNN. Este comportamiento puede atribuirse a la mayor complejidad del modelo y a su sensibilidad al tamaño del dataset disponible, lo que limita su capacidad de generalización en este escenario experimental.

¿Qué significa los datos obtenidos? Debemos dividir esta respuesta en dos.

En cuanto al desempeño de la red CNN vs ViT porque la CNN se desempeña mejor si la arquitectura ViT tiene capacidad de obtener contexto de toda la imagen sobre el contexto local de la CNN. Una de las posibles respuestas es que 50,000 instancias

no hayan sido suficientes datos. Esto afectará el desempeño de la red ViT ya que no podría abstraer características de manera eficiente como lo hizo la CNN.

Por otro lado, el ver que existen en una el desempeño agrupa los resultados de distinta manera, nos da la señal de que efectivamente el preprocesamiento de imágenes juega un rol importante en el desempeño de las arquitecturas, unas siendo más beneficiadas sobre otras. Con ello la búsqueda de nuevas configuraciones de preprocesamiento puede abrir las puertas a mejorar el desempeño de las redes para la conducción autónoma.

5.1 Propuesta de arquitectura

En base a los resultados obtenidos se conoce que las CNN pueden extraer de manera más eficiente características que las redes ViT cuando los datos parecen no ser lo suficientemente amplios. Por eso mismo la propuesta a aplicar en trabajo futuro sería dejar la extracción de características a la red CNN y las capas de toma de decisiones a la red ViT con una red híbrida como resultado, sería un buen inicio a explorar. Además se puede agregar una capa de Dropout con una tasa de 0.5 para regularización y ver si los resultados mejoran o no.

CNN (extracción de características) -> ViT (capas de decisión) = ¿mejora?

5.2 Trabajo futuro

El desarrollo de este proyecto de tesis ha sido de gran influencia no solo para aprender y entender de manera profunda el funcionamiento de las tecnologías de IA para la visión computacional. De igual manera ha servido para buscar nuevas combinaciones y mejoras a las técnicas aplicadas aquí para el desarrollo mejora en el desempeño de las redes.

Investigar aplicar estos entrenamientos al vehículo y medir su desempeño en el mundo real es el siguiente paso a desarrollar, avance que se llevará a cabo en el laboratorio LKE para futuras mejoras y avance en estas técnicas. Áreas de interés, será ver si algoritmos como transformada de Hough y la recopilación de más datos

para probar entrenamientos más profundos son las siguientes tareas a desarrollar para continuar con la aplicación de una red híbrida que nos permita profundizar más en el tema de las redes artificiales aplicadas a la conducción autónoma.

Referencias

- DeepFindr (4 jul 2023), Vision Transformer Quick Guide - Theory and Code[Video]. YouTube. <https://www.youtube.com/watch?v=j3VNqtJUoz0&t=26s>
- *Human-Centered Artificial Intelligence - Stanford University*. (n.d.). Artificial Intelligence Definitions. Retrieved Junio 17, 2025, from <https://hai-production.s3.amazonaws.com/files/2020-09/AI-Definitions-HAI.pdf>
- National Aeronautics and Space Administration. (2024, May 13). *What is Artificial Intelligence?* NASA. Retrieved Junio 17, 2025, from <https://www.nasa.gov/what-is-artificial-intelligence/>
- Cole, R. (2025, December 29). autonomous vehicle. Encyclopedia Britannica. <https://www.britannica.com/technology/autonomous-vehicle>
- Buckner, C., & Garson, J. (1997, May 18). *Connectionism (Stanford Encyclopedia of Philosophy)*. Stanford Encyclopedia of Philosophy. Retrieved Agosto 19, 2026, from <https://plato.stanford.edu/entries/connectionism/>
- Dosovitskiy, A., Beyer, L., & Kolesnikov, A. (2021, Octubre 22). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arxiv*. <https://doi.org/10.48550/arXiv.2010.11929>
- Fernandes, S., Duseja, D., & Muthalagu, R. (2021, Enero 20). Application of Image Processing Techniques for Autonomous Cars. *Proceedings of Engineering and Technology Innovation*, 17. <https://doi.org/10.46604/peti.2021.6074>
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. <https://www.deeplearningbook.org/>

- Haykin, S. (n.d.). *Neural Networks and Learning Machines* (Third edition ed.). Pearson.

- Heylighen, F., & Joslyn, C. (2001). Cybernetics and Second-Order Cybernetics. *Encyclopedia of Physical Science & Technology*.

- *Human-Centered Artificial Intelligence - Stanford University*. (n.d.). Artificial Intelligence Definitions. Retrieved Junio 17, 2025, from <https://hai-production.s3.amazonaws.com/files/2020-09/AI-Definitions-HAI.pdf>

- IBM. (n.d.). *¿Qué es LiDAR?* Retrieved Noviembre 07, 2025, from <https://www.ibm.com/mx-es/think/topics/lidar>

- Jang, S. (2025, Julio 06). *Impact of Data Quality on CNN-Based Sewer Defect Detection*. MDPI. Retrieved Septiembre 20, 2025, from <https://www.mdpi.com/2073-4441/17/13/2028>

- KIA. (n.d.). *¿Son los vehículos autónomos el futuro?* ¿Son los vehículos autónomos el futuro? <https://www.kia.com/pr/discover-kia/ask/Are-self-driving-cars-the-future.html>

- Lecun, Y., Bengio, Y., & Hinton, G. (2015, Mayo 27). Deep learning. *Nature*, 521((7553)), 436-444. <https://www.nature.com/articles/nature14539>

- Lopez Saca, F. (n.d.). *Preprocesamiento de bases de datos de imágenes para mejorar el rendimiento de redes neuronales convolucionales* [Artículo científico]. Research in Computing Science. https://rcs.cic.ipn.mx/RCS_editorial.html

- Luca, A. R., Ursuleanu, T. F., Gheorghe, L., & Grigorovici, R. (2022, Marzo 9). Impact of quality, type and volume of data used by deep learning models in the analysis of medical images. *Elsevier*, 29. <https://doi.org/10.1016/j.imu.2022.100911>

- Microblink. (n.d.). *Image Pre-processing*. Microblink. Retrieved Septiembre 09, 2025, from <https://microblink.com/resources/glossary/image-pre-processing/>

- Molinero Diez, G. (n.d.). *SEGMENTACIÓN DE IMÁGENES EN COLOR BASADA EN EL CRECIMIENTO DE REGIONES*. e-REdING.
<https://biblus.us.es/bibing/proyectos/abreproy/11875/fichero/Proyecto+Fin+de+Carrera%252F3.Espacios+de+color.pdf>

- National Aeronautics and Space Administration. (2024, May 13). *What is Artificial Intelligence?* NASA. Retrieved Junio 17, 2025, from <https://www.nasa.gov/what-is-artificial-intelligence/>

- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *APA PsycNet*, 65(6), 386.
<https://doi.org/10.1037/h0042519>

- Science Direct. (n.d.). *Preprocessing Technique*. Retrieved Septiembre 21, 2025, from <https://www.sciencedirect.com/topics/computer-science/preprocessing-technique>

- Sonka, M., Hlavac, V., & Boyle DPhil, R. (n.d.). Image pre-processing. In: Image Processing, Analysis and Machine Vision. *Springer Nature Link*, 56–111.
https://doi.org/10.1007/978-1-4899-3216-7_4

- Springer. (2019, Mayo 01). *McCulloch–Pitts Neural Network Model*. Springer Nature Link. Retrieved Agosto 25, 2025, from https://link.springer.com/chapter/10.1007/978-981-13-7430-2_11# citas
- Turing, A. M. (1950, October). COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236), 433-460. <https://doi.org/10.1093/mind/LIX.236.433>
- Universidad de Sevilla. (n.d.). *Procesamiento en el dominio de la Frecuencia*. <https://asignatura.us.es/imagendigital/Tema3-Frecuencia.pdf>
- Visual LED. (n.d.). *El contraste en pantallas led - Visual Led*. Pantallas LED. Retrieved Octubre 02, 2025, from <https://visualled.com/glosario/contraste-pantallas/>
- Viswanathan, V., & Hussein, R. (2017, Abril 01). Applications of Image Processing and Real-Time Embedded Systems in Autonomous Cars: A Short Review. *International Journal of Image Processing*, 11, 35-49. <https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJIP-1114>
- *What Is a Convolutional Neural Network? - MATLAB & Simulink*. (n.d.). MathWorks. Retrieved Septiembre 02, 2025, from <https://www.mathworks.com/discovery/convolutional-neural-network.html>
- Saiz Allende, A. (2021, Marzo). Implementación de redes neuronales para conducción autónoma. <https://repositorio.unican.es/xmlui/bitstream/handle/10902/21899/434502.pdf?sequence=1&isAllowed=y>

- DEVIDAS GORE, C. (2022). *Optimization of the Image Pre processing Pipeline for Autonomous Driving*.
<https://www.diva-portal.org/smash/get/diva2:1737865/FULLTEXT01.pdf>

- Salvi, M. (2021, Enero). The impact of pre- and post-image processing techniques on deep learning frameworks: A comprehensive review for digital pathology image analysis. *Science Direct*, 128.
<https://www.sciencedirect.com/science/article/pii/S0010482520304601>

- Fernandes, S., Duseja, D., & Muthalagu, R. (2020, Diciembre). *Application of Image Processing Techniques for Autonomous Cars*.
https://www.researchgate.net/publication/347938036_Application_of_Image_Processing_Techniques_for_Autonomous_Cars

- Raghuvanshi, S. (2024, January 6). *The Importance of Activation Functions in Neural Networks* | by Suryansh Raghuvanshi | Medium. Suryansh Raghuvanshi. Retrieved January 20, 2026, from
<https://suryansh-raghuvanshi.medium.com/activation-functions-in-neural-networks-45a8b035b14a>

- Tanviruzzama and S. Mehfuz, "Review: Lane Detection for Autonomous Vehicles Using Image Processing Techniques," 2023 International Conference on Power, Instrumentation, Energy and Control (PIECON), Aligarh, India, 2023, pp. 1-6, doi: 10.1109/PIECON56912.2023.10085756.

- P. Liu, C. Zhang, H. Qi, G. Wang and H. Zheng, "Multi-Attention DenseNet: A Scattering Medium Imaging Optimization Framework for Visual Data Pre-Processing of Autonomous Driving Systems," in IEEE Transactions on Intelligent Transportation

Systems, vol. 23, no. 12, pp. 25396-25407, Dec. 2022, doi:
10.1109/TITS.2022.3145815.

- Tanviruzzama and S. Mehfuz, "Review: Lane Detection for Autonomous Vehicles Using Image Processing Techniques," *2023 International Conference on Power, Instrumentation, Energy and Control (PIECON)*, Aligarh, India, 2023, pp. 1-6, doi: 10.1109/PIECON56912.2023.10085756.
- Bengio, Y., Courville, A., & Vincent, P. (2012, 24 junio). Representation Learning: A Review and New Perspectives. arXiv.org. <https://arxiv.org/abs/1206.5538v3>
- Alexander Amini. (2025, 3 marzo). MIT Introduction to Deep Learning | 6.S191 [Video]. YouTube. <https://www.youtube.com/watch?v=alfdI7S6wCY>
- Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66), eabm6074. <https://doi.org/10.1126/scirobotics.abm6074>