# Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

**Maestría en Sistemas Computacionales**



# Sistema inteligente de bajo costo para la gestión de estacionamientos: Detección y clasificación de lugares disponibles mediante Inteligencia Artificial

**TRABAJO RECEPCIONAL PARA LA OBTENCIÓN DEL GRADO DE MAESTRÍA EN SISTEMAS COMPUTACIONALES**


Presenta:

**ING. INGRID SHAIANY CHAN TOPETE**

Asesor: **MS. VÍCTOR HUGO MARTÍNEZ SÁNCHEZ**



Tlaquepaque, Jalisco. Julio 2025

# Agradecimientos

Agradezco a Dios por concederme la entereza necesaria para perseverar, por acompañarme en cada paso y sostenerme en los momentos difíciles. Toda la honra y gloria a Dios por su misericordia, su verdad, así como por ser mi ayuda y escudo, pues el creó todas las cosas y por su voluntad existen y fueron creadas.

A mi mamá, gracias por ser mi mayor apoyo, por motivarme y confiar siempre en mí, recordándome que no existen límites cuando hay determinación. Tu amor transforma mi vida todos los días y me impulsa a luchar por mis sueños.

A mi familia, con especial cariño a mi hermana, papá, abuela y tía, por estar siempre a mi lado y ofrecerme siempre su apoyo sin condiciones, apoyar mis metas y celebrar conmigo cada logro. Los amo con todo mi corazón.

A mis amigos de la maestría, gracias por compartir este proceso conmigo, por su compañerismo, así como por hacer más enriquecedores estos años de estudio. Su amistad, apoyo y cariño hicieron que disfrutara plenamente esta etapa.

A mis profesores, gracias por despertar en mí la pasión por la tecnología, por ser ejemplo de dedicación y por compartir sus conocimientos, que contribuyeron de forma significativa a este trabajo y a mi formación profesional. Es un honor para mí haber tenido la oportunidad de aprender de ustedes.

A Víctor Hugo Martínez Sánchez, mi director de tesis, gracias por su dirección y apoyo a lo largo de este trabajo. Le agradezco su confianza depositada en mí, su forma de guiarme y de transferirme su conocimiento.

Al ITESO, por brindarme el espacio, los recursos y la beca que hicieron posible cursar esta maestría y desarrollar esta investigación. Gracias por su calidad humana y excelencia educativa.

Finalmente, gracias a cada persona que, directa o indirectamente, hizo posible la culminación de este trabajo, y a mí misma, por la determinación, la fe y la perseverancia para alcanzar esta meta.

# Dedicatoria

Dedico esta tesis a Dios, quien ha sido mi fortaleza, mi guía y mi sostén durante cada paso de este proceso. Agradezco profundamente la sabiduría, la paciencia y la determinación que me concedió para alcanzar la meta del presente trabajo.

A mi familia, fuente de amor y fortaleza constante, por su entrega y por estar presentes en cada momento importante. De forma especial, a mis padres, por brindarme un soporte firme e inquebrantable, y a mi hermana, por motivarme siempre y confiar plenamente en mí.

Finalmente, a todas aquellas personas que, de alguna forma, han aportado a mi desarrollo académico y personal. Gracias a su conocimiento, consejos y apoyo que posibilitaron mi desarrollo y la finalización exitosa de este proyecto.

# Resumen

El presente documento desarrolla una solución orientada a la identificación y clasificación de espacios de estacionamiento utilizando imágenes capturadas por dispositivos de vigilancia. Para esto, se emplearon metodologías de Inteligencia Artificial (IA), específicamente de Aprendizaje Automático (Machine Learning, ML) y Aprendizaje Profundo (abreviado como DL en inglés: Deep Learning) aplicados a Visión Computarizada (CV, por sus siglas en inglés de Computer Vision). El objetivo central radica en automatizar la identificación de espacios de estacionamiento ocupados o vacíos mediante el análisis de imágenes tomadas por diferentes cámaras en diferentes condiciones climáticas y desde diversas perspectivas.

Para ello, se combinaron dos enfoques complementarios. En una primera etapa, se emplearon modelos de ML como apoyo para realizar una clasificación básica de las imágenes, facilitando la identificación inicial de espacios de estacionamiento. Sin embargo, la propuesta final se basó en la integración de un modelo de DL, específicamente YOLOv8 (del inglés You Only Look Once), que actúa como núcleo del sistema con el fin de detectar espacios de estacionamiento de manera exacta y clasificarlos eficazmente. Para optimizar su desempeño, se realizaron múltiples experimentos, probando tanto la versión YOLOv8n (nano) como YOLOv8s (small), combinadas con técnicas de aprendizaje por transferencia (transfer learning, TL) y aumento de datos (data augmentation) como rotaciones, escalados, traslaciones, volteos y variaciones de iluminación. Asimismo, se evaluaron diferentes configuraciones de entrenamiento, variando el número de épocas y los parámetros, siempre guiándose por las métricas de validación para alcanzar el mejor resultado posible. Los experimentos se ejecutaron inicialmente de forma local en equipo propio y posteriormente en Google Colab o servicios en la nube, lo que permitió acelerar el proceso de entrenamiento y realizar pruebas más extensas de manera eficiente. Cabe destacar que el dataset final utilizado fue el resultado de una combinación de tres conjuntos de datos distintos, integrados para enriquecer la variedad de escenarios y mejorar la capacidad de generalización del modelo.

La selección de YOLOv8 se fundamenta en su capacidad para realizar la identificación y categorización de múltiples elementos de manera simultánea, convirtiéndolo como una alternativa robusta y rápida, ideal para soluciones en tiempo real como la gestión de plazas de aparcamiento. Para esta solución era indispensable contar con un modelo ligero que pudiera operar con baja latencia y en hardware con recursos limitados, características que hacen a YOLOv8 especialmente adecuado. En contraste, los modelos basados en Transformers, aunque potentes para diversas tareas, no resultaban apropiados para este caso debido a su elevada complejidad computacional, altos requerimientos de memoria y tiempos de procesamiento más largos, lo que dificulta su uso en sistemas en tiempo real y con restricciones de recursos. Además, a diferencia de otras arquitecturas de Deep Learning que requieren procesos separados para detectar y clasificar objetos, YOLOv8 integra ambas tareas en un solo paso, reduciendo la latencia y mejorando la velocidad de respuesta. Esta eficiencia, junto con su alta precisión y estabilidad frente a variaciones en iluminación, ángulos de cámara y condiciones climáticas, hacen de YOLOv8 la opción más adecuada para sistemas de CV en entornos dinámicos y con restricciones de recursos como los estacionamientos.

Para desarrollar este proyecto se decidió usar la versión más reciente y estable de YOLO, la YOLOv8, lanzada en enero de 2023. Esta elección se fundamentó en las mejoras significativas que ofrece respecto

a versiones anteriores, como YOLOv7, especialmente en precisión, velocidad y eficiencia para detección de objetos en tiempo real. Además, YOLOv8 ha recibido actualizaciones continuas que han optimizado su desempeño, consolidándola como una solución confiable y eficiente. Aunque en 2024 se anunció YOLOv9 con innovaciones técnicas interesantes, esta versión aún no ha alcanzado la estabilidad ni adopción generalizada, lo que limita su uso en proyectos prácticos. Por ello, YOLOv8 representó la opción más adecuada para este trabajo, garantizando un equilibrio óptimo entre rendimiento, eficiencia y confiabilidad.

Entre los beneficios más destacados de este modelo fue la disponibilidad de conjuntos de datos específicos y compatibles con su formato de entrenamiento, lo que facilitó la construcción de un dataset sólido adaptado a las necesidades del proyecto. Además, la estructura de YOLO permite ajustar y desplegar el sistema de forma práctica en infraestructuras existentes, utilizando cámaras convencionales y sin necesidad de costosos sensores físicos o instalaciones invasivas.

Gracias a estas características, YOLOv8 se posicionó como una herramienta flexible y escalable, capaz de adaptarse a distintas condiciones ambientales —como cambios de luz, clima y ángulos de visión—, manteniendo resultados consistentes y confiables. Este enfoque posibilita que el sistema se integre fácilmente a plataformas de gestión de estacionamientos y aplicaciones móviles para orientar a los usuarios hacia espacios libres, optimizando así el uso de la infraestructura disponible.

La importancia de este proyecto reside en que responde a un problema real que impacta directamente la movilidad urbana: la búsqueda de espacios de estacionamiento. Esta situación genera tráfico innecesario, aumenta el uso de combustible y eleva los niveles de contaminación ambiental. Al automatizar la detección de espacios vacíos mediante CV, se contribuye a optimizar la eficacia operativa de los estacionamientos, reduciendo periodos de búsqueda y apoyando prácticas sostenibles alineadas con los principios de movilidad y ciudades inteligentes.

En conjunto, la implementación de YOLOv8 permitió construir una solución de bajo costo, de rápida adopción y con potencial de expansión a distintos entornos, demostrando que la combinación de herramientas de ML y DL puede ofrecer respuestas prácticas y efectivas a problemas de la vida cotidiana.

**Palabras clave:** visión computarizada, yolov8, aprendizaje profundo, aprendizaje automático, detección de objetos.

# Tabla de contenido

# Lista de Figuras

# Lista de ecuaciones

# Lista de tablas

# Lista de acrónimos y abreviaturas

| Acrónimo | Definición en inglés | Definición en español |
|---|---|---|
| ANN | Artificial Neural Networks | Redes Neuronales Artificiales |
| API | Application Programming Interface | Interfaz de Programación de Aplicaciones |
| AWS | Amazon Web Services | Servicios Web de Amazon |
| CNN | Convolutional Neural Networks | Redes Neuronales Convolucionales |
| COCO | Common Objects in Context | Objetos Comunes en Contexto |
| CV | Computer Vision | Visión Computarizada |
| DL | Deep Learning | Aprendizaje Profundo |
| FaaS | Function as a Service | Función como servicio |
| FC | Fully Connected Layer | Capa Totalmente Conectada |
| FN | False Negatives | Falsos negativos |
| FPR | False Positive Rate | Tasa de Falsos positivos |
| GAN | Generative Adversarial Network | Red Generativa Antagónica |
| GPU | Graphics Processing Unit | Unidad de Procesamiento de Gráficos |
| IA | Artificial Intelligence | Inteligencia Artificial |
| INEGI | Instituto Nacional de Estadística y Geografía | Instituto Nacional de Estadística y Geografía |
| IoT | Internet of Things | Internet de las cosas |
| IoU | Intersection over Union | Intersección sobre Unión |
| IT | Information Technology | Tecnología de la Información |
| k-NN | k-Nearest Neighbors | k-Vecinos Más Cercanos |
| LiDAR | Light Detection and Ranging | Detección y Medición por Luz |
| LSTMs | Long Short-Term Memory | Memorias de Largo y Corto Plazo |
| Map | Mean Average Precision | Promedio de Precisión Media |
| ML | Machine Learning | Aprendizaje Automático |
| OCI | Oracle Cloud Infrastructure | Infraestructura de Nube de Oracle |
| OT | Operational Technology | Tecnología Operativa |
| RDMA | Remote Direct Memory Access | Acceso Remoto Directo a Memoria |
| ReLU | Rectified Linear Unit | Unidad Lineal Rectificada |
| ResNet | Residual Network | Red Residual |
| RL | Reinforcement Learning | Aprendizaje por refuerzo |
| RNN | Recurrent Neural Networks | Redes Neuronales Recurrentes |
| ROC | Receiver Operating Characteristic | Curva de Característica Operativa del Receptor |
| RoI | Region of Interest | Región de Interés |
| RPN | Region Proposal Network | Red de Propuesta de Regiones |
| SPI | Serial Peripheral Interface | Interfaz Periférica Serial |
| TL | Transfer Learning | Transferencia de Aprendizaje |
| TPR | True Positive Rate | Tasa de Verdaderos Positivos |
| VGG | Visual Geometry Group | Grupo de Geometría Visual |
| Wi-Fi | Wireless Fidelity | Wi-Fi |
| YOLO | You Only Look Once | Solo miras una vez |

# 1. INTRODUCCIÓN

Este capítulo expone el marco general de la investigación, abordando el crecimiento urbano y vehicular como factores que han agravado la problemática de disponibilidad de estacionamiento en zonas de alta afluencia. Se presentan antecedentes respaldados por datos estadísticos recientes que revelan la dimensión del problema, y se justifica la pertinencia de desarrollar una solución tecnológica basada en IA para mitigar sus efectos. Asimismo, se define el problema central a resolver, relacionado con la necesidad de identificar en tiempo real los espacios disponibles en estacionamientos mediante el análisis de imágenes. A partir de ello, se plantean los propósitos globales y específicos que rigen el desarrollo de esta iniciativa. Finalmente, se describe la propuesta de innovación tecnológica, basada en el uso de CV, ML y DL, destacando el uso de modelos como YOLOv8 para ofrecer una alternativa eficiente, automatizada y de bajo costo a los sistemas tradicionales.

La urbanización acelerada y el incremento en el número de vehículos ha generado desafíos significativos en el manejo de tráfico en estacionamientos, específicamente, en áreas de alta concurrencia como plazas comerciales, aeropuertos, escuelas u oficinas. Según la información proporcionada por el Instituto Nacional de Estadística y Geografía (INEGI), en el estado de Jalisco, el número de vehículos aumentó en un 67.37% desde 2010 hasta 2023, registrando un promedio anual de crecimiento del 4.0%. En 2023, el número total de automóviles alcanzó los 4 millones 609 mil 531, 5.49% más respecto al año anterior [1] . Durante los días con mayor tráfico, se estima que al menos el 10% de los clientes en centros comerciales abandonan el lugar, esto se atribuye a la falta de acceso de datos sobre la disponibilidad en un estacionamiento, en adición al tiempo excesivo que dedican a encontrar un espacio disponible [2].

El propósito de este trabajo de investigación radica en emplear métodos de IA con el fin de enfrentar el reto que representa la identificación en tiempo real de la disponibilidad de plazas destinados al parqueo, con la finalidad de disminuir el tiempo que los conductores emplean en la búsqueda. Para ello, se implementarán técnicas de CV combinadas con modelos de ML y DL. Uno de los objetivos principales es que el sistema sea capaz de leer directamente las imágenes provenientes de cámaras de seguridad instaladas en los estacionamientos, procesarlas en tiempo real y ofrecer resultados precisos tanto en la detección automática de los espacios como en la clasificación de su estado (ocupado o vacío).

En contraposición a los sistemas tradicionales que utilizan sensores espaciales, CV ofrece una alternativa más económica y de menor mantenimiento, ya que no requiere hardware adicional como sensores, baterías o sistemas de instalación [2] (véase Apéndice C: Estimación de costos para sistema de sensores SPI — Ejemplo con 275 lugares de estacionamiento). Para desarrollar este sistema, se emplearon librerías de CV en Python, junto con diversos enfoques de IA, incluyendo modelos clásicos de ML y técnicas de DL como las Redes Neuronales Convolucionales (*Convolutional Neural Networks*, CNN), conocidas por su capacidad para procesar imágenes y reconocer objetos de manera eficaz [3]. La decisión de investigar este tema se basa en la creciente necesidad de soluciones eficientes en la vida cotidiana. En Guadalajara y Zapopan, por ejemplo, existen 749 lugares que han requerido la categoría de estacionamiento [4]. La

búsqueda prolongada de estacionamiento no solo aumenta el estrés y disminuye la atención de los conductores, lo que incrementa el riesgo de accidentes, sino que también contribuye a la congestión vehicular, mayor consumo de combustible, contaminación y provocando un descenso en la satisfacción del usuario [5].

Para abordar el problema, se exploraron diferentes enfoques de ML. Inicialmente, se aplicaron modelos clásicos de ML como regresión logística, k-NN (K-Nearest Neighbors, en inglés) y árboles de decisión, enfocados en la clasificación del estado de los espacios (ocupado o vacío) a partir de características extraídas manualmente de las imágenes. Posteriormente, se utilizó el algoritmo K-Means con fines exploratorios, con el objetivo de identificar posibles agrupamientos naturales en los datos sin etiquetas, mediante técnicas de aprendizaje no supervisado.

Como solución más robusta y precisa, se implementó un modelo basado en DL utilizando YOLOv8 (You Only Look Once, versión 8), una CNN de última generación especializada en el monitoreo y detección de elementos en tiempo real. Este modelo permite no solo clasificar el estado de ocupación, sino también detectar automáticamente la ubicación de los sitios de estacionamiento dentro de la imagen, sin necesidad de una segmentación previa o etiquetado manual, haciendo viable su uso con videovigilancia en tiempo real.

La metodología empleada consta de una evaluación cuantitativa para mostrar el desempeño de cada modelo de IA considerando las curvas de entrenamiento y de validación, la exactitud (accuracy), sensibilidad (recall) y precisión (precision), incluyendo además métricas avanzadas como mAP y la matriz de confusión. Por otra parte, el presente documento está estructurado de manera que abarca un análisis integral del estado del arte y del marco teórico, seguido de una exposición detallada del desarrollo y la metodología empleada en la investigación. Asimismo, se incluyen los resultados obtenidos y las conclusiones derivadas, con énfasis en los puntos fuertes y las restricciones de cada enfoque, así como propuestas de trabajo futuro.

## 1.1. Antecedentes

La IA es un término frecuentemente utilizado en la última década, abarca una variedad de aplicaciones diseñadas para llevar a cabo procesos complejos que antes demandaban participación humana, por ejemplo, la detección de patrones y la solución de problemas. Esta tecnología ha demostrado ser capaz de analizar y dar sentido a datos en una escala inalcanzable manualmente [6]. Según una encuesta de McKinsey realizada en 2021, el porcentaje de empresas que reportaron haber adoptado IA en al menos una función aumentó al 56%, comparado con el 50% del año anterior [6]. Además, el 27% de los encuestados indicó que al menos el 5% de sus ingresos estaba relacionado con la IA [7].

ML es una ciencia que utiliza algoritmos para entrenar programas capaces de realizar tareas sin necesidad de instrucciones explícitas. Su objetivo principal es procesar volúmenes masivos de datos y desarrollar algoritmos que detecten patrones dentro de ellos, para luego generar predicciones precisas en situaciones nuevas o desconocidas. Por otra parte, DL constituye una extensión avanzada del ML, sustentada en arquitecturas algorítmicas conocidas como redes neuronales, las cuales toman como referencia la dinámica de funcionamiento del cerebro humano. Mediante estas redes, DL permite automatizar procesos de alta complejidad que, de forma convencional, demandarían la implicación del intelecto humano, tales como la descripción de imágenes, la transcripción de audio a texto o la traducción de documentos. No obstante, estas soluciones suelen demandar mayores recursos: conjuntos de datos extensos, infraestructura robusta y costos más elevados. Tanto ML como DL se integran dentro del ámbito de investigación de la IA [8].

En ML, los ingenieros experimentan con diferentes modelos para aplicaciones de CV ajustando parámetros y proporcionando datos hasta que los modelos alcancen el umbral de precisión esperado. Las tecnologías de infraestructura fundamentales para entrenar sistemas de IA a gran escala abarcan redes de clústeres, como por ejemplo el Acceso Remoto Directo a Memoria (RDMA, por sus siglas en inglés) e InfiniBand, así como recursos informáticos especializados como las Unidades de Procesamiento de Gráficos (GPU) y sistemas de almacenamiento de eficacia notable [7].

CV es una tecnología que facilita a las computadoras y sistemas la interpretación de datos provenientes de imágenes digitales, vídeos y otros datos visuales. Esta tecnología implementa redes neuronales y técnicas de procesamiento de imágenes para hacer recomendaciones o tomar decisiones basadas en la información visual. CV ha experimentado un desarrollo acelerado, ya que su precisión ha aumentado del 50% al 99% en la última década [8].

En el ámbito del CV, las CNN proporcionan una estrategia altamente escalable para la clasificación visual y la identificación de elementos dentro de las imágenes. Estas redes se basan en conceptos del álgebra lineal, como la operación de multiplicación de matrices, utilizada para detectar patrones en las imágenes. No obstante, la capacitación de estas redes puede ser computacionalmente rigurosa, necesitando GPU para su eficaz implementación. Principalmente, las CNN están formadas por la capa de agrupación, la capa convolucional, y la capa totalmente conectada (FC, por sus siglas en inglés) [3].

En la gestión de estacionamientos, los métodos tradicionales, como los sistemas de conteo simples o la señalización manual, han resultado insuficientes para manejar la creciente demanda. Desde la década del 2000, la incorporación de tecnologías de sensores ha buscado facilitar el proceso de búsqueda de

estacionamiento para los usuarios. Sin embargo, estos sistemas avanzados implican costos adicionales significativos en términos de hardware, instalación, así como mantenimiento [2] (véase Apéndice C: Estimación de costos para sistema de sensores SPI — Ejemplo con 275 lugares de estacionamiento). El desafío de optimizar la gestión de estacionamientos mediante tecnologías más eficientes y rentables resalta la relevancia de aplicar IA y CV para abordar estos problemas.

En primeras etapas se pueden utilizar técnicas de aprendizaje de ML, como el algoritmo KNN, para realizar tareas de clasificación básica, sin embargo, aunque estos enfoques ofrecen soluciones iniciales, presentaban limitaciones notables en entornos reales, además, los modelos de ML tradicional necesitan que las características sean seleccionadas de forma manual (feature engineering), lo que reduce su capacidad de generalización y los hace menos robustos ante cambios contextuales.

Con el avance del DL se ha revolucionado el campo de CV. Este enfoque, basado en CNNs, facilita que los modelos adquieran conocimiento automático de representaciones complejas que se generan mediante de los datos sin implicar acción humana para extraer características. Modelos como YOLO han mostrado un alto desempeño en procedimientos para la detección de elementos, dado que tienen la capacidad de localizar y clasificar en tiempo real múltiples instancias en una imagen. Estas capacidades los hacen especialmente adecuados para aplicaciones de monitoreo y control automatizado, como la detección de espacios de estacionamiento disponibles [9].

## 1.2. Justificación

Este proyecto de tesis propone la creación de un sistema de IA que emplea CNN para identificar espacios vacíos de estacionamientos locales. Este enfoque no solo representa una innovación significativa en la gestión de estacionamientos, sino que también responde a necesidades urgentes y crecientes en la sociedad moderna [1].

El aumento del mercado automotriz y, por lo tanto, el incremento significativo en la demanda de estacionamientos [1] es una oportunidad para ofrecer soluciones tecnológicas que permitan proveer un servicio de calidad, con una gestión enfocada en la eficiencia. Los métodos tradicionales, como la señalización manual y los sistemas de conteo básicos, han demostrado ser insuficientes para abordar de manera efectiva estos problemas. Las tecnologías emergentes, como los sensores avanzados, han mostrado potencial, pero implican altos costos de instalación y mantenimiento [2].

La aplicación de CNN para detectar espacios vacíos de estacionamiento se basa en su capacidad para la evaluación y el tratamiento de grandes cantidades de datos visuales con gran exactitud. De acuerdo con estudios recientes, las CNN resultan extremadamente útiles para labores de categorización e identificación de patrones en imágenes [8]. Esta tecnología posibilitaría la automatización del procedimiento de detección de áreas vacías, disminuiría la necesidad de intervención humana y aumentaría la optimización de los sistemas administrativos de estacionamientos.

La incorporación de modelos de IA basados en CNN se alinea con las tendencias actuales en CV y ML. Según un estudio de Statista, el mercado de CV está creciendo considerablemente, alcanzando los 26000 millones de dólares en el periodo de 2024, con un ritmo de expansión anual compuesto cercano al 12% [8]. Esta expansión refleja el avance continuo en la precisión y capacidad de las tecnologías de CV, que resultan esenciales para el objetivo de este proyecto.

Además, la integración de estas tecnologías en el ámbito de estacionamientos es una extensión lógica del uso creciente de la IA en diversas aplicaciones urbanas y comerciales. La habilidad de las CNN para examinar imágenes en tiempo real [3] facilita una administración más activa y eficaz de los lugares de aparcamiento, lo cual conduce a un incremento notable en la experiencia del usuario y una optimización de los recursos existentes.

El potencial de la solución propuesta comprende una gran variedad de sectores, que van desde centros comerciales hasta estacionamientos de gran capacidad, pues proporciona una herramienta escalable sofisticada y adaptable a diferentes tipos de entornos de aparcamiento.

La orientación de este proyecto sugiere una resolución técnica avanzada al problema de la administración de estacionamientos, además, proporciona un incremento notable en la eficiencia en las operaciones. Al automatizar la identificación de espacios disponibles, se suprime la necesidad de técnicas manuales y se optimiza el aprovechamiento de los recursos disponibles. Esta innovación puede servir como modelo de implementaciones venideras de IA en la administración urbana y puede ser adaptada a otros contextos similares.

En resumen, la creación de un modelo de IA fundamentado en CNN para identificar los espacios disponibles en estacionamientos locales representa un aporte relevante e innovador. Este proyecto no solo

se alinea con las tendencias tecnológicas, sino que también aborda necesidades concretas de una localidad, en este caso, Jalisco, y ofrece una solución eficiente, así como escalable a un problema urbano persistente.

## 1.3. Problema

La problemática derivada de la administración ineficaz de los espacios de estacionamiento afecta directamente a una amplia variedad de usuarios, incluyendo conductores que buscan estacionar sus vehículos, así como a los administradores de estos estacionamientos. Este problema se manifiesta principalmente en zonas de alta concentración, tales como centros comerciales, escuelas, zonas empresariales y residenciales en ciudades grandes, como Jalisco.

La dificultad principal reside en la ausencia de acceso e integración de un sistema eficiente y automatizado para detectar y administrar los espacios disponibles en tiempo real. Los métodos tradicionales, como la señalización manual, los sistemas de conteo básicos y los sensores, no son capaces de manejar la creciente demanda y complejidad de las infraestructuras urbanas. La ineficiencia en la detección de espacios libres lleva a una búsqueda prolongada y frustrante, congestionando aún más el tráfico y disminuyendo la satisfacción del usuario [5].

Las CNN son particularmente adecuadas para mejorar la gestión de estacionamientos mediante la automatización y la precisión en la identificación de espacios vacíos, gracias a su habilidad para analizar imágenes y detectar patrones complejos [8]. Hasta la fecha, se han implementado varias soluciones, incluyendo tecnologías de señalización inteligente basadas en sensores que detectan la ocupación de los espacios, no obstante, estos sistemas suelen tener un alto costo, debido al hardware utilizado y requieren mantenimiento constante, lo que limita su escalabilidad [2] (véase Apéndice C: Estimación de costos para sistema de sensores SPI — Ejemplo con 275 lugares de estacionamiento). En años recientes, se han implementado técnicas fundamentadas en IA, aunque gran cantidad de estos aún están en etapas experimentales y no han sido implementados de forma amplia en diversas regiones, incluyendo en gran medida a México.

La propuesta se centra en el desarrollo de un modelo de IA fundamentado en CNN con el fin de identificar áreas de parqueo vacíos en tiempo real, mediante el análisis de imágenes, se buscará una solución más accesible y eficiente frente a los métodos actuales.

No se incluirá la realización física del modelo en un estacionamiento real, ni se enfocará en el monitoreo de seguridad o la detección de tipos específicos de vehículos. Asimismo, no se abordará la optimización del tránsito en las áreas circundantes del estacionamiento. No obstante, la mejora en la gestión de espacios de estacionamiento podría generar indirectamente una mejor experiencia para los usuarios mediante la disminución de los tiempos de búsqueda.

Resolver este reto es esencial para optimizar la eficacia en la administración de estacionamientos, además de contribuir a disminuir la congestión vehicular en áreas urbanas. Un sistema eficaz de identificación de espacios libres ahorra tiempo y reduce el estrés para los conductores, lo cual favorece la disminución de las emisiones contaminantes de vehículos al minimizar el tiempo de búsqueda de estacionamiento [5].

Si el problema no se resuelve, la congestión del tráfico y la insatisfacción del usuario continuarán aumentando. Los conductores seguirán enfrentando dificultades para encontrar espacios de estacionamiento disponibles, lo que genera una pérdida de tiempo importante y una repercusión adversa en la calidad de vida urbana [5]. Además, los administradores de estacionamientos y las empresas que

dependen de una gestión eficiente de estos perderán oportunidades para optimizar sus recursos y mejorar la renta generada [2].

## 1.4. Objetivos

### *1.4.1. Objetivo general*

Formular un prototipo de IA fundamentado en CNNs destinado a la identificación y categorización precisa, eficiente y en tiempo real de espacios vacíos en estacionamientos locales, con el fin de optimizar la gestión del estacionamiento y mejorar la experiencia del usuario.

### *1.4.2. Objetivo específico*

- Comprender el problema y la necesidad actual relacionada con la disponibilidad de lugares de estacionamiento en zonas urbanos reales.
- Seleccionar las colecciones de datos más adecuados para entrenar y evaluar distintos enfoques de modelado, incluyendo modelos clásicos y de DL.
- Preprocesar y limpiar los datos seleccionados, asegurando su calidad en la etapa de entrenamiento.
- Entrenar modelos clásicos de ML (como KNN, regresión logística y árbol de decisión) para categorizar imágenes de espacios entre ocupados o vacíos.
- Implementar un modelo orientado a la identificación de objetos basado en YOLOv8, capaz de identificar automáticamente la localización y ocupación de los sitios de parqueo en imágenes completas captadas por cámaras de seguridad.
- Analizar el desempeño de todos los modelos a través de indicadores cuantitativos como exactitud, precisión, sensibilidad, mAP y matriz de confusión, junto con las curvas de entrenamiento y validación, con el fin de validar su comportamiento y generalización.
- Comparar cuantitativamente el rendimiento del modelo basado en CNN con los métodos tradicionales, destacando las mejoras en precisión, eficiencia operativa y escalabilidad.
- Proponer una interfaz gráfica que permita la observación de la accesibilidad en tiempo real de espacios y facilite la integración con sistemas de administración de estacionamientos.
- Formular recomendaciones para la implementación práctica del sistema, considerando la infraestructura necesaria, limitaciones y posibles desafíos de integración.

## 1.5. Innovación tecnológica

El uso de CNN en CV para identificar en tiempo real los espacios vacíos en estacionamientos locales representa una innovación tecnológica, pues aprovecha la singular habilidad de las CNN para procesar imágenes y detectar patrones complejos, ofreciendo una solución más exacta y automatizada que sobrepasa los procedimientos convencionales.

En contraposición a los sistemas basados en sensores, que generalmente son caros y demandan hardware y personal especializado, este proyecto sugiere una opción más asequible y escalable (véase Apéndice C:

Estimación de costos para sistema de sensores SPI — Ejemplo con 275 lugares de estacionamiento). La aplicación de CNN no solo incrementa la eficacia en la identificación de espacios vacíos, sino que también mejora la viabilidad financiera del sistema. Este elemento revolucionario es esencial para la maximización de recursos en los estacionamientos.

Como aporte adicional, se llevó a cabo el uso combinado de datasets públicos, seleccionando el más adecuado según el tipo de modelo empleado: imágenes segmentadas para modelos clásicos de ML, e imágenes completas con anotaciones para el modelo de detección con YOLOv8. Esta metodología no solo posibilitó entrenar modelos con distintos niveles de complejidad, sino también comparar objetivamente el rendimiento entre técnicas tradicionales y modernas, justificando con datos la adopción de enfoques basados en DL.

Se integró la herramienta MLflow para el seguimiento y gestión de los experimentos, lo cual facilitó la organización de pruebas, el control de versiones de los modelos y la valoración estructurada de resultados. Esta práctica fortalece la trazabilidad, la reproducibilidad del trabajo y representa un paso hacia la profesionalización del desarrollo de soluciones basadas en IA. Asimismo, los experimentos se ejecutaron utilizando Google Colab como entorno de desarrollo colaborativo, lo que permitió aprovechar recursos de cómputo en la nube de forma flexible.

La innovación propuesta promete minimizar el tiempo requerido para encontrar estacionamiento y maximizar el uso eficiente de los espacios, beneficiando tanto a los usuarios como a los administradores de estacionamientos. Este modelo sugiere un progreso en el acceso y aplicabilidad práctica de la tecnología, pues facilita la adopción del modelo en entornos reales y ofrece una solución completa y operativa.

El proyecto incluye una valoración del desempeño del modelo implementado en comparación con métricas obtenidas mediante métodos tradicionales, con el fin de validar su efectividad y aportar datos cuantitativos sobre mejoras en precisión y eficiencia. Asimismo, se realizó una comparación utilizando diferentes conjuntos de datos, configuraciones de entrenamiento y técnicas complementarias como data augmentation y TL, para así proporcionar información a la comunidad científica sobre la efectividad de las técnicas de IA en aplicaciones prácticas.

En resumen, el proyecto propuesto introduce una aplicación de tecnologías avanzadas en un área específica y proporciona una alternativa práctica y asequible para optimizar la administración de estacionamientos, contribuyendo de manera significativa a la comunidad científica y a la industria.

# 2. ESTADO DEL ARTE

A lo largo del estado del arte, se presenta un estudio detallado de las principales investigaciones, tecnologías y enfoques aplicados en el monitoreo automatizado de estacionamientos. Se examinan soluciones basadas en CV que permiten detectar y gestionar espacios disponibles mediante el procesamiento de imágenes. También se revisa el papel de las CNN en la identificación del estado de ocupación, así como los métodos de ML más empleados en esta área. El uso de enfoques basados de DL para el reconocimiento de elementos se aborda en conjunto con herramientas, incluyendo PyTorch y Torchvision, fundamentales para la implementación de modelos avanzados. Asimismo, se consideran estrategias modernas como la formación de datos sintéticos mediante redes neuronales generativas, el uso de sistemas multimodales, junto con la adopción de RL en entornos dinámicos. La revisión incluye además el análisis del despliegue en la nube, el aprovechamiento de conjuntos de datos públicos, y una comparativa entre modelos tradicionales y redes neuronales profundas. Para concluir, se precisan las problemáticas presentes en la detección de espacios en tiempo real, así como diversas aplicaciones comerciales ya implementadas en contextos reales.

## 2.1 Introducción al monitoreo automatizado de estacionamientos

Actualmente, nos encontramos ante una oportunidad de perfeccionamiento en la gestión de los espacios de estacionamiento, esto debido a la falta de implementación real de sistemas para solucionar este problema [10]. Este tipo de sistemas de control pueden proporcionar funciones inteligentes como monitorear, gestionar y rastrear el comportamiento de un estacionamiento, lo que permite a las empresas regular el espacio disponible para ofrecer condiciones óptimas a sus clientes, trabajadores y visitantes, pues reduce el tiempo que los empleados pasan buscando un lugar disponible, lo que incrementa la productividad y reduce el estrés [11].

Las cámaras de un estacionamiento, además de ser una herramienta de seguridad, se podrían utilizar también para el monitoreo automático del uso y la disponibilidad de espacios. Cada cámara abarca ciertos números de espacios, y esta información después se procesa para calcular cuántos están ocupados o disponibles. El resultado obtenido se refleja en pantallas instaladas en puntos estratégicos y mediante un sistema de iluminación dinámica, se guía a los clientes hacia los espacios vacantes específicos, reduciendo los periodos de búsqueda y elevando la calidad de la experiencia del usuario [12].

La información que proporcionan estos sistemas facilita la gestión del estacionamiento, ya que registran horarios que permiten reconocer patrones de ocupación y facilitar la toma de decisiones de inversión mejor informadas, además, este sistema permite gestionar eventos de mano de obra, monitorear las instalaciones, así como responder a eventos rápidamente [12].

Una implementación de este tipo de sistemas, utilizada en los últimos años, consiste en el uso de sensores capaces de detectar los automóviles, mediante el cálculo de distancias. Estos sensores, utilizan el protocolo

de comunicación Interfaz Periférica Serial (SPI, por sus siglas en inglés), para comunicarse con una estación central y posteriormente procesar la información [13].

El desarrollo tecnológico actual está impulsando la incorporación de la IA en distintos campos, con el fin de impactar en la potenciación de la calidad de vida por medio de la automatización operativa. Esto se ha visto en diversas áreas como la ciencia, empresas e industrias, permitiendo que las máquinas aprendan de manera autónoma a partir de errores e información disponible, mediante algoritmos [14]. El presente trabajo utiliza cámaras de seguridad conectadas a una red de Fidelidad Inalámbrica (Wi-Fi) y a una tecnología de reconocimiento óptico mediante CV [15] para la identificación de disponibilidad de espacios en un estacionamiento.

En esta infraestructura, la escalabilidad, la funcionalidad y la seguridad deben estar en equilibrio. Para lograrlo, se examinarán los métodos locales y basados en la nube. Al utilizar la estrategia local, las empresas administran y alojan su propia infraestructura de nube privada en el sitio. Esto les proporciona dominio absoluto sobre la seguridad y el ajuste de los centros de datos, lo que les permite adaptarlos a sus propios requisitos. Sin embargo, este grado de control tiene un alto costo porque las empresas necesitan manejar la administración, las actualizaciones y el mantenimiento, además de contratar trabajadores altamente calificados en campos como la administración de sistemas, las redes y las plataformas en la nube [16].

Por el contrario, un proveedor de servicios tiene la responsabilidad de alojar, preservar y gestionar la infraestructura en un modelo de nube privada alojado. Los procedimientos de implementación y aprovisionamiento se agilizan con este método, permitiendo a las empresas concentrarse en sus operaciones principales mientras asignan tareas de gestión y mantenimiento a profesionales calificados [16].

Las CNN representan una herramienta valiosa en segmentación de imágenes, sistemas de identificación y clasificación de objetos, así como procesamiento de lenguaje natural, gracias a su excelente rendimiento y a la falta de preprocesamiento de datos. Estas redes utilizan el procedimiento de convolución en uno de sus estratos, en vez de simplemente la operación de multiplicación matricial. El proceso de convolución amalgama dos funciones para producir una tercera, esta indica el número de superposición entre una función y otra. Si consideramos f como la función inicial y g como el eje, la convolución de ambas funciones (el mapa de características), se denomina f $*$ g y se distingue por la integral de la multiplicación de las dos expresiones, tras efectuar un desplazamiento de magnitud t en una de ellas [17].

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\eta) g(t - \eta) d\eta$$

**Ecuación 1 Definición de la Convolución de dos funciones**

$$s(t) = (f * g)(t) = \sum_{n} f(\eta) g(t - \eta)$$

**Ecuación 2 Definición de la convolución en forma discreta**

En DL la entrada consta de un grupo de datos de diversas dimensiones, mientras que el núcleo está formado por un grupo de parámetros de diversas tamaños que se modifican a través de un modelo de aprendizaje. Por ejemplo, al utilizar como entrada una imagen en dos dimensiones I, se consigue un núcleo en dos dimensiones K. [17].

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)\, K(i-m, j-n)$$

**Ecuación 3 Convolución discreta bidimensional de una imagen**

En contraposición, también existe la correlación cruzada, que opera de forma parecida a la convolución, aunque sin la inversión de la matriz del núcleo.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

**Ecuación 4 Definición de la correlación cruzada bidimensional**

Las CNN constan fundamentalmente de capas convolucionales, capas de reducción y FC, con la finalidad de producir patrones representativos basados en la información suministrada [17].

## 2.2 Arquitecturas CNNs en la detección del estado de lugares de estacionamiento

Existen varias arquitecturas aplicadas a CV para la clasificación de imágenes, entre los más destacados se encuentran: VGG (Visual Geometry Group, por sus siglas en inglés), ResNet y EfficientNet.
La estructura VGG es una CNN construida por el equipo de Geometría Visual de la Universidad de Oxford, sobresale por sus cualidades en clasificación y debido a su popularidad dentro de la comunidad de DL [21].

La arquitectura VGG se distingue por su estructura profunda y homogénea. La red se fundamenta en bloques convolucionales que se agrupan en capas. Cada bloque de convolución emplea filtros de tamaño 3 x 3 y una constante de 64. Estos bloques se unen en agrupaciones de dos o tres, y tras cada bloque convolucional, se sitúa una capa de agrupación utilizando el método de máximo agrupamiento (en inglés, max pooling) utilizando una ventana con dimensión de 2x2 y un deslizamiento de 2. La estructura repetitiva facilita que la red procese datos de las imágenes con un nivel superior de complejidad. Normalmente, la arquitectura VGG consta de 16 o 19 capas, dependiendo de la variante escogida [21]. Luego, las propiedades tridimensionales se convierten en un vector de única entrada mediante la capa de aplanamiento (en inglés, flatten). Posteriormente, se implementan capas FC con una activación Unidad Lineal Rectificada (ReLU, en inglés), seguidas de una última capa que emplea una función de activación softmax, con la finalidad de realizar la clasificación definitiva [21].

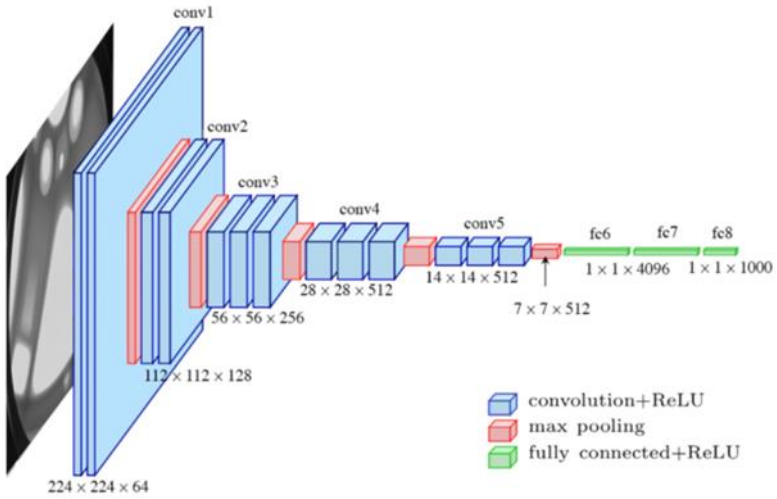



**Figura 1 Arquitectura detallada de la red VGG: Estructura de capas convolucionales con filtros 3x3, max pooling 2x2 y FC con activación ReLU para clasificación de imágenes [21]**

La efectividad de las CNN en clasificación es alta, pero se debe considerar que entre más profunda se vuelve la red, surgen diversos desafíos como la complejidad de entrenar redes aún más profundas, así como el degradado del rendimiento. La arquitectura ResNet (Residual Network, en inglés) aborda estas complicaciones y supera en rendimiento en las competencias de clasificación de imágenes [21]. Esta arquitectura emplea conexiones residuales para transmitir el gradiente a través de las capas de la red mediante la adición de una conexión directa que proporcione una o varias capas. Esta conexión suma los valores de activación de una capa previa a la activación de una capa posterior, lo que permite que el gradiente se extienda sin limitaciones a través de la conexión residual, previniendo de esta manera el desvanecimiento y la explosión del mismo [21].
La función esencial de la arquitectura ResNet es la identidad residual, expresado por la adición de las capas de entrada y salida. Cuando se la incorpora, se asegura que la capa pueda aprender modificaciones en la función identidad, lo que facilita a la red una mejor adaptación a los datos de entrada [21].

ResNet es particularmente apropiada para TL debido a su estructura profunda y conexiones residuales. TL es un método de IA que emplea un modelo previamente entrenado de una Red Neuronal Artificial (ANN) para entrenar un grupo de imágenes de tamaño reducido con numerosas propiedades parecidas al modelo previo [17].

Además, la arquitectura de red neuronal conocida como EfficientNet ha sobresalido en cuanto a desempeño y eficiencia en computación [21]. La arquitectura previa adopta un enfoque escalable, en el que se regula el tamaño de la red a través de un parámetro conocido como "compound scaling". Este

parámetro escala de manera uniforme la resolución, la profundidad y el ancho, por lo que el modelo es flexible ante distintos recursos informáticos y grupos de datos [21].

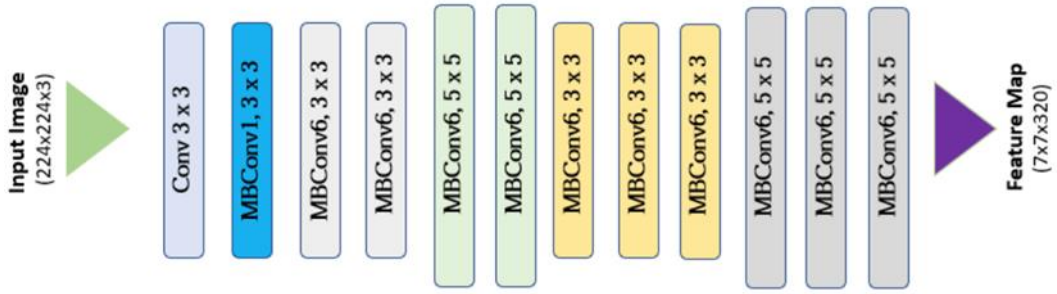



**Figura 2 Arquitectura de la red MobileNetV2: Estructura de bloques convolucionales con bottlenecks invertidos desde entrada 224x224x3 hasta mapa de características 7x7x320 para aplicaciones móviles eficientes [21]**

El bloque fundamental empleado en EfficientNet se fundamenta en la estructura del bloque Inverted Residual empleado en MobileNetV2, que consta de una convolución 1x1, luego una convolución 3x3 y finalmente, otra convolución 1x1. La implementación de estas convoluciones reducidas contribuye a disminuir el número de operaciones y el costo total de computación del modelo, sin afectar de manera significativa su rendimiento [21].

## 2.3 Sistemas de CV para estacionamiento inteligente

Los modelos de detección usan CNNs como “backbones” o “extractores de características”, y encima agregan componentes específicos para localizar y clasificar objetos. La segmentación de imágenes es un modelo de visión computacional que segmenta una imagen digital en conjuntos discretos de píxeles (segmentos de imagen) con el objetivo de identificar objetos o ejecutar funciones vinculadas. Esta metodología de dividir la información visual compleja de una imagen en segmentos con formas determinadas facilita un procesamiento más ágil y eficaz [18]. Dentro de un sistema de reconocimiento, el procedimiento de segmentación se basa en separar los elementos de interés en el fondo de la imagen, otorgándoles una intensidad de 1 a los objetos y un valor de 0 al fondo, lo que produce una imagen binaria. Para identificar espacios disponibles en estacionamientos mediante imágenes de estacionamientos, existen diversos elementos que pueden afectar este proceso, como las fluctuaciones de la luz, el clima, las sombras, la oclusión y los diferentes colores de los automóviles. Si se implementa correctamente, la segmentación puede ayudar a identificar con precisión los espacios de estacionamiento, los vehículos y otras áreas, como carreteras y banquetas. Sin embargo, esta técnica requiere un mayor volumen de datos anotados (con polígonos o máscaras en lugar de solo cajas delimitadoras), consume más recursos de cómputo y puede resultar excesiva en escenarios donde la cámara ofrece buena visibilidad y los espacios están claramente delimitados [19].

La segmentación de imágenes puede ser de dos tipos: Segmentación semántica, la cual asigna una etiqueta a cada píxel (por ejemplo, "vehículo", "espacio libre", "carretera") sin diferenciar instancias específicas de objetos y la Segmentación de instancias, que no solo etiqueta cada píxel, sino que también identifica instancias individuales de objetos, como diferentes autos en una imagen [18].

Un mapa de ocupación en tiempo real constituye una visualización que ilustra la existencia de espacios disponibles para estacionamiento en un área determinada. Utilizando imágenes o videos capturados por cámaras, el sistema procesado por algoritmos de DL detecta y clasifica los lugares de estacionamiento como disponibles u ocupados y visualiza esta información en un mapa. Estos mapas se utilizan en plataformas de gestión de tráfico, aplicaciones móviles para conductores o sistemas de control de estacionamientos en tiempo real.

YOLO, SSD (Single Shot MultiBox Detector, en inglés) y Faster R-CNN son algunos de las técnicas más habituales en el reconocimiento de piezas en tiempo real. El modelo YOLO, implementado por Joseph Redmon en 2016, opera a través de la segmentación de la fotografía en una estructura cuadrada y la realización de proyecciones sobre los cuadros delimitadores y las posibilidades de detección en función de cada tipo de entrenamiento. Debido a su método, YOLO tiene la capacidad de efectuar proyecciones veloces y exactas al poner una única red neuronal en toda la imagen, lo que facilita la identificación casi instantánea de objetos en imágenes y videos [20]. Posteriormente, se realiza un procedimiento de supresión de no-máximos para borrar las diversas identificaciones relacionadas con el objeto en cuestión [16]. Sin embargo, algunas desventajas de YOLO incluyen un rendimiento inferior al detectar objetos que están muy cerca entre sí o que son muy pequeños.

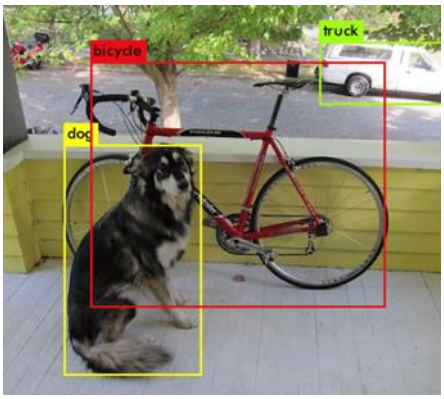



**Figura 3 Demostración práctica del algoritmo YOLO: Detección en tiempo real de múltiples objetos (perro, bicicleta, vehículo) con cajas delimitadoras y clasificación automática [18]**

En contraposición, el modelo SSD es otro algoritmo ampliamente empleado para el reconocimiento en tiempo real, ideado por Wei Liu en 2016. Este modelo se fundamenta en una CNN que genera una serie de cajas delimitadoras y valoraciones de acuerdo con el objeto que se identifica en ellas. Las capas iniciales de la red se basan en el esquema backbone o VGG-16. Después, se introducen capas convolucionales que reducen gradualmente su tamaño, lo que facilita la realización de proyecciones a diversas escalas. El modelo para predecir detecciones cambia para cada nivel de capa [21].

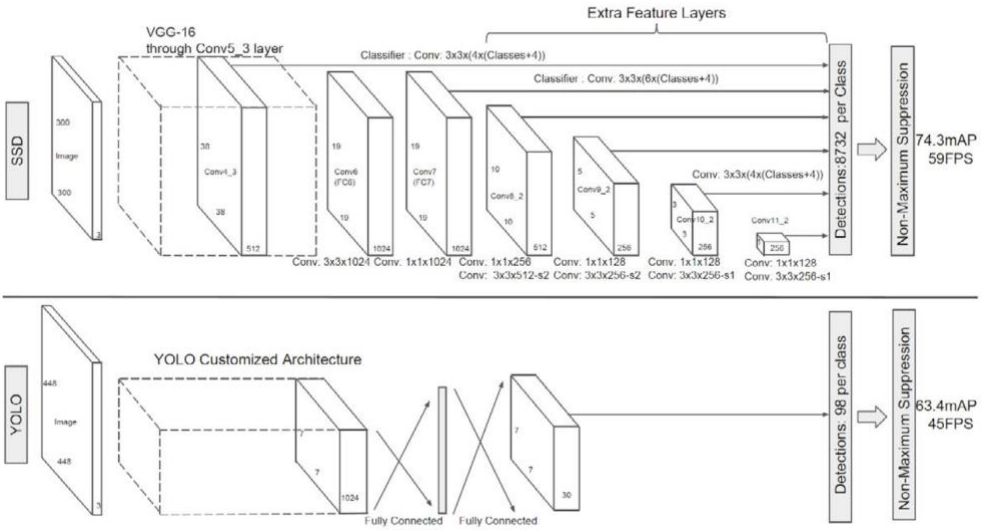



**Figura 4 Comparación de arquitecturas de CNN: Estructura SSD con backbone VGG-16 (superior) y arquitectura personalizada de YOLO (inferior) para detección de objetos en múltiples escalas [18]**

Cada capa adicional tiene la capacidad de producir un conjunto estable de proyecciones mediante un conjunto de filtros de convolución. Para cada capa de dimensiones m×n con p canales, el componente esencial para prever los parámetros de una posible identificación es necesario un diminuto núcleo de dimensión 3 × 3 × p que genera, ya sea una calificación por categoría, o un método ajustable en relación con las coordenadas preestablecidas de la caja. En cada sitio donde se implementa el núcleo, se logra un valor de salida, el cual se evalúa en relación con la ubicación de una caja, que también se relaciona con la localización de cada mapa [21].

En 2015, la Faster R-CNN fue presentada por Jian Sun, Ross B. Girshick, Kaiming He y Shaoqing Ren. La configuración de Faster R-CNN comprende una Red de Propuestas de Regiones (RPN), que intercambia los rasgos convolucionales de la imagen en su totalidad con la red detectora, permitiendo así la adquisición prácticamente sin costo de propuestas de regiones. "El R-CNN más rápido comprende capas convolucionales que crean un mapa de características basándose en la imagen inicial, una capa de agrupación de Región de Interés (RoI) que consigue un vector de características con tamaño fijo a partir del mapa a través de sugerencias (cajas de contorno y sus valoraciones), y capas FC que hacen uso de este vector para categorizar el objetivo y mejorar el cuadro delimitante" [21].

YOLO se destaca sobre SSD en la detección de vehículos en estacionamientos principalmente por su mayor velocidad de procesamiento y arquitectura optimizada para tareas en tiempo real, lo que es crucial para sistemas que requieren respuesta inmediata. Aunque ambos modelos son capaces de detectar objetos de tamaño mediano a grande, como automóviles, YOLO ofrece una mejor precisión en la localización y clasificación debido a su procesamiento global de la imagen, lo que reduce falsos positivos y mejora la detección en situaciones con múltiples vehículos cercanos o parcialmente oculta. Por otro lado, la segmentación, aunque proporciona información detallada a nivel de píxel, requiere una anotación más compleja y un mayor poder computacional, lo cual puede ser innecesario para la simple clasificación binaria de espacios libres u ocupados en estacionamientos [20].

## 2.4 Algoritmos de ML en el monitoreo de estacionamientos

La detección de estacionamientos es un problema en el que se puede aplicar tanto algoritmos supervisados como no supervisados, dependiendo de la información disponible y el tipo de resultado que se quiera obtener. Los algoritmos supervisados requieren etiquetas de los datos destinados al entrenamiento del modelo [22]. En el caso de la detección de lugares de estacionamiento, esto implica contar con un conjunto de imágenes o datos que incluyen las cajas delimitadoras (bounding boxes) que señalan la ubicación exacta de cada espacio dentro de la imagen, así como su respectiva etiqueta de “espacio disponible” o “espacio ocupado”.

Algunos algoritmos supervisados son SVM y K-NN. SVM opera relacionando datos con un espacio de características de gran escala dimensional, permitiendo la categorización de los puntos de datos, incluso si no se pueden separar los datos de manera lineal de otra forma. Se determina una separación entre las categorías y los datos se modifican para que dicha separación tenga la capacidad de representarse mediante un hiperplano [23].

Por otra parte, el algoritmo K-NN utiliza la distancia entre puntos para determinar la categoría o grupo de un dato individual. Para la detección de estacionamientos, si una imagen de un espacio de estacionamiento se parece a otras imágenes etiquetadas como "disponible", se clasificará como disponible [24]. En este caso, ambos servirían para clasificar dos clases: espacio de estacionamiento disponible y ocupado.

El aprendizaje no supervisado utiliza datos sin etiquetas o sin preprocesar [22]. En la detección de estacionamientos, esto puede ser útil cuando no se dispone de etiquetas claras sobre los estados del espacio de estacionamiento (ocupado o disponible), por ejemplo, una cámara de seguridad en tiempo real. Uno de los algoritmos no supervisados es el clustering, el cual organiza y clasifica diferentes objetos, basados en similitudes o patrones [25]. Este es útil en este proyecto, si los lugares de estacionamiento son diversos y los patrones deben aprenderse sobre la marcha sin intervención humana.

Las RNN y las Memorias de Largo Corto Plazo (LSTMs, por sus siglas en inglés) son modelos diseñados específicamente para procesar secuencias y datos con dependencias temporales, lo que los posiciona como un mecanismo eficiente para la identificación de espacios de estacionamiento a partir de videos, ya que permiten modelar la continuidad temporal entre fotogramas. Una RNN es un modelo de DL diseñado para procesar datos secuenciales y transformarlos en una salida secuencial determinada. Una RNN puede procesar una secuencia de fotogramas de video y usar la información de los fotogramas anteriores para tomar decisiones sobre el estado actual de los espacios de estacionamiento [26].

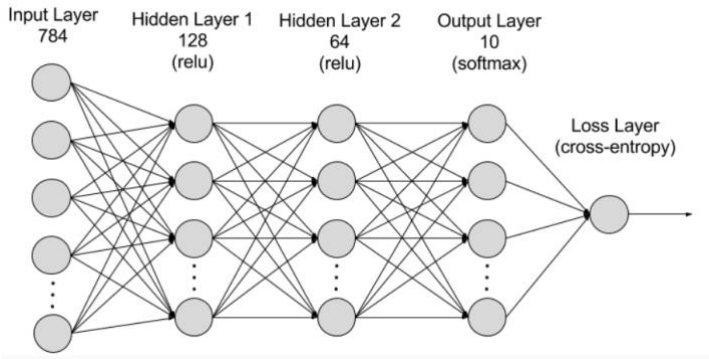



**Figura 5 Arquitectura de red neuronal completamente conectada: Estructura multicapa con capa de entrada de 784 neuronas, capas ocultas de 128 y 64 neuronas con ReLU, capa de salida de 10 neuronas con softmax y función de pérdida cross-entropy [27]**

Las LSTMs son una versión de las RNNs creada para reducir el reto relacionado con el "desvanecimiento del gradiente" mediante una estructura de celdas especializadas que conservan información importante a lo largo de intervalos extensos. Utilizan "puertas" para decidir qué información debe ser almacenada, actualizada o desechada en cada paso de tiempo. Para la detección de espacios de estacionamiento en video, las LSTMs podrían monitorear cada fotograma para detectar eventos clave, como la entrada o salida de un automóvil. A medida que un vehículo se aproxima a un espacio, las LSTMs pueden analizar los cambios sutiles entre fotogramas sucesivos (movimiento del vehículo, cambios en las sombras, etc.) para prever si un espacio está a punto de ser ocupado o liberado. Si el proceso de ocupación de un espacio tarda o se distribuye a lo largo de muchos fotogramas, la LSTM es capaz de mantener en su "memoria" el estado del espacio a lo largo de ese período [27].

## 2.5 Uso de DL para el reconocimiento de elementos

La segmentación semántica es un procedimiento de CV que otorga una clasificación a los píxeles mediante un algoritmo de DL. Esta es una de las tres subcategorías en el proceso global de segmentación de imágenes que asiste a las computadoras en la interpretación de la información visual. La segmentación semántica reconoce grupos de píxeles y los categoriza según distintas propiedades. La segmentación de instancias y la segmentación panóptica son las otras dos subcategorías de la segmentación de imágenes [28]. La segmentación semántica podría emplearse para diferenciar áreas ocupadas de zonas libres en tiempo real.

La versión original de YOLO empleaba la arquitectura YOLOv1, la cual ha sido mejorada en versiones posteriores (YOLOv2, YOLOv3, YOLOv4, YOLOv5, y ahora YOLOv8). A continuación, se describe la versión YOLOv3 para explicar su funcionamiento, así como sus principales componentes.

YOLOv3 se le conoce también como Darknet-53 debido a que está formado por 53 capas convolucionales. Cada capa convolucional se encuentra después de la normalización por lotes (conocida en inglés como normalización de batches) y la función de activación Leaky ReLU. Ninguna capa reductora se emplea, excepto en las CC con paso=2. Lo previamente señalado disminuye la extensión del mapa de características, previniendo la pérdida de las características de jerarquía inferior que se asignan a las capas

de reducción-máxima. Ya que esta estructura emplea capas convolucionales y no tiene capas residuales, se percibe como una red neuronal rápida, aunque su precisión en la identificación y clasificación se reduce.

| | Type | Filters | Size | Output |
|---|---|---|---|---|
| | Convolutional | 32 | 3 × 3 | 256 × 256 |
| | Convolutional | 64 | 3 × 3 / 2 | 128 × 128 |
| 1× | Convolutional | 32 | 1 × 1 | |
| | Convolutional | 64 | 3 × 3 | |
| | Residual | | | 128 × 128 |
| | Convolutional | 128 | 3 × 3 / 2 | 64 × 64 |
| 2× | Convolutional | 64 | 1 × 1 | |
| | Convolutional | 128 | 3 × 3 | |
| | Residual | | | 64 × 64 |
| | Convolutional | 256 | 3 × 3 / 2 | 32 × 32 |
| 8× | Convolutional | 128 | 1 × 1 | |
| | Convolutional | 256 | 3 × 3 | |
| | Residual | | | 32 × 32 |
| | Convolutional | 512 | 3 × 3 / 2 | 16 × 16 |
| 8× | Convolutional | 256 | 1 × 1 | |
| | Convolutional | 512 | 3 × 3 | |
| | Residual | | | 16 × 16 |
| | Convolutional | 1024 | 3 × 3 / 2 | 8 × 8 |
| 4× | Convolutional | 512 | 1 × 1 | |
| | Convolutional | 1024 | 3 × 3 | |
| | Residual | | | 8 × 8 |
| | Avgpool | | Global | |
| | Connected | | 1000 | |
| | Softmax | | | |

**Figura 6 Arquitectura detallada de Darknet-53 (YOLOv3): Especificaciones de capas convolucionales con tipos de filtros, tamaños de kernel, salidas dimensionales y bloques residuales desde entrada 256x256 hasta clasificación final con 1000 clases [17]**

YOLOv3 tendremos:

| Layer | Type | Filters | Size/Stride | Input | Output |
|---|---|---|---|---|---|
| 0 | Convolutional | 16 | 3 × 3/1 | 416 × 416 × 3 | 416 × 416 × 16 |
| 1 | Maxpool | | 2 × 2/2 | 416 × 416 × 16 | 208 × 208 × 16 |
| 2 | Convolutional | 32 | 3 × 3/1 | 208 × 208 × 16 | 208 × 208 × 32 |
| 3 | Maxpool | | 2 × 2/2 | 208 × 208 × 32 | 104 × 104 × 32 |
| 4 | Convolutional | 64 | 3 × 3/1 | 104 × 104 × 32 | 104 × 104 × 64 |
| 5 | Maxpool | | 2 × 2/2 | 104 × 104 × 64 | 52 × 52 × 64 |
| 6 | Convolutional | 128 | 3 × 3/1 | 52 × 52 × 64 | 52 × 52 × 128 |
| 7 | Maxpool | | 2 × 2/2 | 52 × 52 × 128 | 26 × 26 × 128 |
| 8 | Convolutional | 256 | 3 × 3/1 | 26 × 26 × 128 | 26 × 26 × 256 |
| 9 | Maxpool | | 2 × 2/2 | 26 × 26 × 256 | 13 × 13 × 256 |
| 10 | Convolutional | 512 | 3 × 3/1 | 13 × 13 × 256 | 13 × 13 × 512 |
| 11 | Maxpool | | 2 × 2/1 | 13 × 13 × 512 | 13 × 13 × 512 |
| 12 | Convolutional | 1024 | 3 × 3/1 | 13 × 13 × 512 | 13 × 13 × 1024 |
| 13 | Convolutional | 256 | 1 × 1/1 | 13 × 13 × 1024 | 13 × 13 × 256 |
| 14 | Convolutional | 512 | 3 × 3/1 | 13 × 13 × 256 | 13 × 13 × 512 |
| 15 | Convolutional | 255 | 1 × 1/1 | 13 × 13 × 512 | 13 × 13 × 255 |
| 16 | YOLO | | | | |
| 17 | **Route 13** | | | | |
| 18 | Convolutional | 128 | 1 × 1/1 | 13 × 13 × 256 | 13 × 13 × 128 |
| 19 | Up-sampling | | 2 × 2/1 | 13 × 13 × 128 | 26 × 26 × 128 |
| 20 | **Route 19 8** | | | | |
| 21 | Convolutional | 256 | 3 × 3/1 | 13 × 13 × 384 | 13 × 13 × 256 |
| 22 | Convolutional | 255 | 1 × 1/1 | 13 × 13 × 256 | 13 × 13 × 256 |
| 23 | YOLO | | | | |

**Figura 7 Arquitectura detallada de YOLOv3: Especificaciones capa por capa con tipos de operaciones, número de filtros, tamaños de kernel/stride, dimensiones de entrada y salida desde 416x416x3 hasta detección YOLO multi-escala [17]**

La imagen de origen del sistema YOLOv3 puede ser configurada en diversas dimensiones, en cada proceso de entrenamiento, siempre que las imágenes tengan forma cuadrada y una dimensión múltiple de 32, tal como, 608x608 o 416x416. La entrada se proporciona por:

$$Entrada(m, h, w, d)$$

**Ecuación 5 Dimensiones de entrada del modelo YOLOv3**

Donde:

- *m* se refiere al volumen del conjunto de entrenamiento.
- *h* representa la altura.
- *w* representa la anchura.
- *d* corresponden a los canales de la imagen de origen.

Por ejemplo, para representar una entrada de YOLOv3 utilizando un lote de 64 imágenes dividido en subgrupos de 16 (procesando 4 imágenes simultáneamente en cada iteración de CNN, así como imágenes de 3 canales (RGB) de 416x416, tendremos:

$$Entrada(64, 416, 416, 3)$$

**Ecuación 6 Forma del tensor de entrada para YOLOv3 con lote de imágenes RGB de 416x416**

YOLOv3 realiza el procedimiento de identificación en tres niveles de escala diferentes, escalando la imagen de entrada en proporciones de 32, 16 y 8 [17].

Por citar un caso, una fotografía de 416x416 se disminuiría en un factor de 32, por lo que el mapa de características tendría un tamaño de 13x13. Se denomina celda a cada componente del mapa. En el mapa de características, cada celda anticipa una cantidad determinada de cajas de delimitación. Cada bloque dentro de un mapa de atributos posee las siguientes características: BX(5+C). B indica la anticipación de áreas demarcadas correspondientes a cada segmento. Cada delimitador B puede orientarse a reconocer un tipo definido de objeto. 5+C muestra las características de cada uno de los cuadros delimitadores, las cuales detallan las coordenadas centrales, las medidas, la calificación de precisión, así como las probabilidades de clase C correspondientes a cada recuadro. Por lo que YOLOv3 anticipa tres cuadros delimitadores para cada celda [17]. En caso de usar una imagen inicial de 416×416, se producirán 3 mapas descriptivos que incluyen 10,647 cuadros de detección [17].

$$\big((52 * 52) + (26 * 26) + (13 * 13)\big) * 3 = 10647$$

**Ecuación 7 Cálculo del total de cuadros delimitadores generados por YOLOv3 para una imagen de entrada de 416 x 416 píxeles**

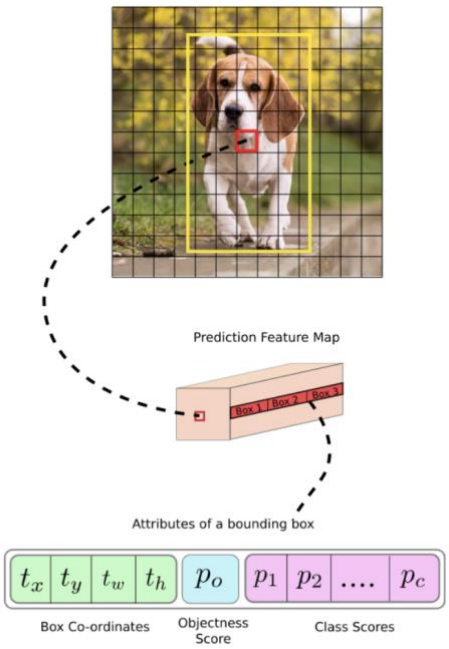



**Figura 8 Funcionamiento de YOLOv3 en detección multi-escala: Ejemplo de segmentación en grilla, generación de mapa de características de predicción y estructura de atributos de cajas delimitadoras con coordenadas, score de objetividad y confidencias de clase [17]**

Con el fin de disminuir la gran cantidad de cajas delimitadoras, se emplean dos filtros. El primero simboliza la mínima restricción en la capacidad de detectar un objeto presente en cada caja delimitadora y, si no se supera, será suprimida. El segundo filtro se emplea para eliminar los no-máximos, con la finalidad de evitar varias detecciones múltiples de un solo objeto, mediante la métrica Intersección sobre Unión (IoU) que presenta la mayor probabilidad de detección y se compara con las demás cajas. Si el resultado del IoU excede un límite fijado, se suprime la caja delimitadora que tiene una probabilidad de detección más reducida [17].


**Figura 9 Proceso de supresión de no-máximos (NMS) en YOLO: Comparación antes y después de la eliminación de cajas delimitadoras redundantes para reducir detecciones múltiples del mismo objeto mediante métrica IoU [17]**

## 2.6 Aplicación de frameworks como PyTorch y torchvision

PyTorch es una de las bibliotecas más destacadas para diseñar modelos de DL debido a su flexibilidad, facilidad de uso, así como su eficiencia. PyTorch es un marco de código abierto que integra la biblioteca de ML de Torch con una Interfaz de programación de aplicaciones (API) de alto nivel en Python, permitiendo la implementación de diversas arquitecturas, desde algoritmos de regresión lineal hasta CNN y modelos generativos.

Un aspecto clave de PyTorch es el uso de tensores, los cuales son estructuras de datos optimizadas para cálculos en GPU, lo que permite un entrenamiento más eficiente en grandes volúmenes de datos. Lo anterior permite el aceleramiento del proceso de imágenes y videos. Además, también facilita la elaboración de modelos adaptados de modo intuitivo mediante clases y módulos, facilitando la exploración de nuevas arquitecturas.

PyTorch cuenta con el sistema de autograd que calcula automáticamente los gradientes necesarios para la optimización de parámetros, simplificando el proceso de entrenamiento y reduciendo los errores. Por otro lado, herramientas como torchvision.transforms permiten realizar transformaciones y preprocesamientos en las imágenes, preparando adecuadamente los datos para el modelo.

Por otra parte, Torchvision es una librería complementaria de PyTorch que ofrece herramientas y utilidades específicas para CV, incluyendo conjuntos de datos, arquitecturas preentrenadas y herramientas de transformación de imágenes.

Una de las mayores fortalezas de torchvision es la oportunidad de aplicar modelos ya capacitados para la identificación de objetos, por ejemplo, Mask R-CNN y Faster R-CNN. Estos modelos pueden ser modificados para funciones particulares, permitiendo también sustituir la base del modelo por una arquitectura distinta de acuerdo con las demandas del proyecto.

Se considera a Faster R-CNN como eficaz, pues predice cuadros delimitadores (bounding boxes) y genera valores de probabilidad por clase para los objetos potenciales en la imagen, alcanzando altos niveles de precisión en conjuntos de datos reconocidos como COCO (Objetos Comunes en Contexto) y PASCAL VOC.

Por su parte, la variante Mask R-CNN de Faster R-CNN, disponible en torchvision, no solo se limita a detectar objetos, sino que también permite segmentarlos, lo que es considerablemente útil en aplicaciones que requieren una identificación precisa de las formas y contornos de los objetos.

Aunque YOLOv5 no está directamente incluido en torchvision, muchos desarrolladores combinan PyTorch y torchvision para el preprocesamiento de datos y el posprocesamiento de resultados de detección. YOLOv5 ha ganado popularidad debido a su velocidad y precisión, siendo una opción destacada en aplicaciones de vigilancia y monitoreo [31].

## 2.7 Modelos de CV para la gestión de estacionamientos en tiempo real

La formación de soluciones utilizando CV en sistemas de monitoreo de estacionamientos en tiempo real presenta varios desafíos técnicos a considerar. Uno de los problemas más críticos es el tiempo de inferencia, que se refiere al periodo requerido de un modelo entrenado en recibir solicitudes y devolver predicciones de manera sincrónica. Un tiempo de inferencia elevado puede resultar en retrasos significativos en la detección de vehículos, lo que afecta la efectividad general del sistema. Por otra parte, en este sistema también es esencial la eficiencia de las cámaras de vigilancia. Por lo tanto, optimizar el rendimiento del modelo para que funcione adecuadamente en estos entornos es crucial para garantizar un monitoreo efectivo y en tiempo real.

Para abordar estos problemas, es fundamental reducir el tamaño de los modelos y optimizarlos para su ejecución en dispositivos con limitaciones de hardware, a partir de las siguientes técnicas:

1. Pruning (Poda): Esta técnica consiste en eliminar pesos o neuronas que generan impacto mínimo en la eficiencia del modelo. La disminución de complejidad del modelo provoca una reducción de su tamaño y se acelera el tiempo de inferencia.
2. Destilación del Conocimiento: Esta técnica implica entrenar un modelo más pequeño, conocido como el "estudiante", utilizando las salidas de un modelo más grande, denominado "maestro". De este modo, el modelo más pequeño puede aprender a generalizar de manera efectiva, manteniendo un rendimiento similar al del modelo más grande, pero con menos recursos computacionales. Es posible entrenar un modelo destilado desde cero o utilizar modelos previamente entrenados que ya están disponibles en diversas bibliotecas.
3. Cuantización: Pasar los pesos del modelo de un formato de punto flotante a uno de menor precisión, como int8, contribuye a disminuir de manera notable la dimensión del modelo y optimizar la velocidad de procesamiento sin comprometer una precisión desmedida en las predicciones. Normalmente, esta técnica se lleva a cabo tras el entrenamiento.
4. Uso de arquitecturas ligeras: Implementar modelos diseñados específicamente para dispositivos con recursos limitados, como MobileNet o SqueezeNet, es una estrategia efectiva. Asimismo, modelos de detección como YOLO (You Only Look Once), especialmente en sus versiones recientes (por ejemplo, YOLOv5 y YOLOv8), se destacan por su capacidad de ofrecer una alta precisión manteniendo tiempos de inferencia reducidos. Estas arquitecturas están optimizadas para ofrecer un balance adecuado entre precisión y eficiencia, permitiendo su uso en entornos con restricciones de hardware.
5. Optimización del código: Además de optimizar el modelo, es importante mejorar la eficiencia del código que ejecuta la inferencia. Esto incluye el uso de bibliotecas optimizadas y el aprovechamiento del paralelismo que ofrecen las GPUs, lo que puede acelerar aún más el proceso de inferencia [32].

## 2.8 Uso de Redes Neuronales Generativas (GAN) para la generación de datos sintéticos

Una GAN es una arquitectura de DL que entrena dos redes neuronales en un proceso competitivo (por eso se le llama antagónica), con el objetivo de producir nuevos datos partiendo de una colección de datos de entrenamiento previamente establecido, al tomar una muestra de datos de entrada y modificarla en la medida de lo posible. La otra red trata de determinar si la salida de datos producida es parte del conjunto de datos inicial (si los datos son reales o falsos), hasta que ya no sea posible diferenciar las nuevas versiones. Lo mencionado anteriormente facilita la construcción de nuevas imágenes mediante una base de datos preexistentes de imágenes. Esta metodología se aplica en ML para incrementar de manera artificial la cantidad de datos de entrenamiento, y, por consiguiente, fortalecer el sistema en situaciones de problemas con conjuntos de datos de acceso complicado. Hay varios tipos de modelos GAN dependiendo de las fórmulas matemáticas empleadas y las distintas maneras en que el generador y el discriminador interactúan entre ellos, algunos de estos incluyen: Vanilla GAN, Conditional GAN y GAN de alta resolución [36].

## 2.9 Sistemas multimodales para la detección de lugares de estacionamiento

La Fusión de cámaras con otros sensores, tales como LiDAR (Detección y Medición por Luz) o ultrasonidos permiten la optimización de la precisión de los modelos. Para lograr esto, se integran técnicas de ML con características de cada sensor, de este modo, los modelos pueden captar patrones más complejos y aumentar precisión en la detección. Esto es de gran utilidad en situaciones desafiantes, como en condiciones con iluminación variable o en zonas densamente pobladas, donde cada tipo de sensor brinda ventajas únicas. LiDAR es una tecnología que emite impulsos láseres de baja potencia, seguros para los ojos, en el espectro del infrarrojo próximo y calcula el tiempo que toma a el láser en completar un desplazamiento de ida y vuelta entre el sensor y un objetivo, lo que permite identificar, clasificar y seguir objetos en movimiento con gran precisión. En condiciones de muy poca luz o de noche, LiDAR mantiene su precisión, por lo que, en conjunto con las cámaras, puede ayudar a hacer más robusto el sistema de detección de lugares de estacionamientos disponibles.

## 2.10 Aprendizaje por refuerzo (RL)

El RL constituye un enfoque que capacita a los programas para tomar decisiones más acertadas. Este proceso simula el método humano de aprendizaje basado en ensayo y error para cumplir metas específicas. Aquellas acciones que aproximan el software a su objetivo son reforzadas, en contraste con las que lo distancian son desestimadas. Los modelos de RL pueden aplicarse para optimizar la asignación y administración de espacios de estacionamiento en tiempo real. En este enfoque, un agente toma decisiones sobre cómo gestionar los espacios de estacionamiento basándose en recompensas y penalizaciones, de esta forma, aprende a optimizar su rendimiento a través de la interacción con las cosas que lo rodean [38].

## 2.11 Implementación de infraestructura en la nube para el monitoreo de estacionamientos

Las plataformas en la nube como Oracle Cloud Infrastructure (OCI), AWS (Servicios Web de Amazon) y Google Colab, son fundamentales para ejecutar y escalar este tipo de modelos de detección de estacionamiento en tiempo real, pues proporcionan recursos computacionales flexibles y potentes. Amazon SageMaker es un servicio totalmente administrado que integra un amplio conjunto de herramientas para facilitar ML de alto rendimiento y rentable para diversos usos. Con SageMaker, los usuarios son capaces de crear, entrenar y utilizar modelos de ML a escala utilizando herramientas como blocs de notas, generadores de perfiles, canalizaciones, depuradores, y MLOps, todo ello dentro de un entorno de desarrollo integrado (IDE). En adición, SageMaker habilita a los usuarios para acceder a una multitud de modelos previamente entrenados, incluidas máquinas virtuales públicas que se pueden implementar con unos pocos clics [39]. Por otra parte, AWS Lambda permite desplegar microservicios para la recolección de datos y extracción de información, facilitando la gestión de datos en tiempo real. Amazon Rekognition proporciona capacidades de CV personalizables y preentrenadas para extraer información de imágenes digitales y videos [40].

Por otro lado, Google Cloud ofrece herramientas como Google Cloud Functions, un entorno escalable de funciones como servicio (FaaS, por sus siglas en inglés) que ejecuta código ante la ocurrencia de eventos sin la gestión de equipos servidores, y Google Colab, que proporciona un entorno en la nube para ejecutar y compartir notebooks de Python con recursos computacionales escalables, como GPUs y TPUs, sin necesidad de administrar infraestructura local [41].

Los procedimientos para recolectar información de dispositivos IoT (Internet de las cosas) comprenden:

- Conexiones Directas: Los aparatos IoT tienen la capacidad de comunicar datos directamente hacia una plataforma central mediante conexiones de cable o sin cables.
- Gateways IoT: En ambientes industriales, se emplean para recolectar información de varios sensores y aparatos antes de enviarla a sistemas de procesamiento de datos o a plataformas de nube. Es imprescindible en circunstancias en las cuales la información producida es ininterrumpida o en tiempo real y necesita un preprocesamiento con el propósito de fusionar la tecnología de la información (IT) y la tecnología operativa (OT, por sus siglas en inglés).
- APIs y Servicios en la Nube: Las APIs posibilitan la incorporación de dispositivos IoT con los servicios en la nube, lo que simplifica la recolección, gestión y resguardo de información. Algunos casos representativos son Microsoft Azure IoT y AWS IoT.

Además, el edge computing permite que el procesamiento de información se realice en dispositivos de borde, como Raspberry Pi o NVIDIA Jetson. Lo anterior implica que ciertos análisis se llevan a cabo de manera local, cerca de la fuente de datos (la cámara), antes de transmitir información relevante a la nube [42].

## 2.12 Uso y análisis de conjuntos de datos públicos en CV para estacionamientos

La construcción de modelos de CV aplicados a la detección de espacios de estacionamiento requiere de colecciones de datos idóneos para su entrenamiento y validación. Los conjuntos de datos públicos desempeñan un papel fundamental en este contexto, ya que permiten estandarizar las pruebas, facilitar la reproducibilidad de los experimentos y reducir los costos asociados a la recopilación de datos propios.
Entre los conjuntos de datos más utilizados se encuentra PKLot, el cual contiene imágenes reales capturadas desde cámaras instaladas en la parte alta de edificios de estacionamiento en diferentes condiciones de clima (soleado, nublado, lluvioso). Las imágenes están acompañadas de anotaciones detalladas en formatos adecuados para tareas de detección (por ejemplo, en formato YOLO), lo que lo hace ideal para entrenar modelos de DL como YOLOv8 [22].

Por otro lado, datasets como el de Kaggle Parking Lot Dataset ofrecen imágenes individuales ya recortadas de espacios de estacionamiento etiquetados como ocupados o vacíos. Si bien estos conjuntos son útiles para tareas de clasificación supervisada mediante modelos clásicos, presentan limitaciones cuando se requiere detección sobre imágenes completas en tiempo real [23]. La elección adecuada del dataset depende directamente del tipo de modelo y objetivo del sistema final.

## 2.13 Comparativa entre modelos clásicos y redes neuronales profundas en visión por computadora

En el desarrollo de soluciones basadas en CV, existen dos enfoques principales: los modelos clásicos de ML y los modelos avanzados de DL. Ambos tienen aplicaciones válidas, sin embargo, exhiben diferencias marcadas en cuanto a complejidad, requerimientos de datos, precisión y aplicabilidad en tiempo real.

Modelos como KNN, regresión logística y árboles de decisión han sido ampliamente utilizados para tareas de clasificación binaria, especialmente cuando se dispone de imágenes previamente procesadas o de características extraídas manualmente. Estos modelos son fáciles de implementar, requieren menor capacidad computacional y pueden ofrecer resultados aceptables en contextos controlados [24].

En contraste, los modelos basados en CNN, como YOLOv8, están diseñados para operar directamente sobre imágenes completas, permitiendo tanto la detección de objetos como su clasificación simultánea. Requieren mayor poder de cómputo y datos etiquetados más detallados, pero ofrecen una mayor precisión, autonomía y adaptabilidad a entornos complejos y cambiantes [9].

## 2.14 Retos actuales en la detección en tiempo real de lugares de aparcamiento

El diseño de sistemas de CV aplicados al monitoreo de estacionamientos en tiempo real enfrenta múltiples retos técnicos y operativos. Entre los más relevantes se encuentran las condiciones cambiantes de iluminación (día, noche, sombras), variaciones climáticas (lluvia, niebla), ángulos irregulares de cámara, oclusiones parciales (vehículos, árboles, postes) y la baja resolución de algunas cámaras de seguridad.

Además, los espacios de estacionamiento pueden variar considerablemente en tamaño, señalización y distribución según el entorno urbano o privado. Esto añade complejidad al proceso de generalización de los modelos, que deben ser entrenados para adaptarse a diferentes escenarios sin pérdida de precisión.

Otro desafío clave es la necesidad de respuesta en tiempo real, lo que exige modelos optimizados que puedan operar con baja latencia y en hardware limitado, como cámaras inteligentes o sistemas embebidos. Superar estos retos es esencial para lograr una solución práctica, confiable y aplicable en contextos reales.

A esto se suma la dificultad de contar con datasets de calidad y volumen suficiente, pues se requieren imágenes correctamente etiquetadas —con cajas delimitadoras precisas y categorías claras de ocupación— para entrenar modelos robustos. La recolección y anotación de estos datos implica costos considerables y, además, el entrenamiento con grandes volúmenes de imágenes demanda recursos de hardware especializados y largos tiempos de procesamiento, lo que representa un reto adicional para la implementación de estas soluciones en escenarios reales.

## 2.15 Aplicaciones reales y despliegues comerciales de sistemas inteligentes de estacionamiento

En años recientes, diversos proyectos y empresas han iniciado la implementación de sistemas inteligentes de estacionamiento basados en CV e IA. Ejemplos de ello incluyen plataformas como Parkalot, Dibsido, o Flex Parking, que utilizan cámaras de videovigilancia combinadas con algoritmos de detección con el fin de notificar en tiempo real la disponibilidad de lugares. Algunos despliegues a nivel municipal incluyen ciudades como Inglaterra, Nueva Zelanda, Nueva York y Singapur, que han adoptado sistemas de estacionamiento inteligente como parte de sus estrategias de ciudad inteligente (Smart City), integrando sensores, cámaras, redes IoT y plataformas de análisis centralizado [25].

Las tecnologías utilizadas varían, incluyendo sensores magnéticos, sensores ultrasónicos, LIDAR, y cada vez más, CV basada en cámaras por su bajo costo de mantenimiento, escalabilidad y precisión en condiciones variadas. Estos desarrollos demuestran que las soluciones basadas en CV con IA no solo son viables, sino que están comenzando a consolidarse como alternativas más sostenibles, escalables y flexibles que los sistemas físicos tradicionales [2].

Este contexto refuerza la relevancia y aplicabilidad de los enfoques propuestos en este proyecto, al alinearse con tendencias actuales del sector y ofrecer una solución realista y de bajo costo para el monitoreo eficiente de estacionamientos.

# 3. MARCO TEÓRICO

El siguiente apartado establece las bases teóricas que sustentan el enfoque adoptado en la investigación. Se introduce el concepto de IA y sus ramas principales, ML y DL, destacando su capacidad para automatizar tareas complejas mediante el reconocimiento de patrones en extensos conjuntos de datos. Se abordan conceptos esenciales como los métodos de entrenamiento y la estructura de ANNs, incluyendo variantes especializadas como CNNs y RNNs. También se profundiza en el uso de métricas para la evaluación de algoritmos de clasificación y reconocimiento de objetos, tanto gráficas y cuantitativas, así como en el análisis e interpretación de curvas de entrenamiento y validación. Por otro lado, se describen las principales bibliotecas utilizadas en Python para la formulación y aplicación de soluciones sustentadas en IA entre ellas OS, PIL, Matplotlib, Pandas, Scikit-learn, así como frameworks como YOLOv8, basado en PyTorch, y MLflow dentro del enfoque MLOps. Finalmente, se analiza la importancia del diseño de modelos capaces de operar en tiempo real, lo que resulta fundamental para la efectividad de sistemas inteligentes aplicados al monitoreo automatizado de estacionamientos.

## 3.1 IA

La IA se ha consolidado en calidad de concepto general que integra aplicaciones diseñadas para desempeñar tareas complejas que previamente requerían la participación de humanos. Este concepto suele emplearse indistintamente junto con los nombres de sus principales ramas: ML y DL. El fundamento esencial de la IA radica en imitar —y eventualmente superar— la manera en que las personas interpretan y reaccionan al entorno. En la actualidad, la IA se consolida como un factor esencial de la innovación. Impulsada por diversas técnicas de ML que identifican patrones en grandes volúmenes de datos, la IA aporta valor al permitir una comprensión más profunda de la información disponible y automatizar tareas demasiado complejas o repetitivas. Un ejemplo de esto es Netflix, que emplea ML para ofrecer recomendaciones personalizadas, lo que ayudó a incrementar su base de usuarios en más de un 25 % [6].

La adopción de la IA en múltiples sectores se ve impulsada por tres factores clave: la disponibilidad de capacidad computacional asequible y de alto desempeño, la existencia de extensos volúmenes de datos para entrenar modelos, y el hecho de que la aplicación de la IA ofrece una ventaja competitiva [6].

### 3.1.1 ML Aplicado

ML es la disciplina que entrena un sistema o programa informático para realizar tareas sin instrucciones explícitas. Los sistemas informáticos emplean algoritmos de ML para procesar volúmenes masivos de datos, identificar patrones y predecir resultados precisos en escenarios nuevos o desconocidos [24]. ML se enfoca en desarrollar sistemas capaces de aprender y optimizar su rendimiento con base en la información que gestionan. Conviene subrayar que no toda IA corresponde necesariamente a ML, aunque todo ML forma parte de la IA [6]. ML tradicional demanda una considerable intervención humana

mediante la ingeniería de características para obtener resultados efectivos. Sin embargo, dado que el propósito del ML es minimizar la participación humana, las técnicas de DL suprimen el requerimiento de asignar etiquetas manualmente los datos en cada etapa del proceso [24].

### 3.1.2 DL

DL es una rama dentro de ML que emplea estructuras algorítmicas conocidas como redes neuronales, inspiradas en el funcionamiento del cerebro humano. Se puede considerar a DL como una técnica avanzada dentro de ML [8]. Los enfoques de DL buscan automatizar tareas más complejas que, por lo general, demandan capacidades propias de la inteligencia humana [24]. ML detecta patrones en datos estructurados, como ocurre en sistemas de clasificación y recomendación. En contraste, las técnicas de DL resultan más apropiadas para manejar datos no estructurados, que requieren un nivel elevado de abstracción para extraer características relevantes. La elección entre ML y DL depende del tipo de datos a procesar [24].

### 3.1.3 Métodos de entrenamiento

ML comprende cuatro métodos principales de entrenamiento: supervisado, no supervisado, semisupervisado y por refuerzo. Además, existen otros enfoques como el TL y el aprendizaje autosupervisado [24]. En contraste, los algoritmos de DL emplean métodos de entrenamiento más complejos, entre los que se encuentran las CNNs, las redes neuronales recurrentes (RRNs), las GANs y los autocodificadores [24].

### 3.1.4 ANNs, CNNs y RNNs

Una red neuronal funciona como un grupo de nodos conectados. Cada nodo procesa información y la comparte, de forma similar a cómo funcionan las neuronas en nuestro cerebro. Los nodos se organizan en capas para trabajar con la información de entrada. Utilizan herramientas y métodos matemáticos potentes para aprender, descubrir patrones y hacer predicciones. Los enlaces entre estos puntos tienen pesos fijos. Estos pesos cambian a medida que la red aprende a realizar mejor su trabajo.

Cuando las redes neuronales examinan datos y detectan las respuestas correctas una y otra vez, modifican sus ponderaciones. Esto les ayuda a mejorar en ciertas tareas. A este método de aprendizaje lo llamamos entrenamiento de redes neuronales. Permite que redes neuronales como ANNs, CNNs y RNNs resuelvan problemas complejos, lo que las hace esenciales para los servicios de IA modernos [26].

Una ANN es el modelo básico para muchos tipos de redes neuronales. Reproduce mecanismos similares a los del cerebro humano. Las ANNs constan de capas de nodos conectados, conocidos como "neuronas". Estas neuronas gestionan los datos de entrada mediante ponderaciones, sesgos y funciones de activación. Las ANNs constan de una capa de entrada, múltiples capas ocultas y una capa de salida. Las ANNs se emplean comúnmente en tareas de categorización, regresión e identificación de patrones. [26].

Una CNN está pensada para trabajar con datos estructurados, especialmente imágenes. Una CNN utiliza capas convolucionales que utilizan filtros para encontrar patrones relevantes dentro de la información de entrada y generar mapas de características que representan la información extraída. Estos mapas ayudan a Identificar características como texturas, formas en los datos y bordes [26].

Una de las ventajas clave de las CNNs es el intercambio de parámetros (compartición de pesos) dentro de los filtros, lo que disminuye significativamente el número de parámetros en comparación con una ANN tradicional. Esta característica permite que las CNNs trabajen de forma más eficiente con datos de imágenes y videos. Las CNN son muy buenas para detectar patrones en imágenes. Esta capacidad las hace ideales para trabajar con información visual. Entre los casos de uso más habituales de las CNN se encuentran el reconocimiento y la categorización de imágenes, así como la identificación de elementos, como ocurre en identificación biométrica facial o en el análisis de imágenes médicas para diagnósticos asistidos por computadora [26].

Una red neuronal recurrente (RNN) convierte una secuencia de datos de entrada en una secuencia de salida definida, utilizando múltiples capas interconectadas [2]. Estos datos secuenciales pueden ser palabras, frases o series temporales, en las que cada elemento se relaciona con los demás según reglas semánticas y sintácticas complejas. Conceptualmente, una RNN emula la manera en que los humanos interpretan secuencias de información, como al traducir texto de un idioma a otro.

Las RNN se componen por neuronas organizadas en capas de entrada, salida y capas ocultas, donde ocurre la gestión y evaluación de los datos. Actualmente, fueron reemplazadas en gran medida por modelos más avanzados, como los transformadores y los modelos de lenguaje de gran tamaño (LLMs), los cuales ofrecen una mayor eficiencia y precisión para el procesamiento de secuencias extensas [27]. No obstante, las RNN siguen siendo una base importante en el desarrollo de modelos de PLN y tareas de predicción temporal.

Es importante destacar que, mientras las CNNs están diseñadas para analizar datos espaciales, por ejemplo, imágenes y videos mediante capas convolucionales y de agrupamiento, las RNNs se especializan en identificar relaciones a largo plazo en secuencias de datos, lo que las hace complementarias pero distintas en su arquitectura y aplicación [27].

## 3.2 Métricas de Evaluación para Modelos de Categorización y Detección de Objetos

Con el fin de validar la funcionalidad de los modelos de ML, particularmente en problemas de clasificación y detección de objetos, es fundamental utilizar métricas estandarizadas y visualizaciones que permitan comprender no solo el nivel de aciertos, sino también los posibles patrones de error.
Existen diversas métricas estándar y gráficos utilizados con el propósito de medir el rendimiento de los modelos de detección de objetos [33].

### 3.1.1 Principales Métricas gráficas

- Matriz de confusión: ofrece una conceptualización gráfica de cómo un algoritmo de ML realiza fallos sistemáticos en la categorización de cada tipo de objeto.

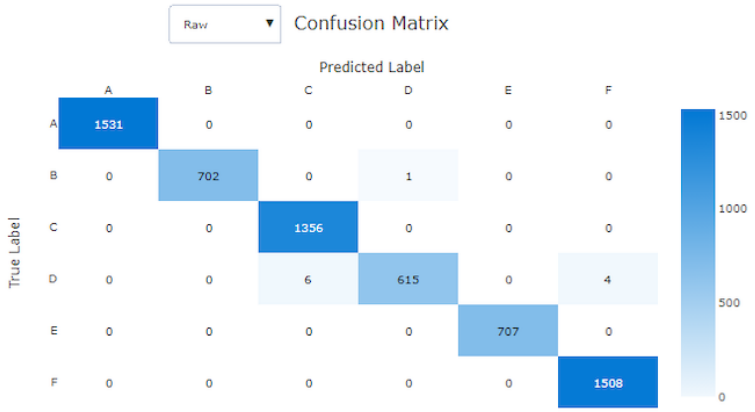



**Figura 10 Matriz de confusión: Ejemplo de clasificación multiclase con valores altos en diagonal principal y bajos fuera de ella, indicando buen rendimiento del modelo [33]**

- Curva de característica operativa del receptor (ROC): La ROC define la relación existente entre la tasa de verdaderos positivos (TPR) y la tasa de falsos positivos (FPR) conforme se modifica el umbral de decisión.

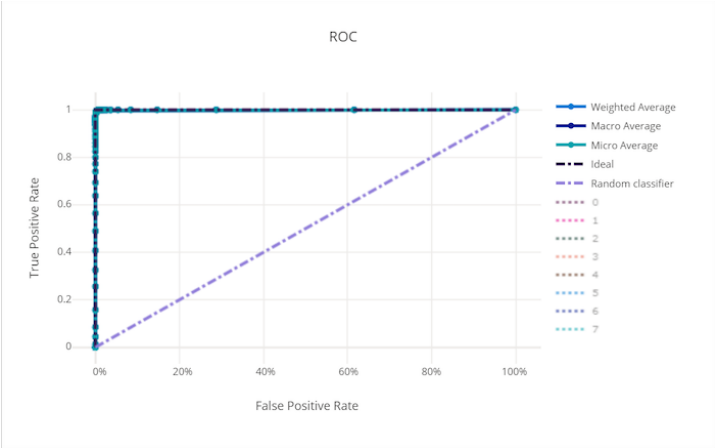



**Figura 11 Ejemplo de curva ROC de un modelo bien entrenado: Representación gráfica mostrando alta sensibilidad y baja FPR, con curva situada cerca del vértice superior izquierdo que refleja excelente discriminación**

- Curva de precisión-coincidencia (PR): Ilustra la correlación entre la precisión y el grado de correspondencia en distintos puntos de referencia.

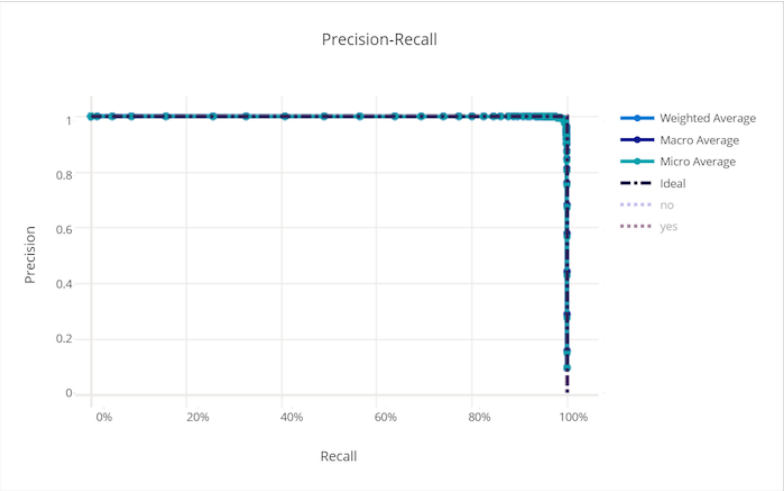



**Figura 12 Curva de precisión-recall (PR) de un modelo bien entrenado: Gráfica de precisión versus recall mostrando la correlación entre exactitud de predicciones positivas y capacidad de detectar todos los casos positivos en diferentes umbrales de clasificación**

- Curva de elevación: Una curva de elevación indica la superioridad de un modelo en relación con un modelo aleatorio. Resulta útil evaluar el desempeño de un modelo de identificación de elementos considerando la cantidad de elementos correctamente identificados.

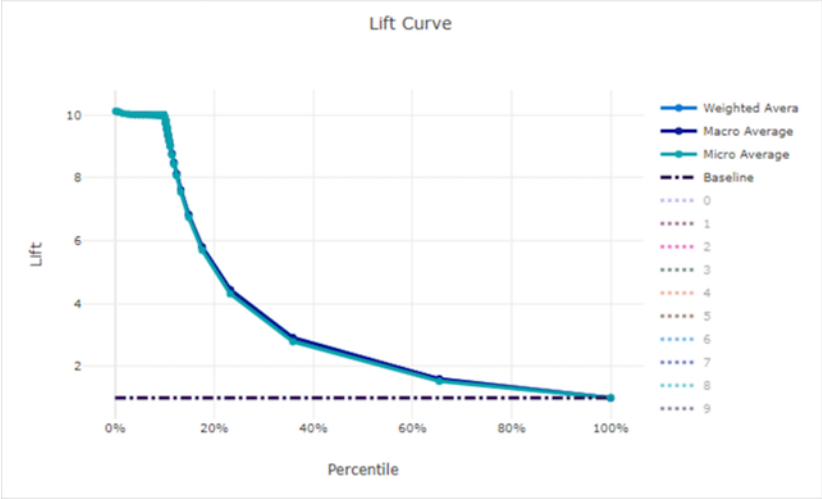



**Figura 13 Ejemplo de curva de elevación para modelo de alto rendimiento: Representación con elevación inicial de 10x que supera significativamente la línea base aleatoria, demostrando capacidad superior de detección en los casos más probables**

- Curva de calibración: La curva de calibración ilustra la confiabilidad de un modelo en sus predicciones, mostrando la relación entre el nivel de confianza asignado y la proporción real de muestras positivas correspondientes a cada nivel de confianza [33].

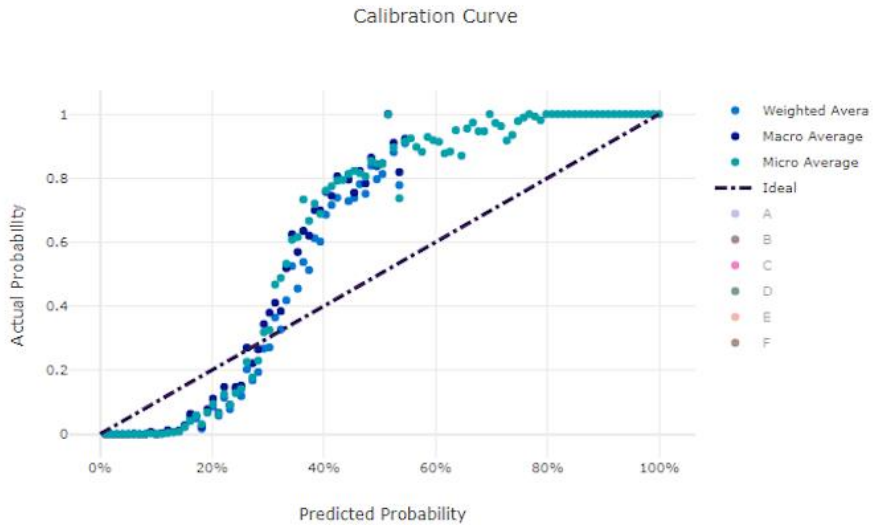



**Figura 14 Ejemplo de curva de calibración para modelo de alto rendimiento: Representación de la relación entre confianza asignada y proporción real de muestras positivas, con puntos de datos siguiendo estrechamente la diagonal ideal**

### 3.1.2 Principales Métricas cuantitativas

**Precisión (Precision, en inglés):** La métrica de precisión se emplea para determinar cuántos de los valores categorizados como positivos son en realidad positivos [34].

$$Precisión = TP / (TP + FP)$$

**Ecuación 8 Fórmula para la precisión basada en verdaderos positivos y falsos positivos**

**Recuperación (Recall, en inglés):** Se emplea para determinar la cantidad correcta de valores positivos clasificados [34].

$$Recuperación = TP / (TP + FN)$$

**Ecuación 9 Cálculo de la recuperación basada en verdaderos positivos y falsos negativos (FN)**

**F1 Score:** Esta es una medida frecuentemente empleada en situaciones con un conjunto de datos desbalanceado. Esta medida fusiona la precisión y el recall para conseguir un valor mucho más imparcial.

$$F1 = 2 * \left(\frac{recall * precision}{recall + precision}\right)$$

**Ecuación 10 Cálculo del F1-score como media armónica entre precisión y recuperación**

**IoU:** El índice de superposición (IoU) es un instrumento de evaluación aplicada para cuantificar la precisión de un detector de elementos sobre un conjunto específico de muestras. Todo método algorítmico que genere cajas delimitadoras como salida puede ser sometido a evaluación utilizando esta métrica [35].

$$J\ A,B = \frac{|A\ \cap\ B|}{|A\ \cup\ B|}$$

**Ecuación 11 Definición del índice de superposición (IoU)**

**Promedio de Precisión Media (Map, por sus siglas en inglés):** En actividades de identificación de objetos, el Map es visto como una de las medidas más sólidas. Se determina como la media de la precisión en distintos grados de recuperación y resulta especialmente útil en la valoración de modelos en grupos de datos de diversas clases [35].

Las técnicas basadas en DL, como Faster R-CNN, YOLO y SSD, son altamente efectivos para la detección de objetos, ya que aprenden características complejas de grandes volúmenes de datos y generalizan bien en escenarios diversos, superando a los métodos tradicionales como la detección de bordes y algoritmos basados en reglas, en precisión y recuperación. Esto debido a que las técnicas tradicionales dependen de características predefinidas y menos adaptativas, lo que limita su rendimiento en escenarios del mundo real, donde la variabilidad en las condiciones como imágenes con ruido o sombras puede afectar su efectividad.

### 3.1.3 Curvas de Entrenamiento y Validación

#### 3.1.3.1 Curvas de Entrenamiento

Durante la fase de aprendizaje de un esquema modelado encargado de la identificación de elementos, como los basados en CNN, es fundamental monitorizar el progreso mediante curvas de entrenamiento. Estas gráficas permiten visualizar la evolución de la función de pérdida (loss function, en inglés) a medida que se ajustan sus valores paramétricos.
Dentro de una curva típica del proceso de entrenamiento:

- El eje X refleja la cantidad de épocas (epochs, en inglés), que corresponde a las iteraciones completas sobre todo la colección de datos destinada al entrenamiento.
- El eje Y muestra el valor de la pérdida (loss, en inglés), que calcula la diferencia existente entre las predicciones generadas por el modelo y los datos reales.

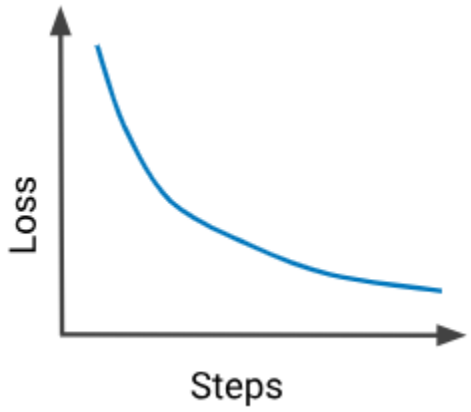



**Figura 15 Curva de pérdida ideal.**

Los valores asociados a la pérdida suelen a separarse por componentes, por ejemplo:

- Box Loss: error asociado a la precisión con que las cajas delimitadoras se ajustan a los objetos.
- Classification Loss: error relacionado con la clasificación correcta de cada objeto detectado.

En cada época, la red calibra los pesos de sus capas a fin de disminuir estas pérdidas. Una tendencia descendente estable en la curva de entrenamiento señala que la red está extrayendo patrones significativos de los datos y ajustando adecuadamente sus parámetros [28].

#### 3.1.1.1 Curvas de Validación

Paralelamente, se utilizan las curvas de validación a fin de cuantificar la capacidad del modelo aplicando lo aprendido a datos nuevos que no formaron parte del entrenamiento.
Estas gráficas tienen la misma estructura:

- El eje de las abscisas corresponde a las épocas.
- El eje de las ordenadas indica la pérdida de validación (validation loss, en inglés), determinada con datos de validación independiente.

Mientras que la curva correspondiente al entrenamiento señala la adaptación del modelo con base a los datos conocidos, la curva de evaluación revela su desempeño en datos nuevos. Idealmente, ambas curvas deberían seguir una trayectoria descendente similar. En caso de que la curva correspondiente al entrenamiento mantenga una caída continua y la curva de validación inicie un ascenso, esto puede indicar un sobreajuste (overfitting, en inglés). En dicha situación, el modelo está aprendiendo detalles específicos o ruido de los datos de entrenamiento que presenta dificultades para generalizar eficazmente a datos externos [28].

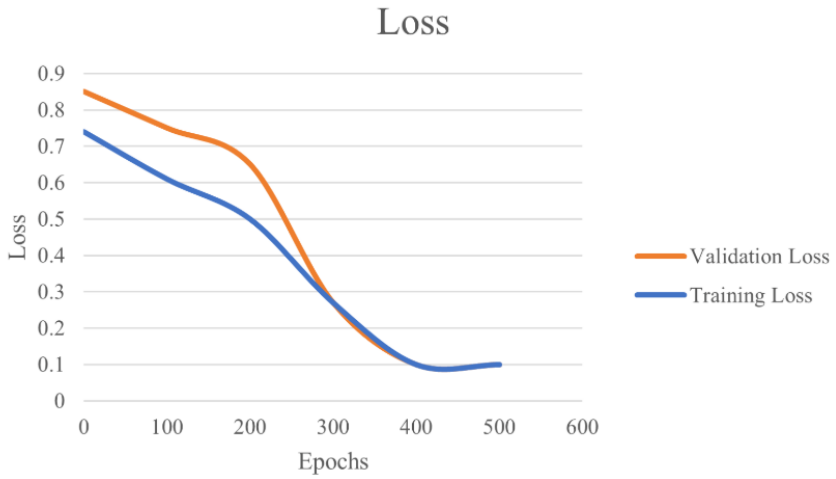



**Figure 16 Curva de validación que evidencia un desempeño equilibrado: sin sobreajuste ni subajuste.**

#### 3.1.1.2 Interpretación

Las curvas de entrenamiento y validación permiten visualizar cómo evoluciona el desempeño de un modelo a lo largo de cada época (epoch). Comprender su forma y sus puntos clave es esencial para localizar errores comunes como el sobreajuste (overfitting) o el subajuste (underfitting).
Interpretación de curva de entrenamiento:

- Una curva descendente y suave señala que el modelo está optimizando la reducción de su error interno de forma constante.
- Si la curva se aplana en un valor elevado y no sigue bajando, es señal de subajuste, pues el modelo no tiene la complejidad suficiente o el entrenamiento no es suficiente para extraer patrones útiles.
- Si la pérdida desciende demasiado rápido y se vuelve casi cero, podría significar que el modelo retiene el conjunto de entrenamiento, lo que requiere monitorear la curva de validación para confirmar o descartar sobreajuste.

Interpretación de curva de validación:

- Idealmente, la pérdida de validación debe descender junto con la de entrenamiento. Esto sugiere que el modelo aprende de forma generalizable.
- Si la pérdida de validación se estabiliza o incrementa a medida que la función de pérdida durante el entrenamiento sigue descendiendo, el modelo inicia a retener los datos, perdiendo así su capacidad de generalizar: este es el patrón clásico de sobreajuste.
- Si la curva muestra altas oscilaciones o picos, podría indicar:
  - Datos ruidosos o inconsistentes
  - Valores elevados en hiperparámetros, como una tasa de aprendizaje excesivamente alta
  - Ausencia de técnicas de regularización, tales como *early stopping* o *dropout* [28]

## 3.2 Python y sus bibliotecas principales

Python se posiciona como uno de los lenguajes de programación más populares en el desarrollo de aplicaciones web, software, análisis de datos y ML. Su amplia adopción se explica por su eficiencia, facilidad para entrenar y capacidad para ejecutarse en múltiples plataformas. A continuación, se presentan algunas de las bibliotecas más destacadas para proyectos relacionados con ML, DL y detección de objetos [29].

### 3.2.1 OS

El módulo os está integrado dentro de la biblioteca estándar de Python. Se emplea para gestionar operaciones relacionadas con el sistema operativo, entre los que destacan la administración de rutas, nombres de archivos, carpetas y permisos. Es especialmente útil para automatizar flujos de trabajo que requieren explorar grandes volúmenes de datos almacenados en múltiples directorios [30].

```
import os
```

```
# Obtener la lista de imágenes en un directorio
carpeta_imagenes = 'dataset/images/train'
imagenes = [f for f in os.listdir(carpeta_imagenes) if f.endswith('.jpg')]
print(imagenes[:5])
```

### 3.2.2 Pillow (PIL)

Pillow, sucesor de la biblioteca PIL (Python Imaging Library), es una de las bibliotecas más populares para manipular, editar y procesar imágenes. Permite realizar operaciones como redimensionamiento, recorte, conversión de formatos, anotaciones y creación de visualizaciones que ayudan a interpretar resultados de modelos de visión por computadora [31].

```
from PIL import Image, ImageDraw

# Abrir imagen
imagen = Image.open('dataset/images/train/ejemplo.jpg')

# Dibujar un cuadro delimitador (bounding box)
draw = ImageDraw.Draw(imagen)
draw.rectangle([60, 60, 200, 200], outline="blue", width=2)

# Mostrar imagen anotada
imagen.show()
```

### 3.2.3 Matplotlib

Matplotlib se refiere a la biblioteca destinada a la presentación visual de datos, la cual es muy utilizada en los ámbitos científico y de ingeniería. Permite crear gráficos tanto estáticos como dinámicos, tales como histogramas, curvas de aprendizaje, matrices de confusión y gráficos de dispersión. Constituye una herramienta esencial para examinar tendencias, monitorear el desarrollo de métricas en el transcurso del entrenamiento y mostrar resultados de manera clara y comprensible [32].

```
import matplotlib.pyplot as plt
import pandas as pd

# Leer resultados simulados de entrenamiento
```

```
resultados = pd.DataFrame({
    'epoch': [1, 2, 3, 4, 5],
    'loss': [0.8, 0.6, 0.4, 0.3, 0.2]
})

# Graficar pérdida por época
plt.plot(resultados['epoch'], resultados['loss'])
plt.title('Curva de pérdida')
plt.xlabel('Época')
plt.ylabel('Pérdida')
plt.grid(True)
plt.show()
```

### 3.2.4 Pandas

Pandas es una biblioteca diseñada para analizar y gestionar datos en formato tabular. Permite estructurar, limpiar, filtrar y transformar datos en formatos como CSV o Excel, facilitando tareas de preprocesamiento antes de alimentar modelos de ML. También se utiliza para organizar y examinar resultados obtenidos tras la ejecución de experimentos [33].

```
import pandas as pd

# Leer etiquetas de prueba simuladas
df = pd.read_csv('resultados.csv')

print(df.head())
```

### 3.2.5 Scikit-learn

Scikit-learn es una de las bibliotecas más reconocidas para implementar algoritmos de ML de manera sencilla y modular. Proporciona recursos para funciones tales como clasificación, agrupamiento (clustering), regresión y disminución de dimensionalidad. Además, integra funciones para la partición de conjuntos de datos, el cálculo de indicadores de rendimiento — entre las que se incluyen recall, precisión y matriz de confusión— y la validación de modelos mediante técnicas como la validación cruzada [34].

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Etiquetas reales y predichas de ejemplo
```

```
y_true = [0, 1, 1, 0, 1, 1, 0]
y_pred = [0, 1, 0, 0, 1, 1, 1]

# Matriz de confusión
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

### 3.2.6 Ultralytics YOLOv8 (basado en PyTorch)

PyTorch opera como un entorno de desarrollo de código abierto orientado a desarrollo y entrenamiento de ANNs mediante operaciones con tensores. Es muy valorado por su flexibilidad, facilidad de depuración y compatibilidad con GPUs para acelerar cálculos de gran volumen. Frameworks como Ultralytics YOLOv8 se basan en PyTorch para implementar CNN optimizadas para operaciones de identificación y localización de objetos de manera inmediata, combinando precisión y eficiencia computacional [35].

```
from ultralytics import YOLO

# Cargar un modelo YOLOv8 preentrenado (por ejemplo, YOLOv8n)
model = YOLO('yolov8n.pt')
```

### 3.2.7 MLflow (MLOps)

MLflow es un entorno diseñado para la gestión de ciclos de vida de enfoques modelados de ML alineados con la filosofía MLOps (ML Operations). Su función principal es registrar y rastrear experimentos, almacenar métricas y parámetros de entrenamiento, y gestionar versiones de modelos de manera estructurada y reproducible. Esta práctica es esencial para garantizar la trazabilidad y la eficiencia en proyectos de IA que requieren múltiples iteraciones y evaluaciones comparativas [36].

```
import mlflow

# Simular registro de parámetros de entrenamiento
with mlflow.start_run():
    mlflow.log_param("epochs", 25)
    mlflow.log_param("batch_size", 16)
```

```
    mlflow.log_metric("accuracy", 0.92)
```

## 3.3 Modelos en tiempo real

En aplicaciones de IA, los modelos en tiempo real desempeñan un papel esencial cuando se requiere procesar datos y entregar resultados de forma inmediata. Una canalización (pipeline) en tiempo real debe ser capaz de recopilar y preparar todos los datos de entrada de manera eficiente y continua. En muchos escenarios, los datos se obtienen a través de la llamada carga útil (payload), que consiste en que, tras la acción de un usuario o la activación de un sensor, los datos se empaquetan y se envían directamente al modelo mediante una solicitud o llamada de inferencia [37].

Sin embargo, la simple adquisición de datos no basta: los datos de entrada suelen requerir preprocesamiento para normalizar dimensiones, escalar valores o transformar formatos antes de ser analizados por la red neuronal. Este paso debe ejecutarse con mínima latencia para no comprometer la capacidad de respuesta del sistema. Los modelos diseñados para procesamiento en tiempo real enfrentan el reto de equilibrar precisión y velocidad de ejecución. Para lograrlo, es común aplicar dos enfoques complementarios: optimizar la infraestructura con hardware de alto rendimiento (por ejemplo, GPU especializadas) y diseñar arquitecturas de red más livianas y eficientes, capaces de ejecutar múltiples inferencias por segundo sin sacrificar significativamente la calidad de las predicciones [37].

En este contexto, las CNN optimizadas como YOLOv8 destacan como soluciones robustas destinadas a la localización de objetos con respuesta en tiempo real, pues segmenta la imagen en diferentes regiones y estima las cajas delimitadoras (bounding boxes) junto con la clasificación de objetos en una sola pasada de la red, lo que reduce drásticamente la latencia. Modelos recientes, como YOLOv8, aportan mejoras significativas a la arquitectura subyacente —como capas convolucionales más profundas, mecanismos de normalización y técnicas de anclaje adaptativo— que permiten alcanzar mayores tasas de detección por segundo (FPS) manteniendo una alta precisión [20].

# 4. DESARROLLO Y METODOLOGÍA

Se detalla la selección y uso de las colecciones de datos públicos empleados destinados al entrenamiento y valoración de los modelos. Para los modelos tradicionales de ML, como K-NN, regresión logística y árboles de decisión, se empleó un conjunto de datos específico basado en fotografías recortadas además de etiquetadas, adecuadas para clasificación supervisada con extracción manual de características. Se describen las técnicas de preprocesamiento aplicadas a los datos y el diseño de los modelos, destacando la sencillez, bajo costo computacional y utilidad de estos enfoques como base para comparar con métodos más avanzados.

La elección de los datasets utilizados en este proyecto se basó en su disponibilidad pública, su estructura organizada y su adecuación a los objetivos específicos de cada enfoque metodológico. Para diseñar y evaluar modelos tradicionales de ML — como K-NN, árboles de decisión y regresión logística — se empleó la colección de datos disponible en el portal de Kaggle: Parking Lot Dataset [43]. El presente conjunto de datos está integrado por fotografías individuales de espacios de estacionamiento ya recortados, etiquetados como *ocupado* o *vacío*. Esta estructura resulta adecuada para tareas de clasificación supervisada, ya que permite aplicar modelos tradicionales directamente sobre las imágenes, con extracción manual de características o representaciones simplificadas. Estos enfoques son simples de interpretar, de bajo costo computacional y útiles como línea base para comparar soluciones más complejas.

No obstante, la mayor restricción de este método es que las imágenes están presegmentadas, lo que impide su uso en tareas de detección automática sobre imágenes completas, como las captadas en tiempo real por cámaras de videovigilancia. Para la etapa de DL, se utilizó el dataset PKLot, que contiene imágenes reales de estacionamientos capturadas desde cámaras de seguridad, junto con anotaciones en formato compatible con YOLOv8. Este modelo, basado en CNN, permite realizar detección y clasificación simultánea directamente sobre imágenes completas, sin necesidad de segmentación previa [9].

YOLOv8 resalta por su eficiencia al procesar imágenes en tiempo real manteniendo un balance de precisión con velocidad, lo cual lo convierte en una opción óptima para escenarios dinámicos como estacionamientos urbanos. Además, su compatibilidad con formatos estándar y su integración con librerías como OpenCV facilitan su implementación práctica y futura escalabilidad en sistemas embebidos o de bajo costo.

## 4.1 Modelos de ML

### 4.1.1 Colección de datos

El dataset aplicado a abordar la problemática relacionada con la categorización de imágenes de espacios de estacionamiento fue parking-lot-dataset [43].
Este conjunto de datos incluye imágenes panorámicas a color capturadas desde una vista aérea de estacionamientos completos, así como imágenes segmentadas de espacios individuales de estacionamiento. Las imágenes están etiquetadas como vacías u ocupadas y, además, clasificadas según las condiciones climáticas: nublado, lluvioso o soleado.

La robustez del conjunto radica en su gran volumen, compuesto por miles de imágenes capturadas en distintas fechas y mediante tres tipos diferentes de cámaras. Aunque el tamaño de las imágenes varía, en el caso de las segmentadas, la mayoría tiene una resolución inferior a 64x64 píxeles.

Ejemplo de imágenes:

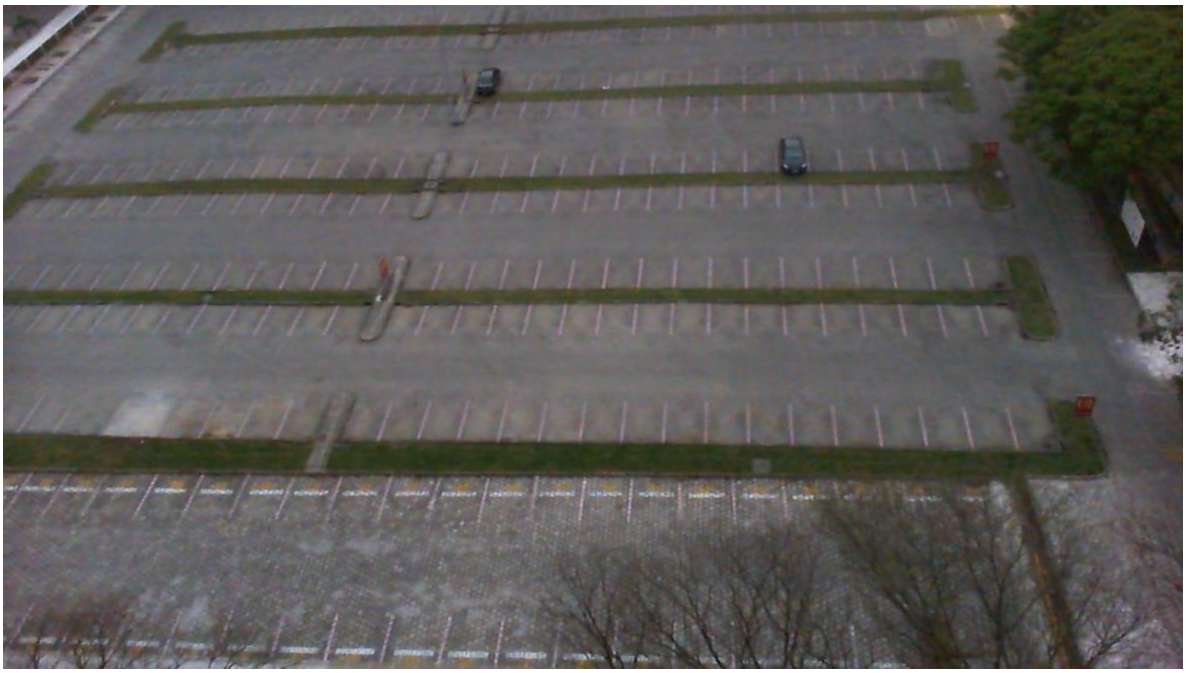


**Figura 17 Imagen aérea de estacionamiento completo del dataset de Kaggle: Vista panorámica a color capturada el 12 de septiembre de 2012 mostrando espacios de estacionamiento organizados en filas con algunos vehículos estacionados y áreas verdes circundantes**

**Figura 18 Imagen segmentada de espacio de estacionamiento ocupado del dataset: Fotografía individual de plaza con vehículo blanco capturada el 12 de septiembre de 2012, etiquetada como 'ocupado' para entrenamiento de clasificación binaria**

**Figura 19 Imagen segmentada de espacio de estacionamiento vacío del dataset: Fotografía individual de plaza libre capturada el 03 de marzo de 2013, etiquetada como 'vacío' para entrenamiento de clasificación binaria de ocupación**

### 4.1.2 Preprocesamiento de datos

La primera etapa fue elegir imágenes del dataset provenientes de las tres cámaras diferentes y de los tres tipos de clima. Se aseguró que las imágenes seleccionadas no fueran repetidas ni similares, garantizando así una mayor diversidad en los datos. En total, se procesaron 9,000 imágenes con clases balanceadas: 4,500 correspondientes a espacios ocupados y 4,500 a espacios vacíos.

Todas las imágenes fueron redimensionadas a 64x64 píxeles utilizando el método Image.LANCZOS, asegurando una alta calidad en la transformación y conservando el formato RGB. Posteriormente, los valores de los píxeles fueron normalizados. Con el fin de optimizar el desempeño del modelo, se realizó un shuffle (mezcla aleatoria) de los datos antes del entrenamiento.

```
import os
import numpy as np
from PIL import Image

IMAGE_SIZE = (64, 64)
```

```python
def preprocesar_imagen(ruta_imagen, tamaño_objetivo=(64, 64)):
    try:
        img = Image.open(ruta_imagen).convert("RGB")
        img = img.resize(tamaño_objetivo, Image.LANCZOS)
        img_array = np.array(img) / 255.0
        return img_array.flatten()
    except Exception as e:
        print(f"Error al procesar la imagen {ruta_imagen}: {e}")
        return None
```

```python
X = np.array(X)
y = np.array(y)

X, y = shuffle(X, y, random_state=42)
```

### 4.1.3 Definición del modelo

#### 4.1.3.1 Modelo de Clasificación Supervisado

##### 4.1.3.1.1 Modelo de Regresión logística

```python
from sklearn.linear_model import LogisticRegression


model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
evaluacion_del_modelo(y_test, y_pred)
```

##### 4.1.3.1.2 Modelo KNN

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,
accuracy_score, confusion_matrix

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

##### 4.1.3.1.3 Modelo de árbol de decisión

```python
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

tree = DecisionTreeClassifier(random_state=42)

tree.fit(X_train, y_train)

y_pred_tree = tree.predict(X_test)

evaluacion_del_modelo(y_test, y_pred_tree)
```

#### 4.1.3.2 Modelo de Clasificación No Supervisado

Se elegirán algoritmos de ML adecuados para la clasificación de imágenes. Se buscará utilizar lo visto en clase como, SGDClassifier, GridSearchCV, así como PCA.

```
from sklearn.decomposition import PCA


pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

print(f"Dimensionalidad original: {X_train.shape}")
print(f"Dimensionalidad después de PCA: {X_train_pca.shape}")
```

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, homogeneity_score,
completeness_score, v_measure_score

# Crear el modelo KMeans (con 2 clusters: 'Empty' y 'Occupied')
kmeans = KMeans(n_clusters=2, random_state=42)

kmeans.fit(X_train_pca)
y_pred_test_pca = kmeans.predict(X_test_pca)
```

## 4.2 Preparación de Datos para Entrenamiento del Modelo YOLOv8

### 4.2.1 Selección y Evaluación de Datasets

Para el entrenamiento del modelo se realizaron pruebas exhaustivas con el objetivo de conformar un conjunto de datos robusto, variado y de alta calidad. El proceso de selección incluyó la evaluación y

comparación de múltiples bases de datos públicas relacionadas con estacionamientos, priorizando aquellas que cumplieran con los siguientes criterios:

- Imágenes capturadas por cámaras de seguridad en escenarios reales
- Vistas panorámicas con distintas perspectivas
- Variaciones climáticas (soleado, nublado, lluvioso)
- Condiciones de iluminación diversas
- Anotaciones precisas y consistentes

#### 4.2.1.1 Generación Automática de Bounding Boxes

Durante la fase inicial se intentó generar automáticamente las cajas delimitadoras (bounding boxes) para los espacios de estacionamiento utilizando modelos de detección de objetos preentrenados. Se implementó un enfoque basado en Faster R-CNN de Detectron2 para procesar imágenes de dos categorías ('Free' y 'Occupied') y generar automáticamente las anotaciones correspondientes.

```
import os
import cv2
import xml.etree.ElementTree as ET
from detectron2 import model_zoo
from detectron2.config import get_cfg
from detectron2.engine import DefaultPredictor
from tqdm import tqdm

# Este script procesa imágenes de dos categorías ('Free' y 'Occupied')
utilizando el modelo Faster R-CNN de Detectron2 para la detección de
objetos.
# Las anotaciones se guardan como archivos XML en formato Pascal VOC con
dimensiones 160x160

def save_to_voc_format(image_name, predictions, save_path):
    annotation = ET.Element("annotation")
    folder = ET.SubElement(annotation, "folder")
    folder.text = "train"

    filename = ET.SubElement(annotation, "filename")
    filename.text = image_name

    size = ET.SubElement(annotation, "size")
    ET.SubElement(size, "width").text = "160"
    ET.SubElement(size, "height").text = "160"
    ET.SubElement(size, "depth").text = "3"

    for pred in predictions:
        xmin, ymin, xmax, ymax = map(int, pred['bbox'])
```

```
        xmin = max(0, xmin)
        ymin = max(0, ymin)
        xmax = min(160, xmax)
        ymax = min(160, ymax)

        obj = ET.SubElement(annotation, "object")
        bndbox = ET.SubElement(obj, "bndbox")
        ET.SubElement(bndbox, "xmin").text = str(xmin)
        ET.SubElement(bndbox, "ymin").text = str(ymin)
        ET.SubElement(bndbox, "xmax").text = str(xmax)
        ET.SubElement(bndbox, "ymax").text = str(ymax)

    tree = ET.ElementTree(annotation)
    xml_file = os.path.join(save_path, image_name.replace('.jpg', '.xml'))
    tree.write(xml_file)

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-
Detection/faster_rcnn_R_50_FPN_3x.yaml"))
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
Detection/faster_rcnn_R_50_FPN_3x.yaml")
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 2
cfg.MODEL.DEVICE = "cpu"

predictor = DefaultPredictor(cfg)

CONFIDENCE_THRESHOLD = 0.3

categories = ['Free', 'Occupied']

for category in categories:
    input_dir = f'./{category}'
    output_dir = f'./annotations/{category}'
    os.makedirs(output_dir, exist_ok=True)

    img_names = [img_name for img_name in os.listdir(input_dir) if
img_name.endswith(".jpg")]

    for img_name in tqdm(img_names, desc=f"Procesando {category}",
unit="imagen"):
        img_path = os.path.join(input_dir, img_name)
        img = cv2.imread(img_path)

        if img is None:
            print(f"❌ No se pudo leer la imagen: {img_path}")
```

```
            continue

        save_img_path = os.path.join(output_dir, img_name)
        cv2.imwrite(save_img_path, img)

        img_resized = cv2.resize(img, (160, 160))
        outputs = predictor(img_resized)
        instances = outputs["instances"]
        scores = instances.scores.numpy()
        pred_boxes = instances.pred_boxes.tensor.numpy()

        predictions = [{'bbox': box} for score, box in zip(scores,
pred_boxes) if score >= CONFIDENCE_THRESHOLD]

        save_to_voc_format(img_name, predictions, output_dir)

print("Imágenes originales guardadas, anotaciones en 160x160 generadas.")
```

**Figura 20 Ejemplo de Conjuntos de datos con etiquetas generadas.**

Sin embargo, este enfoque de generación automática de bounding boxes presentó varios problemas significativos:

1. **Baja precisión en la detección:** El modelo preentrenado no estaba específicamente optimizado para detectar espacios de estacionamiento, resultando en detecciones imprecisas o faltantes.
2. **Inconsistencia en las anotaciones:** Las predicciones variaban considerablemente entre imágenes similares, generando inconsistencias en el dataset.
3. **Pérdida de contexto espacial:** El modelo no comprendía la estructura específica de los estacionamientos, confundiendo objetos como vehículos parcialmente visibles o sombras.

4. **Problemas de escala:** Las dimensiones fijas de 160x160 píxeles no capturaban adecuadamente los detalles necesarios para una detección precisa.

Debido a estas limitaciones, se decidió abandonar este enfoque y buscar datasets que ya contuvieran etiquetas de alta calidad previamente validadas.

#### 4.2.1.2 Conversión de Formatos de Anotación

Adicionalmente, se realizaron intentos de conversión de anotaciones desde formatos como Pascal VOC y COCO hacia el formato YOLOv8. Sin embargo, este proceso también presentó problemas de confiabilidad:

**Validación y Control de Calidad**

- **Pérdida de precisión:** La conversión automática entre formatos a menudo resultaba en coordenadas imprecisas o desalineadas
- **Problemas de normalización:** Las diferencias en sistemas de coordenadas causaban errores en la transformación de bounding boxes
- **Pérdida de información contextual:** Algunos metadatos importantes se perdían durante la conversión
- **Errores de mapeo de clases:** Las categorías originales no siempre se correspondían exactamente con las clases requeridas (vacío/ocupado)

Por estas razones, se decidió priorizar la búsqueda de datasets que ya contuvieran etiquetas en formato YOLOv8, evitando así la necesidad de realizar anotaciones manuales o conversiones automáticas problemáticas.

Durante el proceso de evaluación surgieron inconsistencias en algunas anotaciones, tales como:

- Coordenadas fuera del rango válido de la imagen
- Imágenes sin su archivo de etiqueta correspondiente
- Registros duplicados o corruptos
- Formato incorrecto de las anotaciones

Para garantizar la calidad de los datos, se desarrollaron scripts adicionales para verificar la correspondencia y validez de cada par imagen-etiqueta:

```
import os
import cv2
import xml.etree.ElementTree as ET
import random
import matplotlib.pyplot as plt

# Este script imprime las imágenes a 160x160 con las cajas delimitadoras
plasmadas en sus xml.

# Función para cargar las anotaciones del archivo XML
def load_annotations(xml_path):
    tree = ET.parse(xml_path)
    root = tree.getroot()
```

```
    bboxes = []
    for obj in root.findall('object'):
        bbox = obj.find('bndbox')
        xmin = int(bbox.find('xmin').text)
        ymin = int(bbox.find('ymin').text)
        xmax = int(bbox.find('xmax').text)
        ymax = int(bbox.find('ymax').text)
        bboxes.append([xmin, ymin, xmax, ymax])

    return bboxes

# Función para visualizar la imagen original con las cajas ajustadas a
160x160
def visualize_image_with_bboxes(img_path, xml_path):
    # Cargar la imagen original
    img = cv2.imread(img_path)

    # Obtener las cajas desde el XML (ya están ajustadas a 160x160)
    bboxes = load_annotations(xml_path)

    # Redimensionar la imagen a 160x160 para visualizar
    img_resized = cv2.resize(img, (160, 160))

    # Dibujar las cajas en la imagen redimensionada
    for bbox in bboxes:
        xmin, ymin, xmax, ymax = bbox

        # Dibujar el rectángulo en la imagen redimensionada
        cv2.rectangle(img_resized, (xmin, ymin), (xmax, ymax), (0, 0, 255),
2)

    # Convertir BGR a RGB para mostrar correctamente con matplotlib
    img_rgb = cv2.cvtColor(img_resized, cv2.COLOR_BGR2RGB)

    # Mostrar la imagen
    plt.imshow(img_rgb)
    plt.axis('off')  # Quitar los ejes
    plt.show()

# Rutas de las carpetas
free_dir = 'annotations/Free'
occupied_dir = 'annotations/Occupied'
xml_dir_free = 'annotations/Free'
xml_dir_occupied = 'annotations/Occupied'
```

```
# Función para cargar y mostrar imágenes de ambas carpetas
def visualize_random_images(free_dir, occupied_dir, xml_dir_free,
xml_dir_occupied, num_images=10):
    # Obtener todas las imágenes disponibles en cada categoría
    free_images = [img for img in os.listdir(free_dir) if
img.endswith('.jpg')]
    occupied_images = [img for img in os.listdir(occupied_dir) if
img.endswith('.jpg')]

    # Si hay más de `num_images` imágenes, seleccionar aleatoriamente, sino
mostrar todas las imágenes disponibles
    free_images_to_show = random.sample(free_images, min(num_images,
len(free_images)))
    occupied_images_to_show = random.sample(occupied_images, min(num_images,
len(occupied_images)))

    # Mostrar imágenes de "Free"
    print(f"Mostrando {len(free_images_to_show)} imágenes de 'Free'...")
    for img_name in free_images_to_show:
        img_path = os.path.join(free_dir, img_name)
        xml_path = os.path.join(xml_dir_free, img_name.replace('.jpg',
'.xml'))

        # Verificar si el XML existe
        if os.path.exists(xml_path):
            visualize_image_with_bboxes(img_path, xml_path)

    # Mostrar imágenes de "Occupied"
    print(f"Mostrando {len(occupied_images_to_show)} imágenes de
'Occupied'...")
    for img_name in occupied_images_to_show:
        img_path = os.path.join(occupied_dir, img_name)
        xml_path = os.path.join(xml_dir_occupied, img_name.replace('.jpg',
'.xml'))

        # Verificar si el XML existe
        if os.path.exists(xml_path):
            visualize_image_with_bboxes(img_path, xml_path)

# Llamar la función para visualizar imágenes aleatorias de ambas categorías
visualize_random_images(free_dir, occupied_dir, xml_dir_free,
xml_dir_occupied, num_images=10)
```

Esta validación se aplicó de forma sistemática a todos los conjuntos de datos explorados. Gracias a este proceso de verificación y depuración, se descartaron datasets completos que presentaban inconsistencias graves o etiquetas de baja calidad, manteniendo únicamente aquellos que cumplían con los estándares requeridos por YOLOv8.

#### 4.2.1.3 Script de Verificación de Correspondencia

Para garantizar la integridad y correcta correspondencia entre las imágenes y sus etiquetas respectivas en todos los conjuntos de datos utilizados en esta investigación, se desarrolló un script de verificación automatizada. Este script permite visualizar las imágenes junto con sus labels correspondientes, facilitando la identificación de posibles inconsistencias o errores en el etiquetado. La herramienta se implementó como una medida de control de calidad y fue aplicada sistemáticamente a todos los datasets empleados en el estudio, asegurando así la confiabilidad de los datos de entrada para los procesos de entrenamiento y evaluación de los modelos. A continuación, se presenta el código desarrollado para esta verificación:

```
import os
from PIL import Image, ImageDraw, ImageFont
import matplotlib.pyplot as plt

train_img_dir = 'dataset_test_3/train/images'
train_lbl_dir = 'dataset_test_3/train/labels'

class_names = {
    0: 'empty',
    1: 'occupied'
}

class_colors = {
    0: (0, 255, 255),
    1: (0, 0, 128)
}

image_files = [f for f in os.listdir(train_img_dir) if
f.lower().endswith(('.png', '.jpg', '.jpeg'))]
image_files = sorted(image_files)[:2]

plt.figure(figsize=(12, 6))

for i, img_name in enumerate(image_files):
    img_path = os.path.join(train_img_dir, img_name)
    label_path = os.path.join(train_lbl_dir, os.path.splitext(img_name)[0] +
'.txt')
    print(img_path)
    print(label_path)
```

```
    img = Image.open(img_path).convert("RGBA")
    draw = ImageDraw.Draw(img, "RGBA")

    try:
        font = ImageFont.truetype("arial.ttf", 18)
    except IOError:
        font = ImageFont.load_default()

    if os.path.exists(label_path):
        print(f"Processing labels for {img_name}...")
        with open(label_path, 'r') as f:
            for line in f:
                parts = line.strip().split()
                cls_id, x_center, y_center, w, h = parts
                cls_id = int(cls_id)
                x_center, y_center, w, h = map(float, (x_center, y_center,
w, h))

                img_w, img_h = img.size
                x_min = (x_center - w/2) * img_w
                y_min = (y_center - h/2) * img_h
                x_max = (x_center + w/2) * img_w
                y_max = (y_center + h/2) * img_h

                draw.rectangle([x_min, y_min, x_max, y_max],
outline=class_colors[cls_id], width=2)

                label_text = class_names.get(cls_id, str(cls_id))

                bbox = draw.textbbox((0, 0), label_text, font=font)
                text_width = bbox[2] - bbox[0]
                text_height = bbox[3] - bbox[1]

                draw.rectangle(
                    [x_min, y_min, x_min + text_width + 6, y_min +
text_height + 4],
                    fill=(255, 255, 255, 180)
                )

                draw.text((x_min + 3, y_min + 2), label_text,
fill=class_colors[cls_id], font=font)

    img = img.convert("RGB")
```

```
    plt.subplot(1, 2, i + 1)
    plt.imshow(img)
    plt.title(img_name)
    plt.axis('off')

plt.tight_layout()
plt.show()
```


**Figura 21 Ejemplos de datasets con anotaciones de baja calidad que fueron descartados durante el proceso de selección.**

### 4.2.2 Formato y Estructura de Etiquetas

Para la detección de objetos, cada imagen fue acompañada por una etiqueta en el formato requerido por YOLOv8. Cada etiqueta consiste en una línea con 5 valores organizados de la siguiente manera:

1. **Clase**: 0 para espacio vacío y 1 para espacio ocupado
2. **Coordenadas del cuadro delimitador**: Las siguientes cuatro cifras representan:
    - **x, y**: Coordenadas normalizadas del centro del cuadro delimitador
    - **w, h**: Ancho y altura normalizados del cuadro delimitador

Todas las coordenadas se normalizan con respecto al tamaño total de la imagen, manteniéndose en el rango [0, 1].

**Ejemplo de formato de etiqueta:**
*1 0.23515625 0.28828125 0.0359375 0.0625*

*0 0.259375 0.290625 0.03515625 0.06640625*

Este formato permite que YOLOv8 identifique y localice con precisión los espacios ocupados y vacíos dentro de las imágenes.

Para garantizar que todas las anotaciones cumplieran con los requisitos del formato YOLOv8, se desarrolló un script en Python que inspecciona cada archivo de etiquetas (.txt) dentro de la carpeta correspondiente. Este script valida que cada línea contenga exactamente cinco valores: la clase y las coordenadas normalizadas del cuadro delimitador (centro x, centro y, ancho y alto). Si se detecta alguna línea con un número de elementos distinto, se notifica para descartar registros incorrectos, manteniendo así la calidad del conjunto de datos.

```
import os

labels_dir = 'labels_yolo'

for label_file in os.listdir(labels_dir):
    if not label_file.endswith('.txt'):
        continue

    label_path = os.path.join(labels_dir, label_file)

    with open(label_path, 'r') as f:
        lines = f.readlines()

    for i, line in enumerate(lines):
        parts = line.strip().split()
        if len(parts) != 5:
            print(f"Formato inválido en {label_file} línea {i + 1}:
{line.strip()}")
```

#### 4.2.2.1 Datasets Seleccionados

Después del proceso de evaluación y validación, el modelo fue entrenado utilizando la combinación de tres datasets principales que cumplieron con todos los criterios de calidad y coherencia:

**Dataset PKLot**

- Imágenes de entrenamiento: 8,691 [22]
- Imágenes de validación: 2,483
- Características: Imágenes panorámicas de estacionamientos con etiquetado preciso de espacios ocupados y disponibles


**Figura 22 Ejemplo del dataset PKLot mostrando espacios de estacionamiento con etiquetas de ocupación superpuestas.**

## Dataset CNRPark-EXT v2 (CNRPark.v2i)

- Imágenes de entrenamiento: 2,577 [38]
- Imágenes de validación: 235
- Características: Capturas desde diferentes ángulos de cámara con variaciones en condiciones ambientales

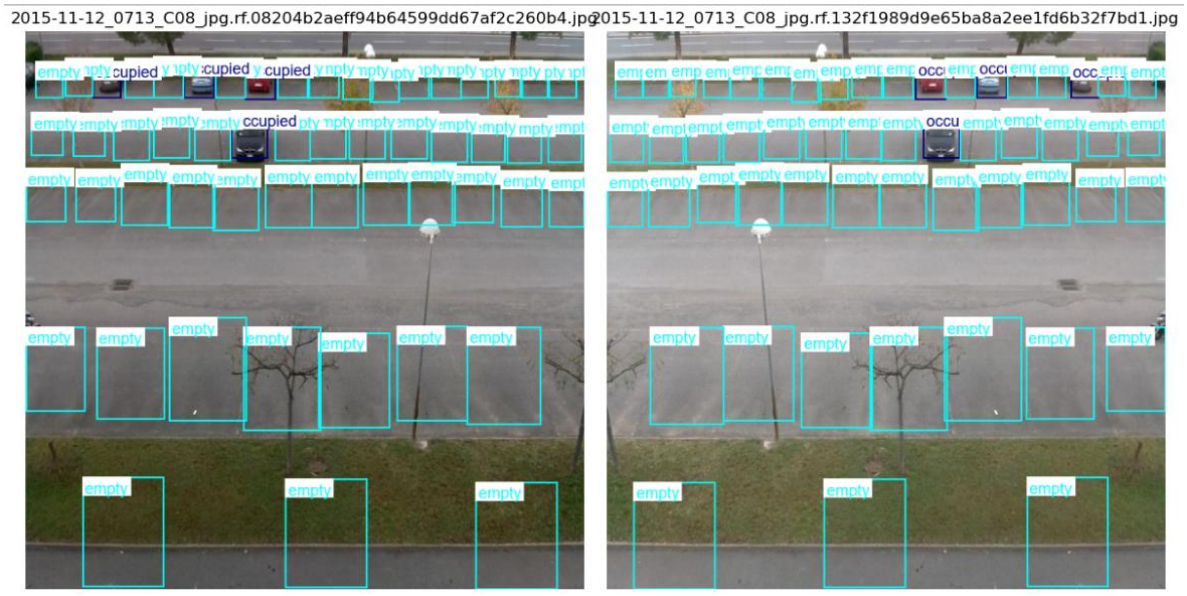



**Figura 23 Imagen representativa del dataset CNRPark.v2i capturada desde perspectiva alternativa de cámara.**

## Dataset Parking.v3i

- Imágenes de entrenamiento: 3,330 [39]
- Imágenes de validación: 317

- Características: Diversidad en perspectivas y condiciones climáticas (soleado, nublado, lluvioso)

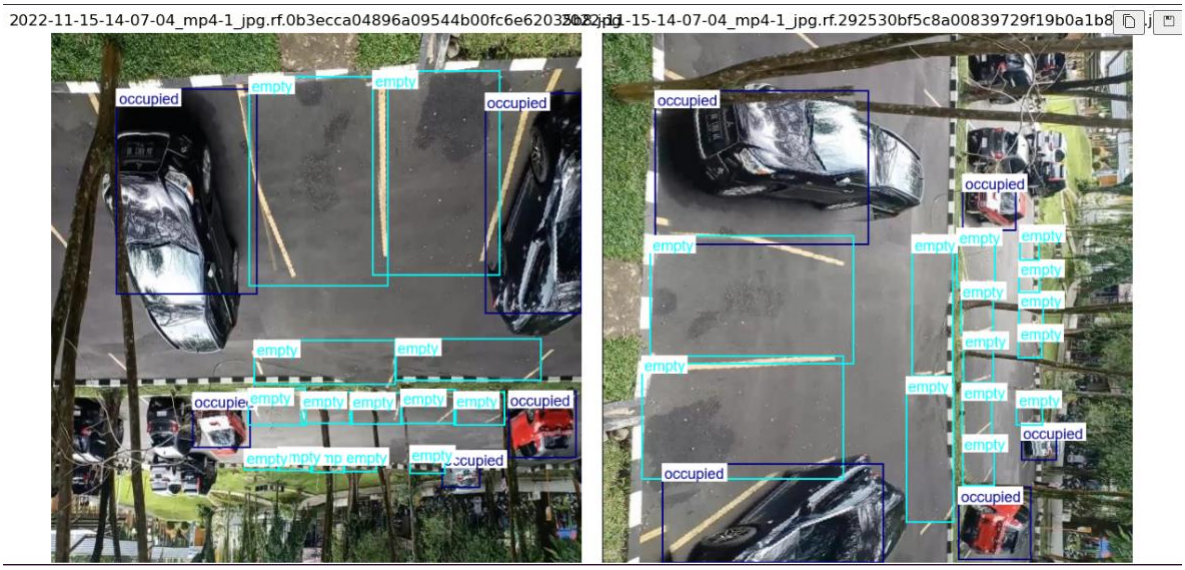



**Figura 24 Muestra del dataset Parking.v3i con una perspectiva alternativa y condiciones climáticas distintas.**

#### 4.2.2.1.1 Estadísticas del Dataset Final

La combinación de los tres datasets resultó en:

- **Total de imágenes de entrenamiento:** 14,598
- **Total de imágenes de validación:** 3,308
- **Relación entrenamiento/validación:** 81.5% / 18.5%

Esta distribución garantiza un volumen adecuado para minimizar el riesgo de sobreajuste y robustece la precisión de la detección. Los datasets seleccionados incluyen:

- Imágenes panorámicas de estacionamientos reales
- Etiquetado cuidadoso de espacios ocupados y disponibles
- Condiciones climáticas variadas (soleado, nublado, lluvioso)
- Múltiples ángulos y perspectivas de cámara
- Diversidad en condiciones de iluminación

#### 4.2.2.1.2 Verificación de Correspondencia Imagen-Etiqueta

Como parte de los controles de calidad implementados durante la preparación del conjunto de datos, se desarrolló un script específico para verificar la correspondencia uno a uno entre cada imagen y su archivo de etiqueta. Este proceso permitió identificar posibles discrepancias, tales como imágenes sin archivo de anotación asociado o archivos de etiquetas sin su imagen correspondiente, situaciones que pueden afectar negativamente el entrenamiento de modelos de detección de objetos. Este script recorre las carpetas de imágenes y etiquetas, compara sus nombres base (sin extensión) y genera reportes de elementos faltantes o inconsistentes.

```python
import os

# Rutas de tus carpetas
images_dir = 'dataset/images/train'
labels_dir = 'dataset/labels/train'

# Obtén listas de nombres de archivos (sin extensión)
image_files = [os.path.splitext(f)[0] for f in os.listdir(images_dir) if
f.lower().endswith(('.jpg', '.jpeg', '.png'))]
label_files = [os.path.splitext(f)[0] for f in os.listdir(labels_dir) if
f.lower().endswith('.txt')]

# Encuentra imágenes sin etiqueta
images_without_labels = [img for img in image_files if img not in
label_files]

# Encuentra etiquetas sin imagen
labels_without_images = [lbl for lbl in label_files if lbl not in
image_files]

print(f'Imágenes sin etiqueta: {images_without_labels}')
print(f'Etiquetas sin imagen: {labels_without_images}')

print(f'Total imágenes: {len(image_files)}')
print(f'Total labels: {len(label_files)}')
```

```
Imágenes sin etiqueta: []
Etiquetas sin imagen: []
Total imágenes: 14598
Total labels: 14598
```

**Figura 25 Salida del script de verificación mostrando la correspondencia perfecta entre imágenes y etiquetas del dataset que se está utilizando**

#### 4.2.2.2 Consideraciones Técnicas y Estándares

Para garantizar la compatibilidad con YOLOv8 de Ultralytics, se aplicaron las siguientes especificaciones técnicas:

**Resolución de Imágenes**

- **Tamaño estándar:** 640 × 640 píxeles (optimizado para YOLOv8)
- **Justificación:** Este tamaño proporciona un balance óptimo entre precisión de detección y tiempo de procesamiento

- **Proceso:** Redimensionamiento automático manteniendo la relación de aspecto

**Formato de Etiquetas**

- **Estructura:** Cada archivo .txt contiene líneas con 5 valores separados por espacios
- **Contenido:** clase, x_center, y_center, width, height
- **Normalización:** Todas las coordenadas en rango [0, 1]
- **Codificación:** UTF-8 para compatibilidad universal

**Estructura de Directorios**

La organización de carpetas sigue el estándar YOLOv8:

dataset/

├── images/

│ ├── train/

│ └── val/

└── labels/

├── train/

└── val/

**Diversidad de Escenarios**

Se verificó la inclusión de:

- **Condiciones de iluminación:** Natural, artificial, mixta
- **Condiciones climáticas:** Soleado, nublado, lluvioso
- **Perspectivas:** Múltiples ángulos de cámara
- **Contextos:** Diferentes tipos de estacionamientos (comerciales, residenciales, públicos)

#### 4.2.2.3 Preprocesamiento Específico para YOLOv8

**Selección de Datos**

Se empleó una muestra representativa que garantizara diversidad en los datos de entrenamiento, incluyendo:

- Diferentes ángulos de captura
- Variaciones en condiciones climáticas
- Múltiples configuraciones de estacionamiento
- Diversidad en tipos de vehículos

**Redimensionamiento de Imágenes**

- **Tamaño objetivo:** 640 × 640 píxeles
- **Método:** Redimensionamiento proporcional con relleno (padding) cuando sea necesario
- **Justificación:** Tamaño estándar que asegura balance entre precisión y eficiencia computacional

**Anotación de Imágenes**

Se anotaron las imágenes con las coordenadas de los objetos (en este caso, los espacios de estacionamiento) y sus respectivas clases (vacío u ocupado). Para ello, se utilizaron herramientas de etiquetado específicas para YOLO que generan archivos en formato .txt, donde cada línea contiene la información de la clase y las coordenadas del cuadro delimitador (bounding box) de cada objeto detectado.

**Normalización y Preparación**

- **Normalización de píxeles:** Valores ajustados al rango [0, 1]
- **Formato de color:** RGB estándar
- **Calidad:** Verificación de integridad de todas las imágenes

**Augmentación de Datos**

Durante el entrenamiento, YOLOv8 aplica automáticamente las siguientes técnicas de augmentación:

- Rotación aleatoria
- Escalado
- Cambios en brillo y contraste
- Desplazamiento horizontal y vertical
- Recorte aleatorio

Esta preparación exhaustiva de datos garantiza que el modelo YOLOv8 pueda generalizar efectivamente a diferentes escenarios de estacionamiento en condiciones reales de operación.

### 4.2.3 Entrenamiento de YOLOv8

La arquitectura YOLOv8 se entrenó con una colección de 14,598 fotografías etiquetadas. Se mantuvo un balance entre las clases para evitar un sesgo hacia alguna categoría predominante. Con el propósito de medir la capacidad de generalización del modelo, se empleó un conjunto de validación compuesto por 3,308 imágenes, con el objetivo de verificar su precisión al predecir la ocupación de espacios en imágenes nuevas no vistas durante el entrenamiento.

#### 4.1.1.1 Implementación Local y Optimización en la Nube

Durante las etapas iniciales del desarrollo del presente proyecto, se optó por ejecutar los experimentos de entrenamiento del modelo de detección de objetos de forma local, utilizando la computadora personal del autor. Esta estrategia resultó ser viable para las pruebas preliminares; sin embargo, al escalar los datos y aumentar la cantidad de épocas de entrenamiento a 25, el tiempo de procesamiento se volvió un factor crítico. En promedio, completar un entrenamiento de esta magnitud requería aproximadamente 50 horas continuas de ejecución, lo que limitaba la capacidad de iterar y ajustar hiperparámetros de forma ágil.

Frente a esta limitación, se exploró la posibilidad de migrar la carga de cómputo a la nube, aprovechando la infraestructura de hardware acelerado disponible en plataformas especializadas. No obstante, soluciones robustas como Amazon Web Services (AWS) o Microsoft Azure implicaban un costo elevado para un proyecto académico. Por esta razón, se eligió Google Colab, una plataforma que ofrece entornos de computación gratuitos y de pago con acceso a unidades de procesamiento gráfico (GPU).
En este contexto, se adquirió la suscripción a Google Colab Pro, lo cual permitió desbloquear recursos adicionales fundamentales para la ejecución eficiente del modelo. Entre los principales beneficios destacan:

- Acceso prioritario a GPUs de alto rendimiento, como la NVIDIA A100, la cual ofrece un rendimiento superior frente a GPUs más básicas como la T4 o K80, gracias a su mayor capacidad de memoria, arquitectura optimizada para operaciones de entrenamiento intensivo y soporte para cargas de trabajo de IA de última generación.
- Sesiones de ejecución de mayor duración, reduciendo la interrupción de procesos extensos.
- Mayores límites de uso de recursos, permitiendo entrenamientos más largos y con mayores volúmenes de datos.

Si bien el entorno de Google Colab resuelve la disponibilidad de hardware, en las primeras iteraciones se identificaron cuellos de botella relacionados con la transferencia y gestión de datos. Inicialmente, se probó subir el conjunto de datos al servicio OneDrive, para luego acceder a él mediante la API correspondiente durante la ejecución de los notebooks en Colab. Sin embargo, esta estrategia resultó poco práctica, ya que la transferencia constante de archivos de gran tamaño generaba retrasos significativos.
La solución más eficiente consistió en subir el dataset comprimido (.zip) directamente a Google Drive y, posteriormente, emplear comandos de Python dentro de Colab para descomprimir el archivo en el entorno

de ejecución (/content). Esta práctica resultó ser notablemente más rápida, ya que evita llamadas de red continuas y reduce la latencia de lectura de archivos grandes al trabajar directamente con el almacenamiento vinculado al entorno de Colab. Gracias a esta optimización, fue posible reducir drásticamente los tiempos de entrenamiento de 50 horas a aproximadamente una hora, facilitando iteraciones constantes y mejoras sucesivas en la arquitectura del modelo.

En conjunto, la combinación de recursos locales, el aprovechamiento de infraestructura en la nube mediante Google Colab Pro, y la optimización de la gestión de datos permitieron que la fase experimental se desarrollara de forma mucho más ágil, eficiente y con un uso racional de recursos económicos.

### 4.1.2 Código

Para verificar la correcta carga de los datos y realizar una inspección visual del conjunto de entrenamiento, se empleó un fragmento de código en Python con las bibliotecas PIL y matplotlib. Se definió la ruta de las imágenes, se filtraron los archivos de formato común y se seleccionaron dos ejemplos para abrirlos y mostrarlos junto con su nombre. Esta visualización permitió constatar la precisión y organización de las imágenes empleadas en el entrenamiento del modelo.

```
from PIL import Image
import os
%matplotlib inline
val_dir = 'dataset/images/train'

image_files = [f for f in os.listdir(val_dir) if f.lower().endswith(('.png',
'.jpg', '.jpeg'))]
image_files = sorted(image_files)[:2]

plt.figure(figsize=(10, 5))

for i, img_name in enumerate(image_files):
    img_path = os.path.join(val_dir, img_name)
    img = Image.open(img_path)
    plt.subplot(1, 2, i + 1)
    plt.imshow(img)
    plt.title(img_name)
    plt.axis('off')

plt.tight_layout()
plt.show()
```

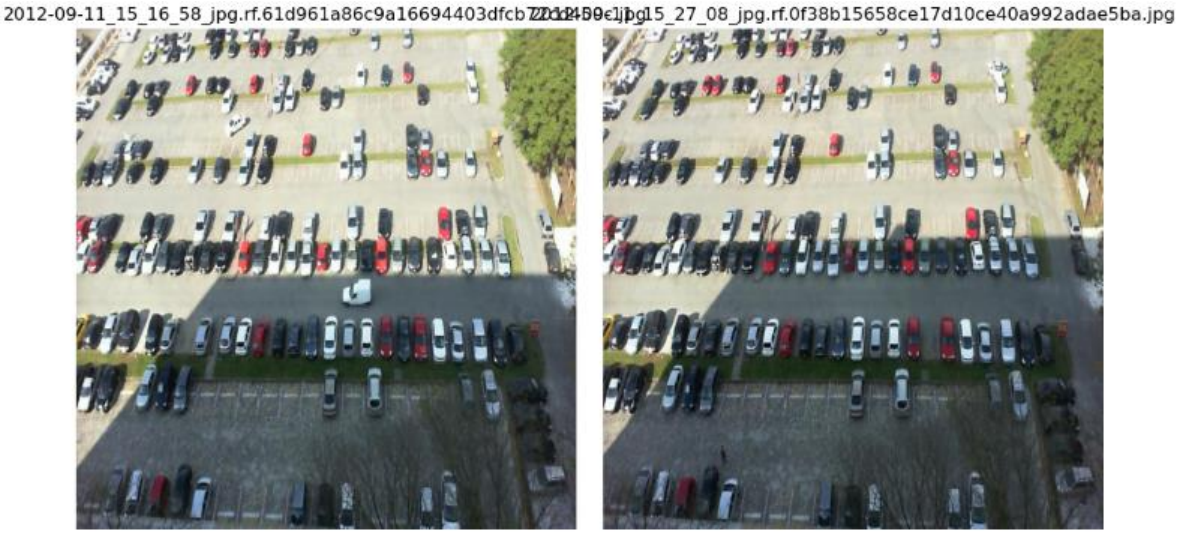



**Figura 26 Visualización de ejemplos representativos del conjunto de entrenamiento: imágenes seleccionadas para inspección y verificación previa al entrenamiento del modelo.**

Para visualizar y verificar las anotaciones de los cuadros delimitadores en el conjunto de entrenamiento, se implementó un código en Python que utiliza las bibliotecas PIL y matplotlib. El código carga imágenes del directorio correspondiente y sus archivos de etiquetas en formato YOLO, que contienen la clase y las coordenadas normalizadas de los cuadros delimitadores. Cada archivo de etiquetas se procesa para extraer las coordenadas del centro, ancho y alto de cada caja, las cuales se convierten a coordenadas en píxeles para superponerlas sobre la imagen original. Se dibujan los cuadros delimitadores con colores específicos según la clase (vacío u ocupado), junto con etiquetas de texto que indican la categoría correspondiente. El resultado es una visualización clara de las imágenes utilizadas en la fase de entrenamiento y sus anotaciones asociadas, facilitando la inspección visual de la calidad y precisión de las etiquetas antes del entrenamiento del modelo.

```
import os
from PIL import Image, ImageDraw, ImageFont
import matplotlib.pyplot as plt

train_img_dir = 'dataset/images/train'
train_lbl_dir = 'dataset/labels/train'

class_names = {
    0: 'empty',
    1: 'occupied'
}

class_colors = {
    0: (0, 255, 255),
    1: (0, 0, 128)
}
```

```
image_files = [f for f in os.listdir(train_img_dir) if
f.lower().endswith(('.png', '.jpg', '.jpeg'))]
image_files = sorted(image_files)[:2]

plt.figure(figsize=(12, 6))

for i, img_name in enumerate(image_files):
    img_path = os.path.join(train_img_dir, img_name)
    label_path = os.path.join(train_lbl_dir, os.path.splitext(img_name)[0] +
'.txt')

    img = Image.open(img_path).convert("RGBA")
    draw = ImageDraw.Draw(img, "RGBA")

    try:
        font = ImageFont.truetype("arial.ttf", 18)
    except IOError:
        font = ImageFont.load_default()

    if os.path.exists(label_path):
        with open(label_path, 'r') as f:
            for line in f:
                parts = line.strip().split()
                cls_id, x_center, y_center, w, h = parts
                cls_id = int(cls_id)
                x_center, y_center, w, h = map(float, (x_center, y_center,
w, h))

                img_w, img_h = img.size
                x_min = (x_center - w/2) * img_w
                y_min = (y_center - h/2) * img_h
                x_max = (x_center + w/2) * img_w
                y_max = (y_center + h/2) * img_h

                draw.rectangle([x_min, y_min, x_max, y_max],
outline=class_colors[cls_id], width=2)

                label_text = class_names.get(cls_id, str(cls_id))

                bbox = draw.textbbox((0, 0), label_text, font=font)
                text_width = bbox[2] - bbox[0]
                text_height = bbox[3] - bbox[1]

                draw.rectangle(
```

```
                    [x_min, y_min, x_min + text_width + 6, y_min +
text_height + 4],
                    fill=(255, 255, 255, 180)
                )

                draw.text((x_min + 3, y_min + 2), label_text,
fill=class_colors[cls_id], font=font)

    # Convert back to RGB for matplotlib display
    img = img.convert("RGB")

    # Plot the image with annotations
    plt.subplot(1, 2, i + 1)
    plt.imshow(img)
    plt.title(img_name)
    plt.axis('off')

plt.tight_layout()
plt.show()
```

**Figura 27 Visualización de imágenes de entrenamiento con sus respectivos cuadros delimitadores y etiquetas de clase para la detección de espacios vacíos y ocupados.**

Se utilizó el framework Ultralytics YOLOv8 a fin de desarrollar un modelo para la detección de elementos aplicado al conjunto de datos PKLot, orientado a la identificación de espacios de estacionamiento vacíos y ocupados. El modelo preentrenado yolov8n.pt fue cargado y configurado para realizar un entrenamiento supervisado durante 25 épocas, utilizando imágenes de tamaño 640×640 píxeles, además de un batch size de 16 instancias. Para seguimiento y registro del experimento, se empleó MLflow, configurando la URI del servidor local y estableciendo el experimento con el nombre PKLot-ParkingDetection.

Para este proyecto se realizaron numerosos experimentos tanto de forma local como en la nube, evaluando distintas configuraciones para optimizar el desempeño del modelo. Se entrenaron redes con diferentes cantidades de épocas, aplicando data augmentation en algunas pruebas y omitiéndolo en otras, además de variar los parámetros de aumento de datos para analizar su impacto en la capacidad de generalización. También se probaron distintos conjuntos de datos, se aplicaron técnicas de TL y se ajustaron los tamaños de batch para lograr una ejecución más estable y eficiente. Como parte de esta etapa experimental, se

entrenaron modelos de distintas escalas, como YOLO Nano y YOLO Small, para comparar su rendimiento y balance entre velocidad y precisión. Tras este proceso de pruebas, se seleccionó la configuración que ofreció los mejores resultados en precisión y tiempo de entrenamiento. A continuación, se describe el procedimiento y se presentan los resultados obtenidos del experimento final optimizado.

#### 4.1.2.1 Entrenamiento del Modelo YOLOv8

Para llevar a cabo el entrenamiento del modelo de detección de espacios de estacionamiento, se utilizó YOLOv8 de Ultralytics, aprovechando su arquitectura optimizada y la disponibilidad de modelos preentrenados que permiten realizar transfer learning de manera eficiente.

El siguiente fragmento muestra el script utilizado para montar Google Drive —asegurando el almacenamiento persistente de resultados—, cargar el modelo YOLOv8 small, definir la configuración de entrenamiento y ejecutar el proceso por 30 épocas consecutivas.

Además, se incluyeron técnicas de aumento de datos (data augmentation) avanzadas (rotaciones, traslaciones, escalado, espejado horizontal y vertical, ajustes de color, oclusiones simuladas, mosaico y mixup) con el objetivo de incrementar la robustez del modelo frente a variaciones de ángulos, condiciones climáticas y posibles oclusiones, características comunes en cámaras de seguridad de estacionamientos.

```
from ultralytics import YOLO
from google.colab import drive
import shutil
import os

# Monta Google Drive
drive.mount('/content/drive')

# Carga el modelo YOLOv8 small preentrenado
model = YOLO('yolov8s.pt')

# Entrena las 30 épocas de una sola vez
results = model.train(
    data='/content/parking_model.yaml',  # Ruta correcta
    epochs=30,
    imgsz=640,
    batch=16,
    project='/content/drive/MyDrive/Colab Notebooks/YOLO_runs_final',
    name='train',
    verbose=True,
    # --- ADDED: Sophisticated Built-in Augmentations ---
    degrees=180.0,      # Rotación total
    translate=0.1,      # Traslación de imagen
    scale=0.5,          # Escalado de imagen
    fliplr=0.5,         # Espejado horizontal
    flipud=0.5,         # Espejado vertical
```

```
    hsv_h=0.015,         # Variación de tono
    hsv_s=0.7,           # Saturación
    hsv_v=0.4,           # Brillo/valor
    erasing=0.4,         # Simulación de oclusión
    mosaic=1.0,          # Aumento tipo mosaico
    mixup=0.1            # Aumento mixup
)
```

Este esquema permitió aprovechar la potencia de cómputo en Google Colab, almacenar resultados de forma segura en Google Drive y realizar un entrenamiento supervisado bajo un esquema reproducible y escalable.

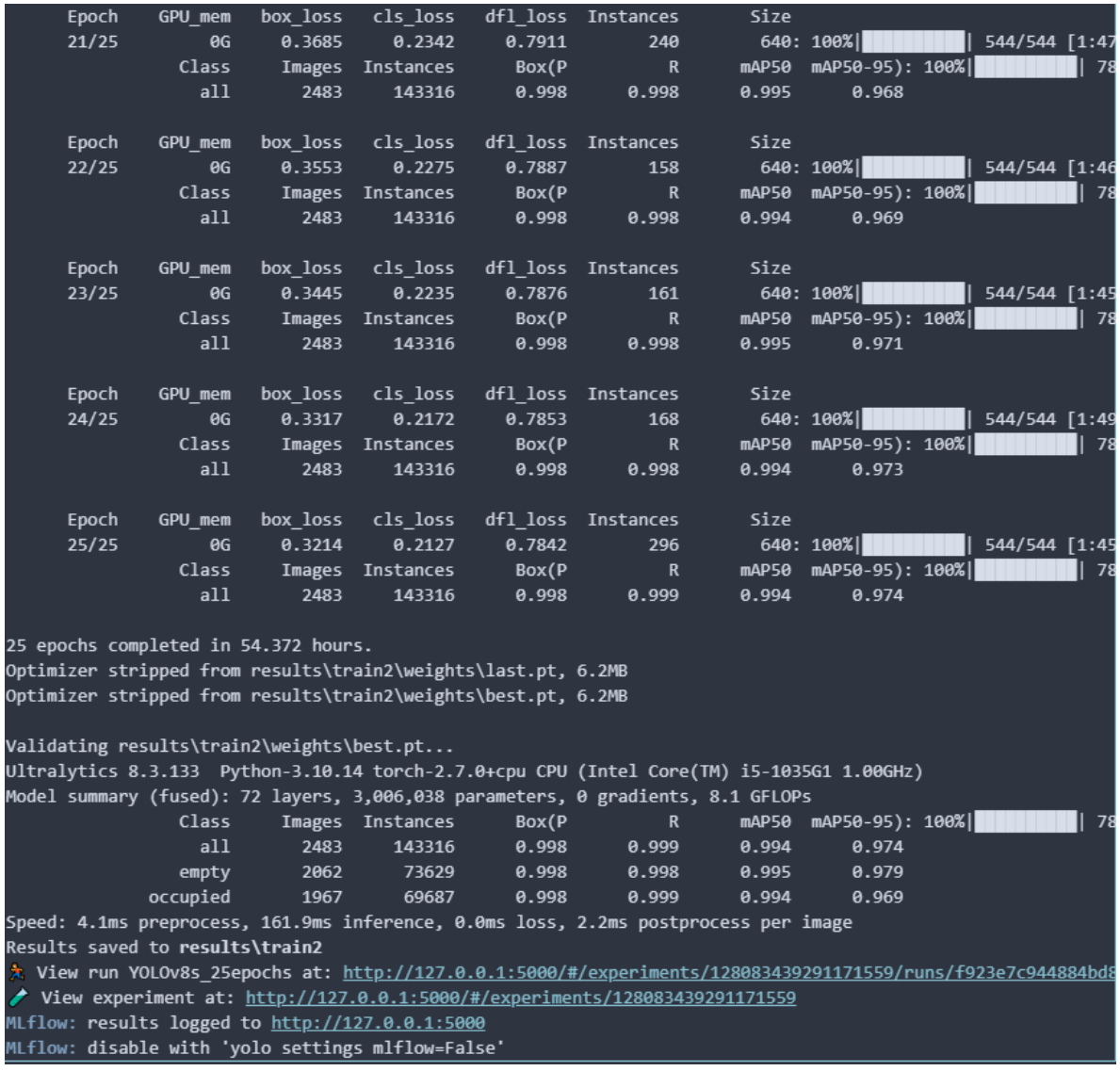



**Figura 28 Resultados del entrenamiento del modelo local de YOLOv8n en el dataset PKLot: desarrollo de indicadores clave durante 25 épocas.**

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
21/25      8.51G     0.5762     0.3349     0.8372         45        640: 100%|██████████| 495/495 [00:42<00:00, 11.51it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 34/34 [00:04<00:00,  6.84it/s]
             all       1075      40748      0.987      0.982      0.994      0.906

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
22/25      8.51G      0.563     0.3285     0.8345         68        640: 100%|██████████| 495/495 [00:43<00:00, 11.43it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 34/34 [00:04<00:00,  6.90it/s]
             all       1075      40748      0.987      0.982      0.994      0.906

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
23/25      8.51G     0.5525     0.3226     0.8322         17        640: 100%|██████████| 495/495 [00:43<00:00, 11.28it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 34/34 [00:04<00:00,  6.85it/s]
             all       1075      40748      0.987      0.982      0.994      0.906

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
24/25      8.51G     0.5312     0.3138     0.8278         16        640: 100%|██████████| 495/495 [00:44<00:00, 11.22it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 34/34 [00:04<00:00,  6.82it/s]
             all       1075      40748      0.987      0.982      0.994      0.906

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
25/25      8.51G     0.5213     0.3103     0.8259        206        640: 100%|██████████| 495/495 [00:44<00:00, 11.22it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 34/34 [00:04<00:00,  6.86it/s]
             all       1075      40748      0.987      0.982      0.994      0.906

25 epochs completed in 0.354 hours.
Optimizer stripped from results/train2/weights/last.pt, 6.2MB
Optimizer stripped from results/train2/weights/best.pt, 6.2MB

Validating results/train2/weights/best.pt...
Ultralytics 8.3.166 🚀 Python-3.11.13 torch-2.6.0+cu124 CUDA:0 (NVIDIA A100-SXM4-40GB, 40507MiB)
Model summary (fused): 72 layers, 3,006,038 parameters, 0 gradients, 8.1 GFLOPs
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 34/34 [00:12<00:00,  2.62it/s]
                 all       1075      40748      0.987      0.982      0.994      0.906
               empty        985      19883      0.982      0.976      0.993      0.912
            occupied       1026      20865      0.992      0.988      0.995      0.901
Speed: 0.2ms preprocess, 3.0ms inference, 0.1ms loss, 4.3ms postprocess per image
Results saved to results/train2
MLflow: results logged to runs/mlflow
```

**Figura 29 Resultados del entrenamiento en Google colab del modelo local de YOLOv8n en el dataset PKLot: desarrollo de indicadores clave durante 25 épocas.**

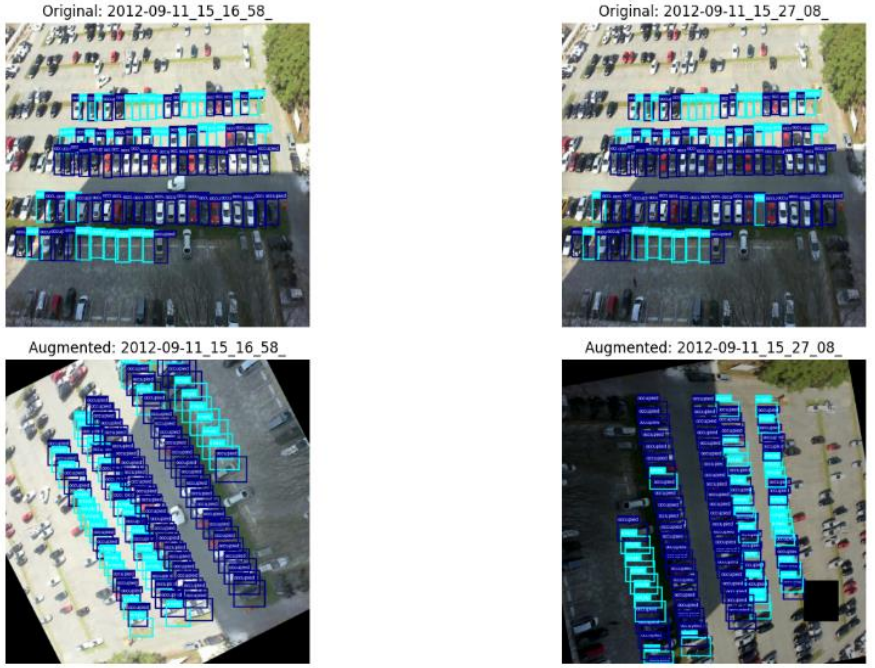



**Figura 30 Ejemplo de imágenes originales vs. aumentadas con anotaciones de detección de objetos.**

Para analizar la progresión de la función de pérdida a lo largo del proceso de entrenamiento del modelo, se utilizó un script en Python que emplea pandas, matplotlib y PIL. A partir del archivo results.csv, generado automáticamente por YOLOv8 tras finalizar el entrenamiento, se cargaron los datos de cada época para graficar la función de error de las cajas delimitadoras (box loss) y el costo relacionado con la clasificación (classification loss). La visualización de estas curvas hace posible analizar el comportamiento del modelo respecto a lo largo de las épocas, identificando la tendencia de ajuste y potenciales dificultades relacionadas con el sobreajuste o subajuste durante el proceso de entrenamiento.

```
# Training Curves
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
%matplotlib inline

results_path = 'results/train2/results.csv'
df = pd.read_csv(results_path)

plt.figure(figsize=(10, 6))
plt.plot(df['epoch'], df['train/box_loss'], label='Box Loss')
plt.plot(df['epoch'], df['train/cls_loss'], label='Classification Loss')
plt.title('Training Losses')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```

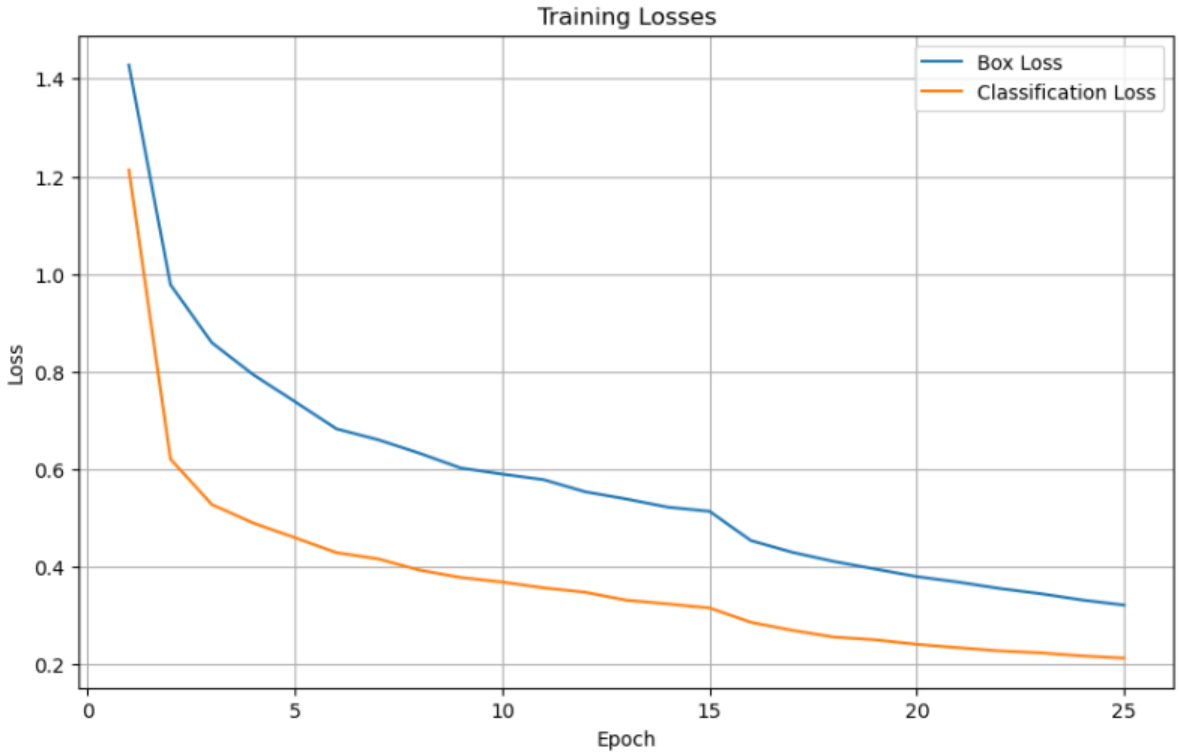



**Figura 31 Curvas de pérdida de cajas delimitadoras y clasificación durante el entrenamiento del modelo YOLOv8.**

Para complementar el análisis del entrenamiento, se elaboró un script en Python que permite visualizar la evolución de la función de error en la fase de evaluación. Con este propósito, se cargaron los datos del archivo results.csv generado por YOLOv8, utilizando la biblioteca panda. Posteriormente, se graficaron

las curvas correspondientes a la función de error correspondiente a las cajas delimitadoras (Validation Box Loss) y la función de error asociada a la clasificación (Validation Classification Loss) en función del número de épocas. Esta visualización facilita la comparación de la pérdida cuando se está entrenando y validando el módelo, permitiendo detectar posibles problemas de sobreajuste o divergencia en el rendimiento del modelo.

```
results_path = 'results/train2/results.csv'
df = pd.read_csv(results_path)

plt.figure(figsize=(10, 6))

plt.plot(df["epoch"], df["val/box_loss"], label="Validation Box Loss")
plt.plot(df["epoch"], df["val/cls_loss"], label="Validation Classification
Loss")

plt.title("Validation Losses")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

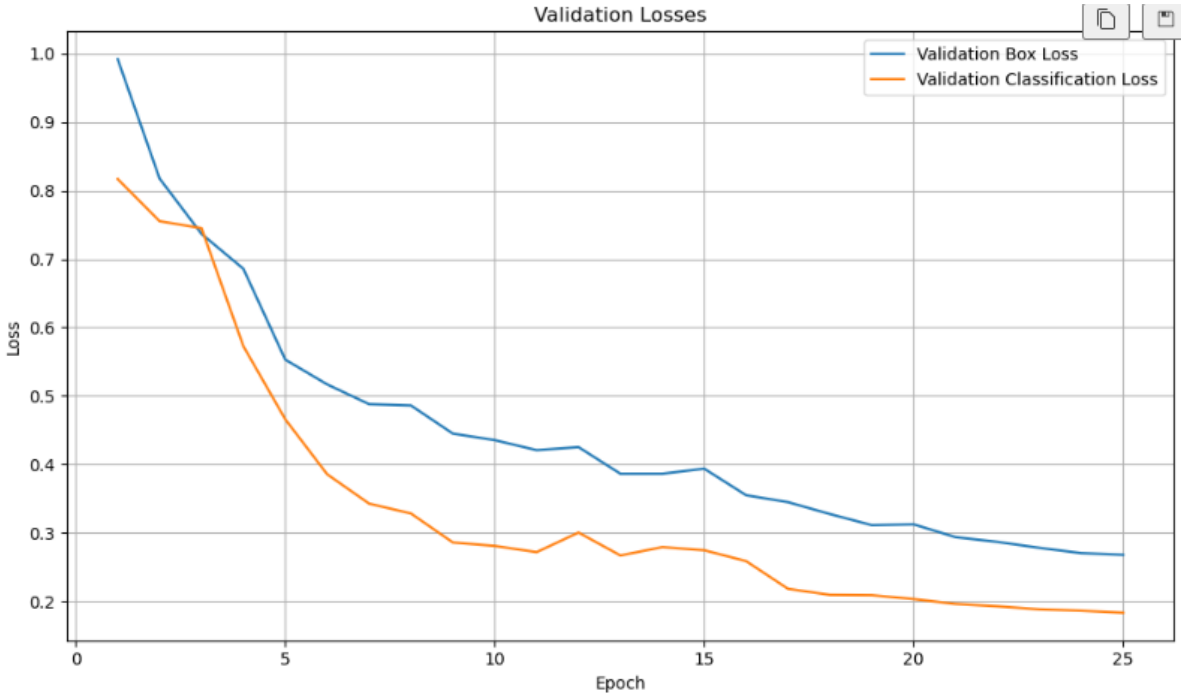



**Figura 32 Curvas de pérdida de validación para cajas delimitadoras y clasificación mientras se entrena el modelo YOLOv8s.**

Con el propósito de analizar el desempeño de YOLOv8 entrenado, se cargaron los pesos finales (best.pt) y se ejecutó el proceso de validación utilizando el archivo de configuración parking_model.yaml. Mediante la función de validación de Ultralytics, se calcularon los indicadores de precisión y detección sobre los datos de validación, guardando los resultados y generando archivos en formato JSON para su posterior análisis y comparación.

| Class | Images | Instances | Box(P | R | mAP50 | mAP50-95): |
|---|---|---|---|---|---|---|
| all | 2483 | 143316 | 0.998 | 0.999 | 0.994 | 0.974 |
| empty | 2062 | 73629 | 0.998 | 0.998 | 0.995 | 0.979 |
| occupied | 1967 | 69687 | 0.998 | 0.999 | 0.994 | 0.969 |

**Figura 33 Métricas de desempeño obtenidas en la validación del modelo YOLOv8 entrenado para detección de espacios de estacionamiento.**

Para verificar visualmente los resultados de la fase de validación, se cargaron imágenes de salida generadas por el modelo YOLOv8, almacenadas en el directorio correspondiente a las predicciones (val_preds). Mediante el uso de las bibliotecas PIL y matplotlib, se seleccionaron dos imágenes de ejemplo para ser mostradas junto con sus nombres de archivo, permitiendo observar cómo se realizaron las detecciones y cuadros delimitadores sobre el conjunto de validación.

```
from PIL import Image
import os

val_dir = 'runs/detect/val_preds'

#  Show two sample images
image_files = [f for f in os.listdir(val_dir) if f.lower().endswith(('.png',
'.jpg', '.jpeg'))]
image_files = sorted(image_files)[:2]

plt.figure(figsize=(10, 5))

for i, img_name in enumerate(image_files):
    img_path = os.path.join(val_dir, img_name)
    img = Image.open(img_path)
    plt.subplot(1, 2, i + 1)
    plt.imshow(img)
    plt.title(img_name)
    plt.axis('off')

plt.tight_layout()
plt.show()
```

**Figura 34 Ejemplos de imágenes del conjunto de validación con predicciones generadas por YOLOv8.**

### 4.1.3 Curvas de Entrenamiento y Validación

Se elaboraron curvas de precisión y recall durante la fase de entrenamiento y validación en el transcurso de las épocas. Estas curvas evidenciaron rápida convergencia durante las fases iniciales del entrenamiento, seguida de un desempeño estable durante el resto del proceso. En particular, se observó que el modelo alcanzó niveles elevados de precisión y recall desde las primeras iteraciones, manteniendo un rendimiento consistente hacia las fases finales del entrenamiento.

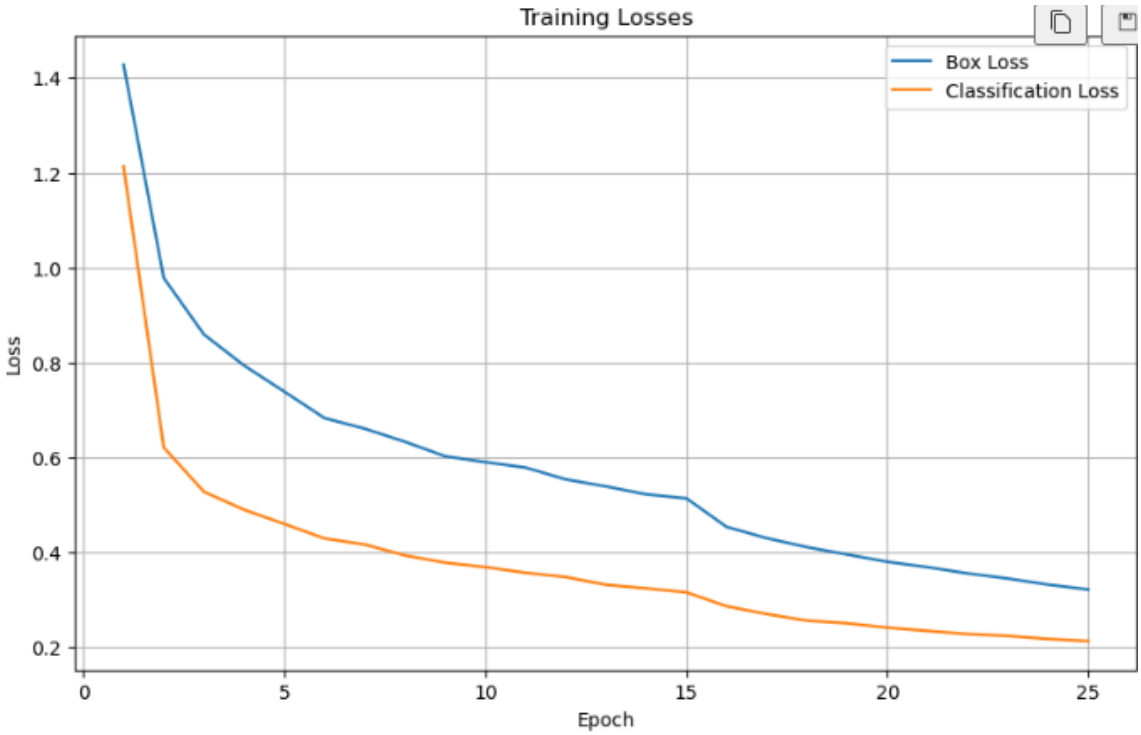



**Figura 35 Curvas de pérdida en el proceso de entrenamiento: Evolución de Box Loss y Classification Loss a lo largo de 25 épocas, mostrando convergencia rápida en las primeras iteraciones y estabilización gradual hacia valores bajos**

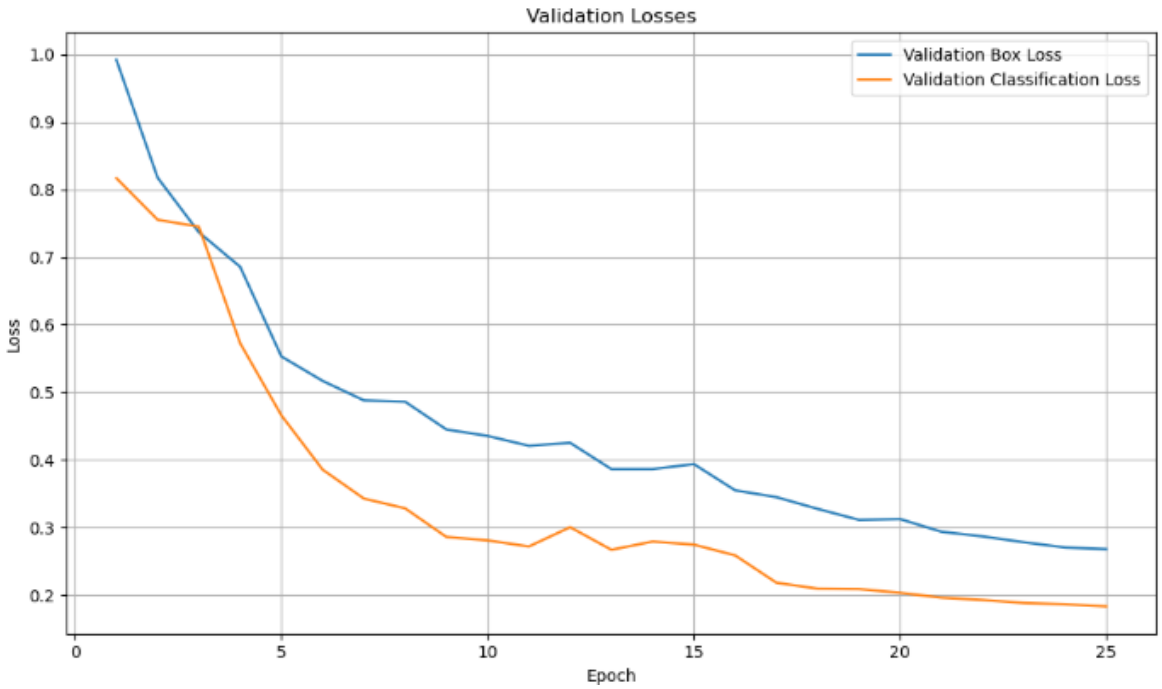



**Figura 36 Desarrollo y valoración de un Modelo de Detección de Objetos Basado en CNN: Análisis de Convergencia y Optimización de Pérdidas de Clasificación y Localización**

### 4.1.4 Métricas de Evaluación

Al evaluar el modelo YOLOv8, se emplearon varias métricas desempeño para cuantificar su efectividad en la tarea de detección de objetos y clasificación de los espacios de estacionamiento como ocupados o vacíos. Las métricas obtenidas para el conjunto de validación fueron las siguientes:

Precisión (P): La precisión medida para el conjunto de validación fue 0.998 para todas las clases. Este valor indica que el modelo fue muy preciso al predecir los espacios ocupados y vacíos.

Recuperación (R): La recuperación fue 0.999, lo que demuestra que el modelo logró identificar casi todos los espacios ocupados y vacíos correctamente.

mAP50 (mean Average Precision a 50% de IoU): El valor de mAP50 fue 0.994, lo que señala que el modelo muestra un rendimiento sobresaliente al estimar de manera precisa la ubicación y clasificación de los elementos en las imágenes con una IoU de 50%.

mAP50-95 (mean Average Precision a IoU entre 50% y 95%): El valor de mAP50-95 fue 0.974, lo que muestra que el modelo también fue muy efectivo al predecir objetos con una mayor precisión, incluso cuando la IoU era más estricta.

**Resultados por Clases**

El rendimiento del modelo también fue evaluado por clases individuales, es decir, espacios vacíos y espacios ocupados:

Espacios vacíos: El modelo obtuvo un nivel de precisión de 0.998 además de un nivel de recuperación de 0.998 para esta clase, con un mAP50 de 0.995 y un mAP50-95 de 0.979.

Espacios ocupados: El modelo logró una precisión equivalente a 0.998 y una recuperación de 0.999 para esta clase, con un mAP50 de 0.994 y un mAP50-95 de 0.969.

Los valores obtenidos reflejan que YOLOv8 logró clasificar y detectar tanto los espacios ocupados como los vacíos con un alto nivel de precisión, superando con creces los modelos tradicionales de Machine.

**Matriz de confusión**

Para una valoración visual de la efectividad, se empleó una matriz de confusión, que mostró cómo el algoritmo categorizó las imágenes de acuerdo con las clases de espacios ocupados y espacios vacíos. La matriz de confusión reveló que el algoritmo presentó un número reducido de errores, manteniendo tasas mínimas de FPR y FN.

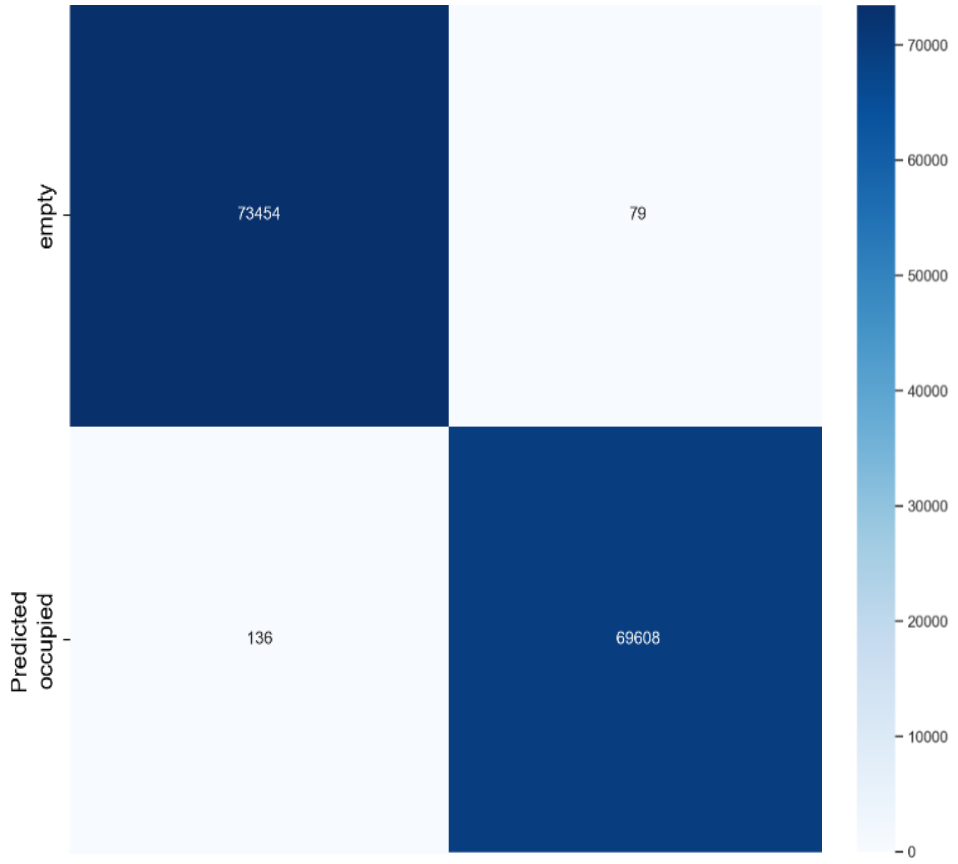



**Figura 37 Matriz de Confusión para la Clasificación de Espacios de Estacionamiento: la matriz muestra la distribución de predicciones correctas e incorrectas del modelo, con 73,454 verdaderos negativos (espacios vacíos), 69,608 verdaderos positivos (espacios ocupados), 79 falsos positivos y 136 FN, alcanzando una exactitud del 99.85%.**

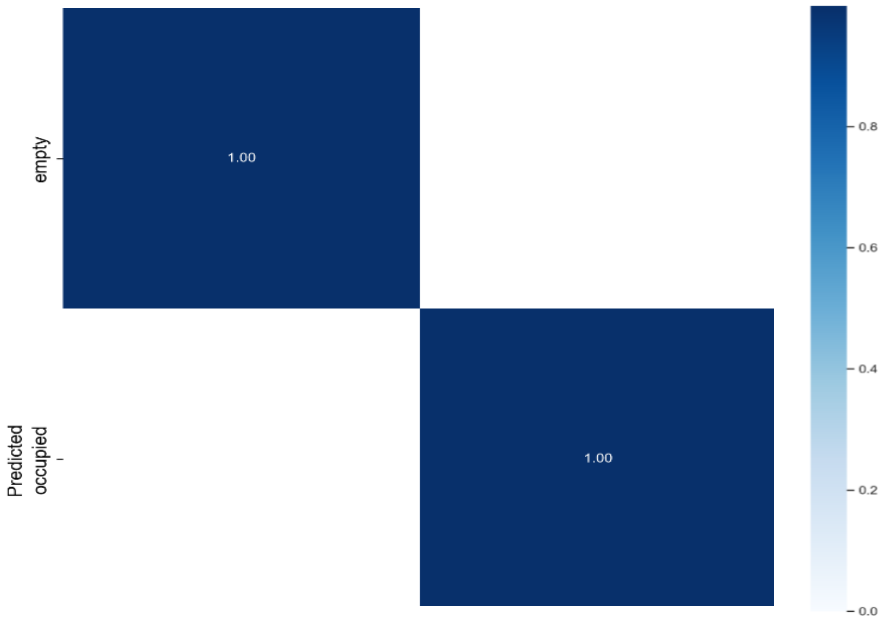



**Figura 38 Matriz de Confusión Normalizada para la Clasificación de Espacios de Estacionamiento: la matriz presenta los valores normalizados de las predicciones del modelo, mostrando una clasificación perfecta (1.00) tanto para espacios vacíos como ocupados, con valores de 0.00 en los falsos positivos y FN**

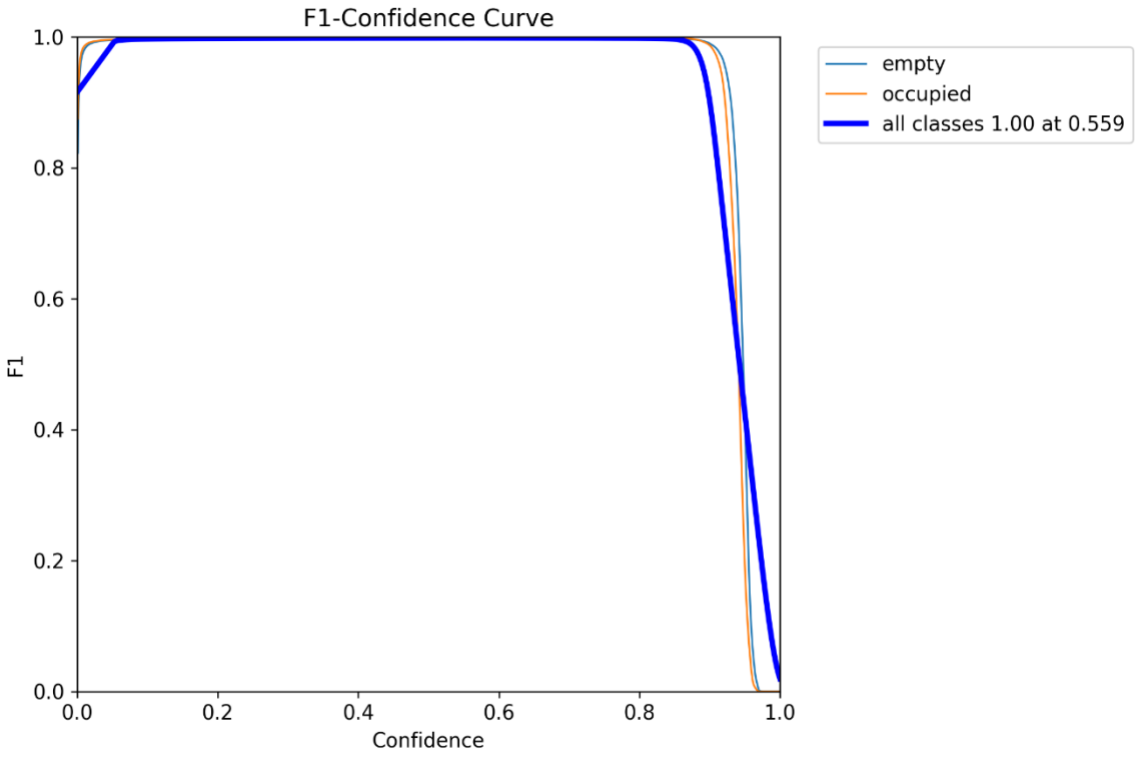



**Figura 39 Evolución de F1-score durante el entrenamiento del modelo YOLOv8.**

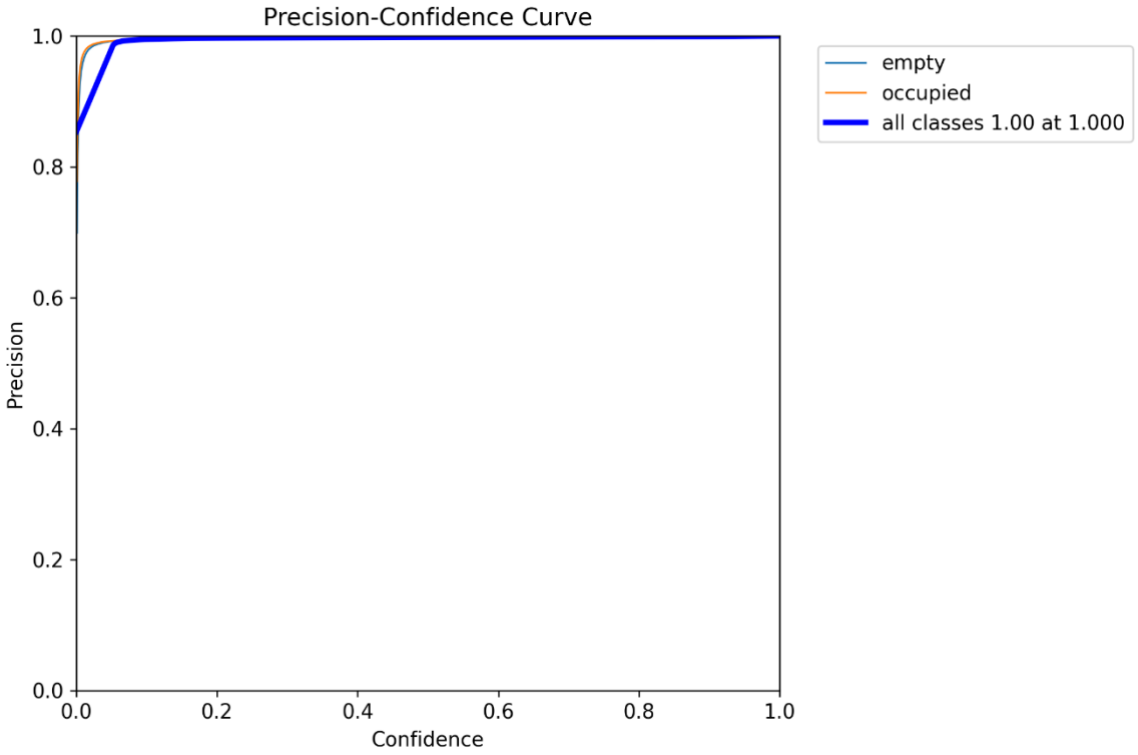



**Figura 40 Curva de precisión obtenida en la fase de entrenamiento del modelo YOLOv8.**

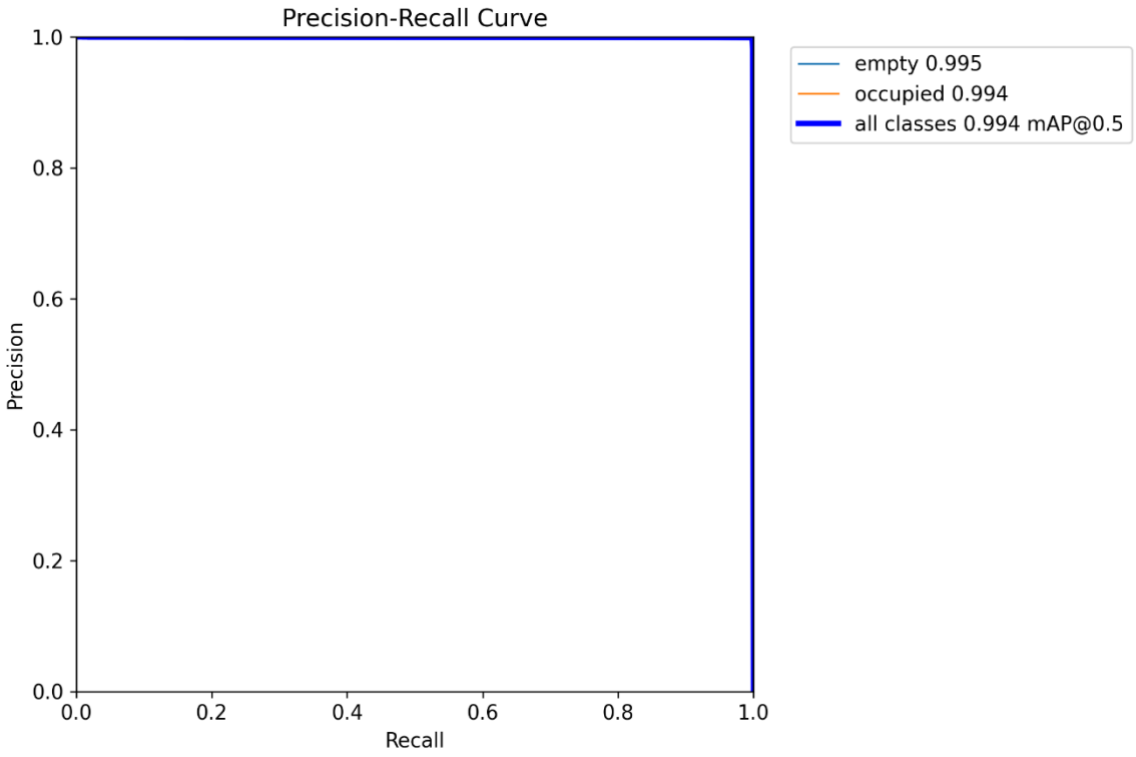



**Figura 41 Curva de Precisión-Recuperación (PR Curve) del modelo YOLOv8 evaluado sobre el conjunto de validación.**

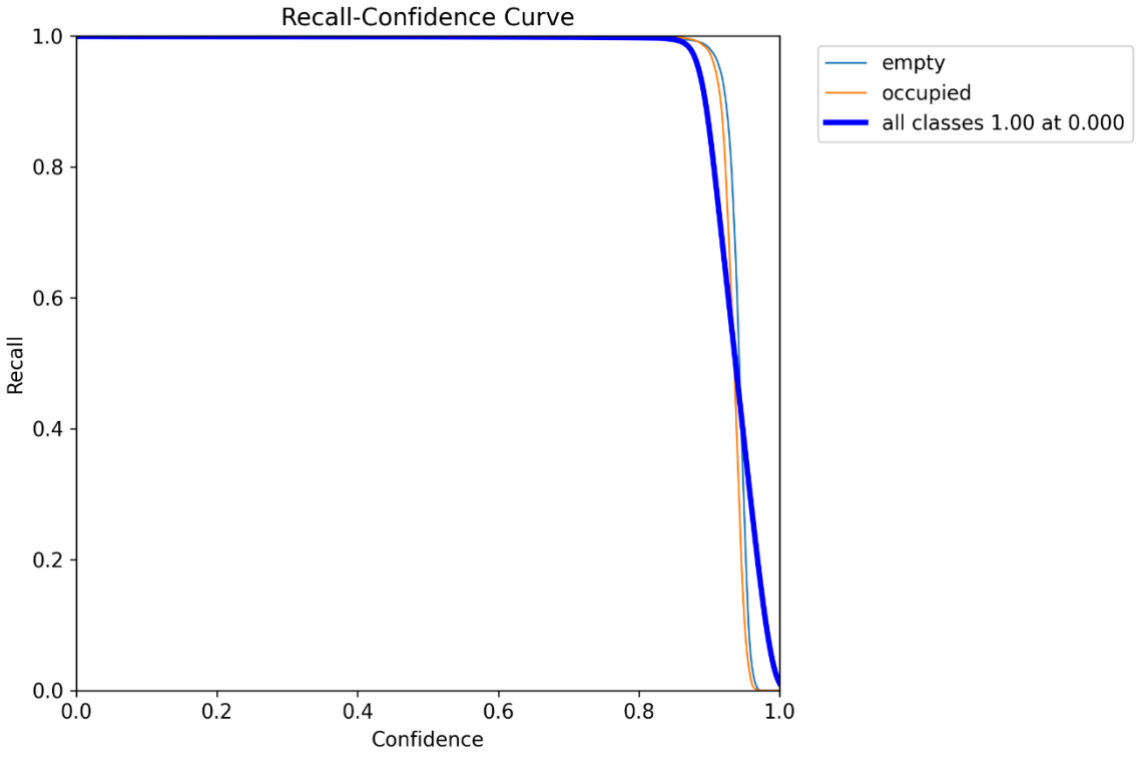



**Figura 42 Relación entre confianza y recuperación para distintos umbrales de detección.**

# 5. Resultados y discusión

En este capítulo se exponen y examinan los hallazgos alcanzados al aplicar diversos enfoques para resolver el problema planteado. Se evalúan varios modelos de ML para la clasificación de espacios específicos, incluyendo regresión logística, K-NN, árbol de decisión y K-Means. Además, se expone el desempeño de un modelo de DL, YOLOv8, utilizado para la detección y clasificación en imágenes de espacios ocupados o disponibles, comparando su precisión y eficiencia con las técnicas clásicas.

En los reportes de clasificación, las clases se representan numéricamente como 0 y 1, donde la clase 0 corresponde a lugares de estacionamiento vacíos y la clase 1 a lugares ocupados por vehículos. Las métricas principales incluyen: accuracy (proporción de aciertos totales), precision (grado de fiabilidad de las predicciones clasificadas como positivas), recall (capacidad del modelo para identificar correctamente todos los casos positivos), f1-score (balance entre precision y recall), y support (número de muestras por clase). El macro avg se obtiene al promediar de forma uniforme las métricas por clase sin considerar sus tamaños por clase, mientras que el weighted avg pondera las métricas según el número de muestras de cada clase, proporcionando una evaluación más representativa en datasets desbalanceados. La matriz de confusión visualiza los aciertos y errores de predicción, donde la diagonal principal muestra aciertos y los elementos fuera de ella corresponde a desaciertos en la clasificación. Para modelos de clustering, se emplean métricas como inercia (compacidad de clusters), silhouette score (separación entre clusters), homogeneidad y completitud (pureza y agrupación de clases), y V-measure (balance entre homogeneidad y completitud).

## 5.1 Modelos de ML

### 5.1.1 Modelo de Regresión Logística

```
Accuracy: 0.9716666666666667
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       913
           1       0.98      0.96      0.97       887

    accuracy                           0.97      1800
   macro avg       0.97      0.97      0.97      1800
weighted avg       0.97      0.97      0.97      1800
```

**Figura 43 Reporte de Clasificación del Modelo de Regresión Logística mostrando una precisión del 97.1% con métricas de precisión, recall y f1-score para clasificación binaria.**

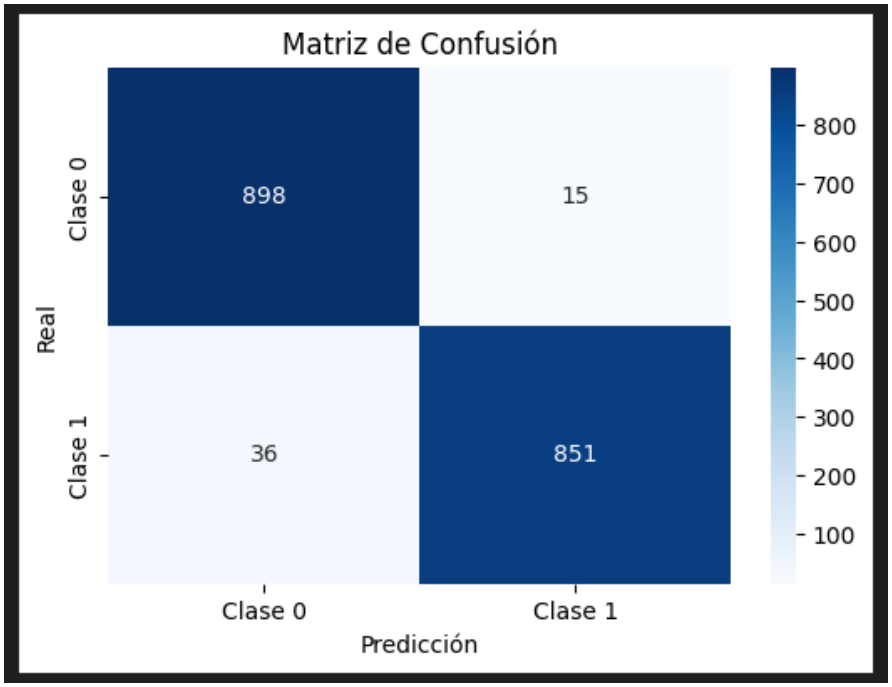


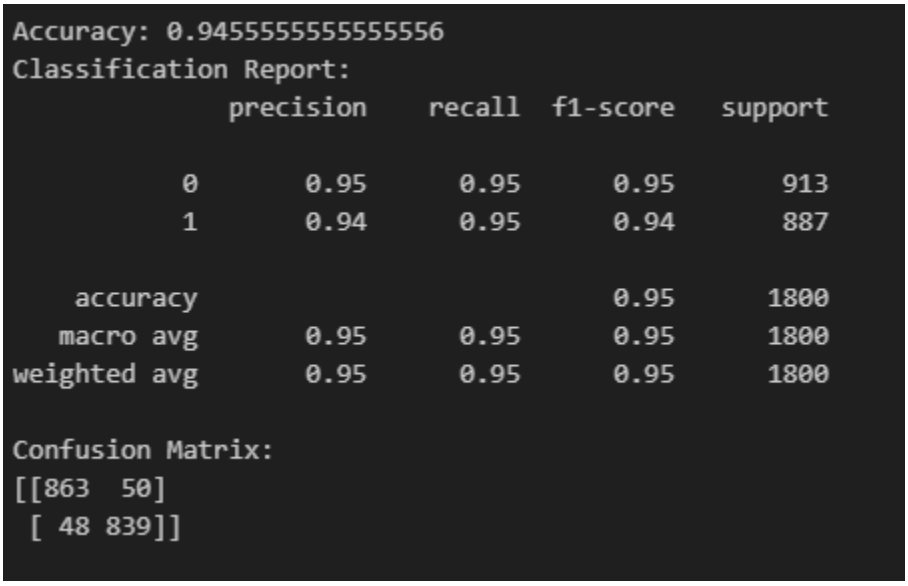

**Figura 44 Matriz de Confusión del Modelo de Regresión Logística mostrando la distribución de predicciones correctas e incorrectas para clasificación binaria con 898 verdaderos negativos, 851 verdaderos positivos, 15 falsos positivos y 36 FN.**

## 5.1.2 Modelo KNN

```
Accuracy: 0.9794444444444445
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       913
           1       1.00      0.96      0.98       887

    accuracy                           0.98      1800
   macro avg       0.98      0.98      0.98      1800
weighted avg       0.98      0.98      0.98      1800

Confusion Matrix:
[[913   0]
 [ 37 850]]
```

**Figura 45 Reporte de Clasificación del Modelo KNN mostrando una precisión del 97.94% con métricas de precisión, recall y f1-score para clasificación binaria, incluyendo matriz de confusión.**

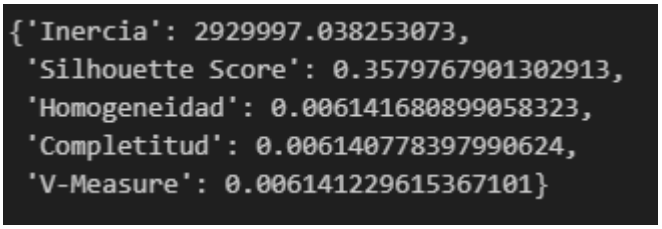
### 5.1.3 Modelo de árbol de decisión



```
Accuracy: 0.9455555555555556
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       913
           1       0.94      0.95      0.94       887

    accuracy                           0.95      1800
   macro avg       0.95      0.95      0.95      1800
weighted avg       0.95      0.95      0.95      1800

Confusion Matrix:
[[863  50]
 [ 48 839]]
```

**Figura 46 Reporte de Clasificación del Modelo de Árbol de Decisión mostrando una precisión del 94.56% con métricas de precisión, recall y f1-score para clasificación binaria, incluyendo matriz de confusión.**

### 5.1.4 Modelo Kmeans



```
{'Inercia': 2929997.038253073,
 'Silhouette Score': 0.3579767901302913,
 'Homogeneidad': 0.006141680899058323,
 'Completitud': 0.006140778397990624,
 'V-Measure': 0.006141229615367101}
```

**Figura 47 Métricas de Evaluación del Modelo K-means mostrando Inercia, Silhouette Score, Homogeneidad, Completitud y V-Measure para análisis de clustering.**

### 5.1.5 Análisis de resultados de Modelos de ML

**Tabla 1 Resultados de Modelos de Machine Learning**

| Modelo | Accuracy | Precision | Recall | F1-Score | Clase 0 Support | Clase 1 Support |
|---|---|---|---|---|---|---|
| **Regresión Logística** | 0.9717 | 0.96 (Clase 0)<br>0.98 (Clase 1) | 0.98 (Clase 0)<br>0.96 (Clase 1) | 0.97 (Clase 0)<br>0.97 (Clase 1) | 913 | 887 |
| **KNN** | 0.9794 | 0.96 (Clase 0)<br>1.00 (Clase 1) | 1.00 (Clase 0)<br>0.96 (Clase 1) | 0.98 (Clase 0)<br>0.98 (Clase 1) | 913 | 887 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Árbol de Decisión** | 0.9456 | 0.95 (Clase 0)<br>0.94 (Clase 1) | 0.95 (Clase 0)<br>0.95 (Clase 1) | 0.95 (Clase 0)<br>0.94 (Clase 1) | 913 | 887 |

**Tabla 2 Métricas de Clustering (K-means)**

| Métrica | Valor |
|---|---|
| Inercia | 2929997.038253073 |
| Silhouette Score | 0.357976790130291 |
| Homogeneidad | 0.006141068089058323 |
| Completitud | 0.006140778397990624 |
| V-Measure | 0.006141229615367101 |

- **Mejor modelo para clasificación**: KNN con 97.94% de accuracy
- **Modelo más balanceado**: Regresión Logística con métricas consistentes
- **Clustering**: K-means muestra baja homogeneidad y completitud, sugiriendo que los datos no se agrupan naturalmente en clusters bien definidos

## 5.2 Modelos de DL: YOLOv8

Para abordar la tarea más compleja de detectar espacios a partir de imágenes y luego clasificar su estado como ocupado o disponible, se recurrió a un modelo de DL utilizando YOLOv8. Este modelo, conocido principalmente por su desempeño en procesos de localización de objetos en imágenes, demostró ser una herramienta poderosa para identificar el espacio de interés en una imagen capturada por una cámara.
El modelo YOLOv8 no solo consiguió detectar el espacio dentro de la imagen, además, clasificó de manera precisa si el espacio estaba ocupado o disponible. A pesar de que este modelo requiere un alto poder computacional y tiempos de entrenamiento más largos, su capacidad para manejar imágenes en tiempo real y su notable precisión en la detección lo hicieron mucho más eficaz que los enfoques tradicionales de ML.

La implementación de YOLOv8 demostró que los métodos de DL son más adecuados cuando se trata de procesar imágenes y realizar tareas de detección y clasificación complejas, como en el caso del espacio ocupado o disponible, donde los modelos clásicos de ML no podían competir porque no está diseñado para procesar datos visuales sin intervención adicional.

**Métricas Cuantitativas Generales**

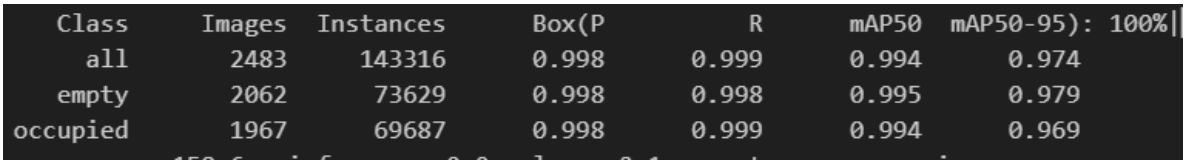



**Figura 48 Métricas de rendimiento del modelo YOLOv8 para detección de espacios de estacionamiento mostrando precisión, recall y mAP50 para las clases 'empty' y 'occupied' con un rendimiento general del 99.4% en mAP50.**

**Ejemplo de imágenes detectadas y clasificadas**


**Figura 49 Ejemplo de detección y clasificación en tiempo real del modelo YOLOv8 mostrando espacios de estacionamiento identificados con etiquetas 'occupied' y 'empty' junto con sus respectivos niveles de confianza en una imagen aérea de estacionamiento.**


**Figura 50 Ejemplos de imágenes aleatorias con predicciones generadas por YOLOv8.**

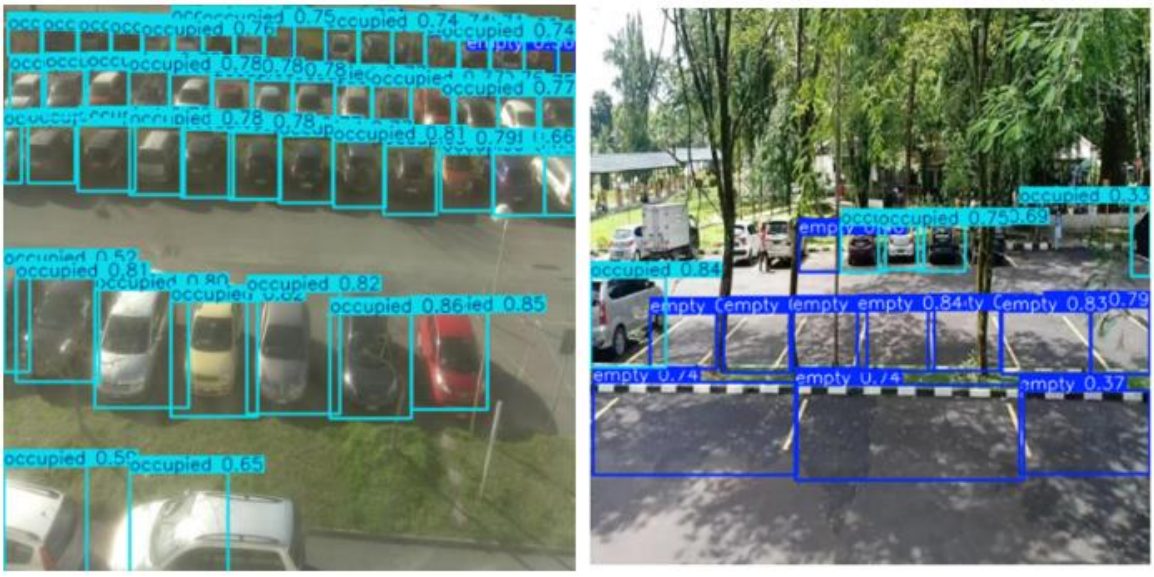



**Figura 51 Ejemplos de imágenes aleatorias con predicciones generadas por YOLOv8.**

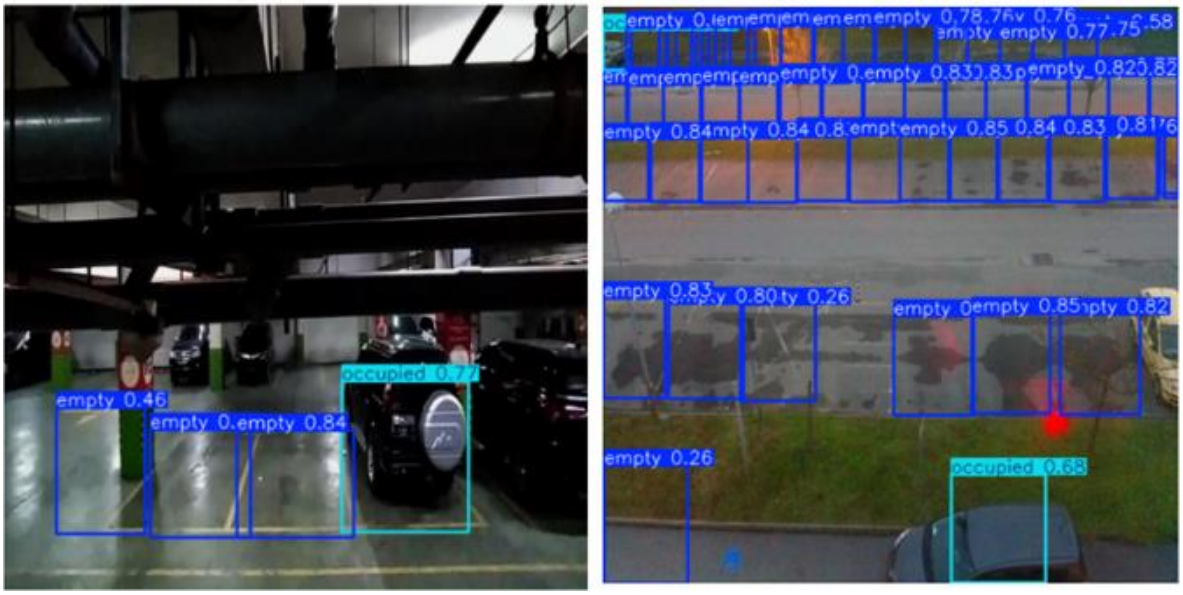



**Figura 52 Ejemplos de imágenes aleatorias con predicciones generadas por YOLOv8.**

# 6. CONCLUSIONES

El propósito central de la investigación actual fue elaborar una estrategia efectiva para la detección y clasificación de lugares de estacionamiento mediante el análisis de imágenes, empleando enfoques de ML y DL. Mediante el uso de distintos modelos de ML y la integración de YOLOv8, se consiguió desarrollar una solución robusta, capaz no solo de determinar si un lugar de estacionamiento se encuentra libre u ocupado, sino también de adaptarse a distintas condiciones meteorológicas y a diversas perspectivas capturadas por las cámaras.

## 6.1 Comparativa entre ML y DL

El análisis comparativo entre ML (Regresión Logística, KNN, Árbol de Decisión y KMeans) y el enfoque de DL (YOLOv8) evidenció diferencias notables en desempeño y precisión. Aunque los algoritmos de ML obtuvieron resultados aceptables en precisión y recall para tareas de clasificación básica, no lograron igualar la capacidad de YOLOv8 para procesar la complejidad visual de las fotografías y adaptarse a la variedad de escenarios de iluminación y clima. El modelo basado en DL demostró una detección de objetos altamente precisa, constituyéndose como una alternativa más sólida y efectiva para resolver el problema abordado.

## 6.2 Potencial de YOLOv8 en la detección de espacios de estacionamiento

La utilización de YOLOv8 con la finalidad de detectar elementos en tiempo real fue la alternativa más eficiente, alcanzando métricas sobresalientes de precisión (0.998) y recall (0.999). Los valores de mAP50 (0.994) y mAP50-95 (0.974) respaldaron la eficacia del modelo para identificar de forma precisa los espacios de estacionamiento bajo condiciones variables. Cabe destacar que la adopción de técnicas de data augmentation y un preprocesamiento adecuado fortalecieron la capacidad del modelo para generalizar frente a cambios en el conjunto de datos, tales como diferentes perspectivas, variaciones climáticas o iluminación, resaltando la importancia de contar con un dataset diverso y enriquecido para obtener resultados más generales y robustos.

## 6.3 Desafíos y Oportunidades

Si bien se obtuvieron resultados excepcionales, la presente investigación también identificó algunos desafíos y áreas de mejora:

1. Condiciones Climáticas Extremas: Aunque el modelo mostró un buen rendimiento bajo condiciones comunes (soleado, nublado y lluvioso), sería interesante evaluar su capacidad para trabajar en condiciones extremas, como neblina o tormentas severas, donde la visibilidad de los objetos puede verse significativamente afectada.

2. Escalabilidad e Implementación en Tiempo Real: A pesar de que YOLOv8 es muy rápido frente a otros modelos de DL, su implementación en tiempo real, especialmente en entornos con cámaras de baja resolución, podría verse limitada. Trabajos futuros podrían contemplar la optimización adicional del modelo para dispositivos de borde, así como la integración de técnicas como la cuantización de modelos para acelerar la inferencia.
3. Diversidad de Escenarios: Si bien el conjunto de datos PKLot fue muy útil, sería relevante enriquecer la colección de datos con fotografías de diferentes clases de estacionamientos, incluidos aquellos con diferentes tipos de vehículos o configuraciones más complejas.

## 6.4 Implicaciones Prácticas

Este trabajo tiene varias aplicaciones prácticas en el mundo real, concretamente en el sector de la gestión inteligente de estacionamientos y la automatización del control de acceso, el desarrollo de un sistema basado en la detección y clasificación mediante imágenes permite optimizar la utilización de los espacios disponibles y reducir la congestión vehicular y facilitar la implementación de sistemas de gestión automática de estacionamientos, tanto en entornos urbanos como en empresas o centros comerciales.
Además, la precisión y rapidez del modelo YOLOv8 permiten que este tipo de soluciones se implementen en dispositivos con capacidad de procesamiento limitada, como cámaras de seguridad o sistemas de visión artificial para estacionamientos, sin necesidad de servidores costosos.

## 6.5 Trabajo futuro

Como continuación de esta investigación, se propone:

- Expandir y diversificar el dataset con imágenes capturadas en distintas condiciones, tipos de estacionamientos y variedad de vehículos, para fortalecer la robustez del modelo.
- Investigar y aplicar técnicas de optimización para dispositivos de borde, tales como la cuantización y poda de modelos, para facilitar la implementación en tiempo real en hardware con recursos limitados.
- Evaluar la integración del sistema con plataformas en la nube para la gestión centralizada y el análisis en tiempo real. Se recomienda revisar los Apéndices A y B, donde se detallan respectivamente la arquitectura del sistema en la nube y el código para la ingesta de video desde cámaras de seguridad y detección en tiempo real, elementos que servirán como base para futuras implementaciones y mejoras.
- Desarrollar una interfaz de usuario intuitiva que facilite la interacción con el sistema, permitiendo a los usuarios visualizar resultados, gestionar configuraciones y monitorear el estado de los estacionamientos de forma eficiente.

## 6.6 Conclusiones

En resumen, esta investigación confirma que la aplicación de Deep Learning mediante YOLOv8 constituye un método altamente eficiente para la detección y clasificación de lugares de estacionamiento. Mientras

que los modelos tradicionales de Machine Learning ofrecieron soluciones aceptables para tareas básicas, fue el enfoque de Deep Learning el que permitió superar las limitaciones de precisión y adaptabilidad inherentes a las técnicas convencionales. La incorporación de técnicas de data augmentation y la construcción de un dataset variado demostraron ser elementos clave para alcanzar resultados generales sólidos, capaces de adaptarse a distintas condiciones y escenarios. El éxito del modelo abre nuevas oportunidades para la automatización y mejora en la gestión de estacionamientos, contribuyendo a una movilidad urbana más inteligente y al uso óptimo de los recursos.

# 7. REFERENCES


[1] Jalisco, I. d. I. E. y. G. d., «Crecimiento del parque vehicular en Jalisco y AMG 2023,» 30 Agosto 2024. [En línea]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://iieg.gob.mx/ns/wp-content/uploads/2024/08/Ficha-Informativa_Parque-vehicular-2023.pdf.

[2] Dimonoff, «Sensor MPS,» Agosto 30 2020. [En línea]. Available: https://www.dimonoff.com/es/soluciones/movilidad-aparcamientos-inteligente/productos/mps-sensor-ocupacion/.

[3] IBM, «¿Qué son las redes neuronales convolucionales?,» 29 Mayo 2024. [En línea]. Available: https://www.ibm.com/mx-es/topics/convolutional-neural-networks.

[4] V. Ramírez, «Estacionamientos ubicados en la Zona Metropolitana de Guadalajara son caros e inseguros,» El Occidental, 11 septiembre 2023. [En línea]. Available: https://www.eloccidental.com.mx/local/estacionamientos-ubicados-en-la-zona-metropolitana-de-guadalajara-son-caros-e-inseguros-10675096.html.

[5] E. Sara, «La pesadilla de buscar estacionamiento, sobre la mesa,» Moviliblog, 13 abril 2020. [En línea]. Available: https://blogs.iadb.org/transporte/es/la-pesadilla-de-buscar-estacionamiento-sobre-la-mesa/.

[6] Oracle, «¿Qué es la IA?,» [En línea]. Available: https://www.oracle.com/mx/artificial-intelligence/what-is-ai/#:~:text=la%20inteligencia%20artificial-,Terminolog%C3%ADa%20de%20la%20inteligencia%20artificial,clientes%20o%20jugar%20al%20ajedrez.

[7] McKinsey, «The state of AI in 2021,» QuantumBlack, 2021. [En línea].

[8] M. Schellhas, «Computer Vision: ¿cómo está transformando los negocios?,» Linkedin, 22 Enero 2024. [En línea]. Available: https://www.linkedin.com/pulse/computer-vision-c%C3%B3mo-est%C3%A1-transformando-los-negocios-matias-schellhas-jitzf/.

[9] IBM, «¿Qué es la detección de objetos?,» 03 01 2024. [En línea]. Available: https://www.ibm.com/mx-es/think/topics/object-detection.

[10] Parklio, «DETECT- DETECCIÓN DE PLAZAS DE APARCAMIENTO,» Parklio, 2024. [En línea]. Available: https://parklio.com/es/soluciones-de-estacionamiento/detect.

[11] Parklio, «¿Por qué el control de acceso al estacionamiento es vital para las empresas?,» [En línea]. Available: https://parklio.com/es/blog/por-que-el-control-de-acceso-al-estacionamiento-es-vital-para-las-empresas.

[12] Hikvision, «Descubra cómo la gestión inteligente de estacionamientos resuelve los problemas de estacionamiento",» [En línea]. Available: https://www.hikvision.com/es-co/newsroom/blog/Descubra-como-la-gestion-inteligente-de-estacionamientos-resuelve-los-problemas-de-estacionamiento/.

[13] C. I. Córdoba, «Prototipo de control y monitoreo para parqueaderos vehiculares,» Junio 2015. [En línea]. Available: https://geox.udistrital.edu.co/index.php/tekhne/article/view/10443/11460.

[14] Infaimon, «Inteligencia Artificial y Deep Learning,» Noviembre 2020. [En línea]. Available: https://www.interempresas.net/Robotica/Articulos/320015- La-vision-artificial-en-entorno-cientifico.html.

[15] C. A. H. Moreno, «Diseño de un sistema prototipo de uso de parqueaderos bajo disponibilidad, mediante IoT y sensórica,» Febrero 2024. [En línea]. Available: https://repositorio.uisek.edu.ec/bitstream/123456789/5221/1/Hidrobo%20Moreno%20Carlos%20Andr%c3%a9s%20.pdf https://repositorio.upse.edu.ec/bitstream/46000/11837/1/UPSE-TTI-2024-0033.pdf.

[16] Administración de sistemas, «Nube privada alojada vs On-Premise: Comparación y tendencias en el mercado,» Julio 2023. [En línea]. Available: https://administracionde sistemas.com/nube-privada-alojada-vs-on-premise-comparacion-y-tendencias-en-el-mercado/.

[17] A. T. Martín, «Investigación de Detección de Bordes a Radiografías con Deep Learning,» Mayo 2023. [En línea]. Available: https://oa.upm.es/74981/1/TFG_ALBERTO_TOMAS_MARTIN.pdf.

[18] IBM, «¿Qué es la segmentación de imágenes?,» [En línea]. Available: https://www.ibm.com/mx-es/topics/image-segmentation.

[19] V. Romero, «SEGMENTACIÓN DE LUGARES DISPONIBLES EN ESTACIONAMIENTOS HACIENDO USO DE REDES NEURONALES PULSO-ACOPLADAS,» Noviembre 2021. [En línea]. Available: file:///Users/ichantop/Downloads/2547-9820-2-PB.pdf.

[20] Z. Keita, «Explicación de la detección de objetos YOLO,» Enero 2024. [En línea]. Available: https://www.datacamp.com/es/blog/yolo-object- detection-explained.

[21] M. Pérez, «Reconocimiento eficiente de caras mediante Deep Learning a partir de imágenes en el espectro visible,» Julio 2021. [En línea]. Available: https://uvadoc.uva.es/bitstream/handle/10324/50029/TFG-G5204.pdf;jsessionid=868A33160A5EA58369EBD866E7237488?sequence=1.

[22] P. O. L. S. S. J. E. B. J. A. K. A. Almeida, «PKLot – A robust dataset for parking lot classification,» 2015. [En línea]. Available: https://public.roboflow.com/object-detection/pklot.

[23] B. Fedorak, «Parking Lot Dataset,» 2018. [En línea]. Available: https://www.kaggle.com/datasets/blanderbuss/parking-lot-dataset/data.

[24] AWS, «¿Cuál es la diferencia entre el ML y el aprendizaje profundo?,» 2024. [En línea]. Available: https://aws.amazon.com/es/compare/the-difference-between-machine-learning-and-deep-learning/.

[25] Capterra, «Parking Management Software,» [En línea]. Available: https://www.capterra.com/sem-compare/parking-management-software/?utm_source=ps-google&utm_medium=ppc&utm_campaign=:1:CAP:2:COM:3:All:4:INTL:5:BAU:6:SOF:7:Desktop:8:BR:9:Parking_Management&network=g&gclid=CjwKCAjwvO7CBhAqEiwA9q2YJZP71532ccEcCkNtdAA2fkI3Dr.

[26] H. Rivera, «ANN vs CNN vs RNN: Entendiendo la diferencia,» Codoid, 19 01 2025. [En línea]. Available: https://codoid.com/ai/ann-vs-cnn-vs-rnn-understanding-the-difference/.

[27] AWS, «¿Qué es una RNN (red neuronal recurrente)?,» 2023. [En línea]. Available: https://aws.amazon.com/es/what-is/recurrent-neural-network/.

[28] K. Pykes, «Tutorial sobre curvas de aprendizaje: ¿Qué son las curvas de aprendizaje?,» 09 Marzo 2022. [En línea]. Available: https://www.datacamp.com/tutorial/tutorial-learning-curves.

[29] AWS, «¿Qué es Python?,» 2024. [En línea]. Available: https://aws.amazon.com/es/what-is/python/.

[30] python, «os — Interfaces misceláneas del sistema operativo,» Abril 12 2025. [En línea]. Available: https://docs.python.org/es/3.10/library/os.html.

[31] Python, «pillow 11.3.0,» 2025. [En línea]. Available: https://pypi.org/project/pillow/.

[32] Matplotlib, «Matplotlib: Visualization with Python,» 2025. [En línea]. Available: https://matplotlib.org/.

[33] V. Chugh, «Tutorial de pandas en Python: La guía definitiva para principiantes,» 09 02 2025. [En línea]. Available: https://www.datacamp.com/es/tutorial/pandas.

[34] scikit-learn, «scikit-learn,» 2025. [En línea]. Available: https://scikit-learn.org/stable/.

[35] Ultralytics Inc, «Presentación de Ultralytics yolo11,» 2025. [En línea]. Available: https://docs.ultralytics.com/es/.

[36] H. Nguyen, «MLflow: una herramienta MLOps moderna para la colaboración en proyectos de datos,» 11 02 2023. [En línea]. Available: https://medium.com/sfu-cspmp/mlflow-a-modern-mlops-tool-for-data-project-collaboration-704ca299d9c3.

[37] A. Martinazzo, «Modelos de aprendizaje automático en tiempo real en la vida real,» NU, 23 septiembre 2022. [En línea]. Available: https://building.nubank.com.br/es/modelos-de-aprendizaje-automatico-en-tiempo-real-en-la-vida-real/.

[38] Zainab, «yolov8-parking-spot_dataset,» Robloflow, 14 Abril 2024. [En línea]. Available: https://universe.roboflow.com/zainab-t1no1/yolov8-parking-spot.

[39] Robloflow, «cnrpark-hnvuk_dataset,» Marzo 2025. [En línea]. Available: https://universe.roboflow.com/projects-oqpad/cnrpark-hnvuk.

[40] Digikey, «Productos,» Julio 2025. [En línea]. Available: https://www.digikey.com.mx/.

[41] E. ROBERTO, «biorobotics,» Marzo 2020. [En línea]. Available: https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/reconocimiento_de_patrones/tutoriales/YOLO-Introducci%C3%B3n-e-implementaci%C3%B3n-.pdf.

[42] O. Adeojo, «Desplegando la arquitectura y eficiencia de Fast y Faster R-CNN para la detección de objetos,» Agosto 2023. [En línea]. Available: https://docs.kanaries.net/es/topics/Python/fast-rcnn.

[43] Universidad Europea, «Aprendizaje supervisado y no supervisado,» Octubre 2022. [En línea]. Available: https://universidadeuropea.com/blog/aprendizaje-supervisado-no-supervisado/.

[44] IBM, «Funcionamiento de SVM,» Agosto 2021. [En línea]. Available: https://www.ibm.com/docs/es/spss-modeler/saas?topic=models-how-svm-works.

[45] IBM, «¿Qué es KNN?,» [En línea]. Available: https://www.ibm.com/mx-es/topics/knn.

[46] IBM, «¿Qué es el clustering?,» [En línea]. Available: https://www.ibm.com/es-es/topics/clustering.

[47] Gamco, «¿Qué es LSTM: Long short-term memory?,» 2021. [En línea]. Available: https://gamco.es/glosario/lstm-long-short-term-memory/.

[48] IBM, «¿Qué es la segmentación semántica?,» Julio 2021. [En línea]. Available: https://www.ibm.com/mx-es/topics/semantic-segmentation.

[49] C. S. Dave Bergmann, «¿Qué es PyTorch?,» IBM, 23 noviembre 2023. [En línea].

[50] The Linux Foundation, «TorchVision Object Detection Finetuning Tutorial,» 2024. [En línea]. Available: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html.

[51] Microsoft, «Evaluación de los resultados del experimento de aprendizaje automático automatizado,» 03 septiembre 2024. [En línea]. Available: https://learn.microsoft.com/es-es/azure/machine-learning/how-to-understand-automated-ml?view=azureml-api-2#confusion-matrix.

[52] R. Díaz, «Métricas de Clasificación,» The Machine Learners, [En línea]. Available: https://www.themachinelearners.com/metricas-de-clasificacion/.

[53] J. Sanchez, «Evaluación de algoritmos de detección de objetos basados en deep learning para detección de incidencias en carreteras,» 30 septiembre 2020. [En línea]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://uvadoc.uva.es/bitstream/handle/10324/43277/TFG-G4450.pdf?sequence=1.

[54] AWS, «¿Qué es una GAN?,» [En línea]. Available: https://aws.amazon.com/es/what-is/gan/.

[55] Quanergy, «LiDAR,» [En línea]. Available: https://quanergy.com/es/what-is-lidar/.

[56] AWS, «¿Qué es el aprendizaje mediante refuerzo?,» [En línea]. Available: https://aws.amazon.com/es/what-is/reinforcement-learning/.

[57] AWS, «Amazon Sagemaker,» [En línea]. Available: https://aws.amazon.com/es/sagemaker/.

[58] AWS, «Amazon Rekognition,» [En línea]. Available: https://aws.amazon.com/es/rekognition/?nc=sn&loc=0.

[59] Google Cloud, «Cloud Functions,» [En línea]. Available: https://console.cloud.google.com/marketplace/product/google-cloud-platform/cloud-functions?pli=1.

[60] C. Polo, «Cómo generar un modelo de Machine Learning a partir de todos los datos generados por un proyecto IOT,» Seidor, 12 febrero 2024. [En línea]. Available: https://www.seidor.com/es-es/blog/machine-learning-generacion-iot.

[61] U. o. Pisa, «cnrpark-ext-mdzfq_dataset,» Roboflow, Septiembre 2023. [En línea]. Available: https://universe.roboflow.com/university-of-pisa-cddhu/cnrpark-ext-mdzfq.

[62] mi3ad, «PKLot available Dataset,» Roboflow, Diciembre 2023. [En línea]. Available: https://universe.roboflow.com/mi3ad/pklot-available.

# 8. Apéndice A: Arquitectura del Sistema en la Nube

## 8.1 Visión General

La arquitectura propuesta se basa en un diseño de microservicios nativos en la nube, diseñada para ofrecer escalabilidad y procesamiento en tiempo real. El sistema está compuesto por tres capas principales: presentación, servicios backend, y datos.

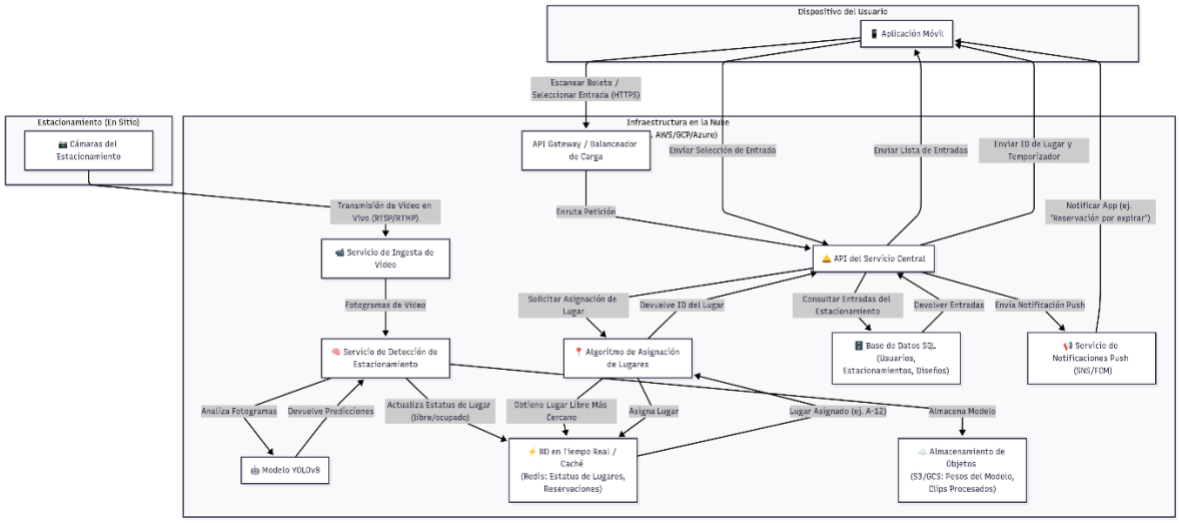



**Figura 53 Arquitectura completa del sistema de gestión de estacionamientos inteligente.**

## 8.2 Componentes Principales

### 8.2.1 Capa de Presentación - Aplicación Móvil

La aplicación móvil descargable permite a los usuarios:

- **Registro de vehículos**: Vinculación de placas con la cuenta del usuario
- **Usuarios no registrados**: Escaneo de código QR del ticket de estacionamiento
- **Asignación inteligente**: Selección de edificio destino para asignar el espacio más cercano
- **Gestión dinámica**:
    - Reasignación automática si otro usuario ocupa el espacio

- Liberación por tiempo límite si no se usa
- Bloqueo exclusivo una vez ocupado el espacio

### 8.2.2 Servicios Backend

**API de Servicio Central**: Maneja peticiones de usuarios, reservaciones y lógica de negocio.

**Servicio de Ingesta de Video**: Procesa transmisiones en vivo de cámaras RTSP/RTMP, extrae fotogramas y los preprocesa para análisis.

**Servicio de Detección de Estacionamiento**: Utiliza modelo YOLOv8 para detectar y clasificar espacios como ocupados o disponibles en tiempo real.

### 8.2.3 Capa de datos

**Base de Datos SQL**: Almacena información de usuarios, configuración de estacionamientos e historial de transacciones.

**Base de Datos Redis**: Gestiona datos en tiempo real como estatus de ocupación, reservaciones activas y caché de consultas.

## 8.3 Características Técnicas

### 8.3.1 Escalabilidad y Disponibilidad

- Escalabilidad horizontal independiente por servicio
- Auto-escalado basado en métricas de uso
- Redundancia en múltiples zonas de disponibilidad
- Tolerancia a fallos con circuit breaker patterns

## 8.4 Seguridad y Comunicación

- Comunicación segura TLS/SSL entre servicios
- Autenticación JWT stateless
- API Gateway para balanceo de carga y rate limiting
- Comunicación asíncrona mediante message queues

## 8.5 Tecnologías Implementadas

- **Contenedores**: Docker y Kubernetes para orquestación
- **Infraestructura**: AWS/Azure con servicios gestionados
- **Monitoreo**: Métricas, logs y alertas distribuidas

- **CI/CD**: Pipelines automatizados de despliegue

## 8.6 Beneficios del Diseño

Este diseño desacoplado proporciona alta disponibilidad, manejo eficiente de transacciones de usuario y tareas computacionalmente intensivas, escalabilidad granular, y facilidad de mantenimiento con desarrollo independiente de servicios.

# 9. Apéndice B: Código para la ingesta de video desde cámaras de seguridad y detección en tiempo real

En este apéndice se muestra un ejemplo de código sugerido para capturar imágenes desde una cámara de seguridad de un estacionamiento y procesarlas mediante un modelo YOLOv8, con el fin de detectar y clasificar espacios ocupados o libres en tiempo real.

## 9.1 Fragmento de código

El fragmento de código presentado permite la captura continua de imágenes desde una cámara de seguridad de un estacionamiento y su procesamiento en tiempo real mediante un modelo YOLOv8 previamente entrenado. La lógica implementada evalúa cada fotograma capturado para detectar la presencia de vehículos u objetos de interés y, en caso de detección positiva, almacena la imagen en disco únicamente si ha transcurrido un intervalo mínimo de tiempo desde la última captura. De esta forma, se optimiza el uso de almacenamiento y se evitan registros redundantes, garantizando que solo se conserven evidencias relevantes del estado de ocupación de los espacios de estacionamiento. Además, el sistema permite la visualización inmediata de los resultados de la detección, facilitando su supervisión y validación en tiempo real.

```
from ultralytics import YOLO
import cv2
import time
from datetime import datetime

# Cargar modelo YOLOv8 entrenado
model = YOLO("best.pt")  # Cambia por la ruta de tu modelo

# Abrir cámara de seguridad (local o IP)
cap = cv2.VideoCapture(0)  # 0 = webcam local, o usa RTSP/URL de cámara real

# Intervalo mínimo entre capturas en segundos
save_interval = 30  # por ejemplo, guardar máximo cada 30 segundos
last_saved_time = 0  # para controlar cuándo fue la última imagen guardada
```

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Ejecutar detección
    results = model.predict(frame, conf=0.5)

    # Extraer detecciones
    detections = results[0].boxes

    # Obtener tiempo actual
    current_time = time.time()

    # Si hay detecciones y pasó suficiente tiempo desde la última
    if len(detections) > 0 and (current_time - last_saved_time) >=
save_interval:
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"detected_{timestamp}.jpg"
        cv2.imwrite(filename, frame)
        print(f"Imagen guardada: {filename}")
        last_saved_time = current_time  # actualizar tiempo de última
captura

    # Mostrar resultado en tiempo real
    cv2.imshow("Detección en tiempo real", results[0].plot())

    # Salir con tecla 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Liberar recursos
cap.release()
cv2.destroyAllWindows()
```

# 10. Apéndice C: Estimación de costos para sistema de sensores SPI (Ejemplo con 275 lugares de estacionamiento)

A continuación, se presenta un ejemplo de estimación de costos para implementar un sistema de detección de ocupación en un estacionamiento con 275 espacios, utilizando sensores SPI. La cantidad de 275 lugares se utiliza únicamente como referencia para dimensionar el sistema y estimar costos aproximados. El número real puede variar según el proyecto.

Para este caso, se considera un diseño modular, donde se agrupan los sensores en bloques de aproximadamente 25, controlados por un microcontrolador con apoyo de multiplexores. Se requiere cableado, alimentación eléctrica y programación del sistema para visualización de datos en tiempo real.

Los costos aproximados en pesos mexicanos (MXN), son los siguientes (todos los precios unitarios fueron consultados en Digi-Key y corresponden a disponibilidad pública en julio de 2025) [40]:

- Sensores VL53L1X (275 unidades)
    - Costo por unidad: $425.82 MXN
    - Subtotal: $117,100.50 MXN
- Microcontroladores ESP32-PICO-DEVKITM-2 (11 unidades)
    - Costo por unidad: $198.81 MXN
    - Subtotal: $2,186.91 MXN
- Multiplexores 74HC4067 (18 unidades)
    - Costo por unidad: $120.64 MXN
    - Subtotal: $2,171.52 MXN
- Cableado, conectores y PCB (UTP Cat6 blindado + JST)
    - Estimado: $9,900 – $24,750 MXN
- Fuente de poder Mean Well LRS-350-5 (5V, 60A)
    - Cantidad: 1 unidad
    - Costo: $1,428.69 MXN
- Reguladores buck Pololu 5570 (3.3 V, 6 A)
    - Cantidad: 6 unidades
    - Costo por unidad: $592.97 MXN
    - Subtotal: $3,557.82 MXN
- Instalación de sensores (275 unidades)
    - Estimado: $49,500 – $247,500 MXN
- Desarrollo e integración de software (ESP-IDF, Node.js, MongoDB Atlas, React, Flutter)

- Estimado: $90,000 – $450,000 MXN

El total estimado del proyecto para implementar un sistema de monitoreo de 275 espacios de estacionamiento es de $275,845 a $848,695 MXN, dependiendo del alcance, calidad de componentes, nivel de instalación y complejidad del software requerido. Este sistema ofrece permite monitorear cada lugar de forma individual, y puede escalarse por módulos. No obstante, implica una instalación compleja, requiere mantenimiento periódico y la participación de personal técnico especializado para su montaje, programación e integración.