Universidad Miguel Hernández de Elche

MÁSTER UNIVERSITARIO EN ROBÓTICA

“EVALUACIÓN DETECCIÓN DE OBJETOS EN INTERIOR DE EDIFICIOS UTILIZANDO LA RED NEURONAL YOLOv8”

Trabajo de Fin de Máster

Curso académico 2023-2024

Autor: Alejandro Orgiler Castelló
Tutor/es: Luis Payá Castelló
María Flores Tenza

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todos mis compañeros, que, con su apoyo, colaboración y ánimo, hicieron que este curso académico fuera más llevadero y enriquecedor. Su disposición para compartir conocimientos y ayudarnos mutuamente en los momentos de dificultad.

De igual manera, agradezco profundamente a los compañeros del laboratorio, quienes, con su amabilidad, agudos comentarios y los conocimientos compartidos, han contribuido al progreso de este trabajo. En especial, quiero extender mi gratitud a mis tutores Luis Payá y María Flores, cuyas orientaciones, sabiduría y constante apoyo fueron fundamentales para la culminación de este proyecto. Sus consejos han sido una guía invaluable a lo largo de este proceso.

Por último, pero no menos importantes, quiero agradecer a mi familia por su paciencia, apoyo y afecto incondicional durante este tiempo. En especial, a mi hermana, cuyo respaldo ha sido esencial para mí. Sin su ayuda y comprensión, todo este curso habría sido mucho más difícil de sobrellevar.

ÍNDICE GENERAL:

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

1.1 MOTIVACIÓN

El presente trabajo se desarrolla en el ámbito de la robótica móvil, una disciplina que sigue ganando relevancia y ejerciendo un creciente impacto en la sociedad. Las ventajas del uso de robots móviles autónomos e inteligentes son múltiples, sobre todo en ciertos ámbitos, como por ejemplo la exploración de entornos peligrosos o de difícil acceso para los seres humanos (Figura 1- 1), la asistencia a personas (Figura 1- 2), prestación de servicios cotidianos (Figura 1- 3) o la ejecución de tareas de vigilancia y seguridad (Figura 1- 4).

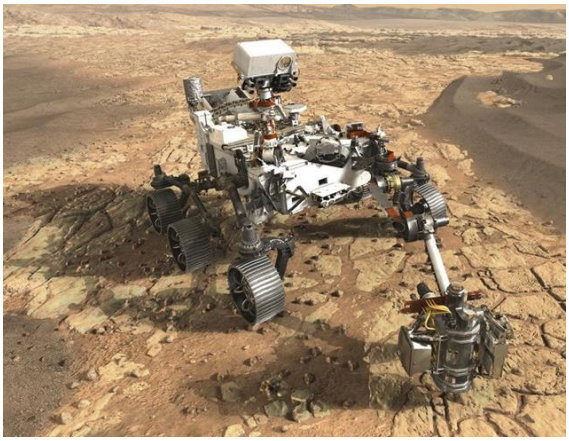


Figura 1- 1. Robot explorador Perseverence usado por la NASA para la exploración del planeta Marte. (Imagen sacada de [110])

Figura 1- 2. Robot Bellabot diseñado por Pudu Robotics para entrega de alimentos. (Imagen sacada de [111])

Figura 1- 3. Robot Roomba dieseñado por iRobot como aspirador doméstico. (Imagen sacada de [112])

Figura 1- 4. Robot RB-WATCHER diseñado por Robotnik destinado a tareas de vigilancia autónoma. (Imagen sacada de [113])

Los robots móviles autónomos tienen la capacidad de desplazarse de un punto a otro para cumplir sus objetivos sin la intervención de operadores humanos [1], es decir, funcionan de manera independiente sin recibir comandos externos,

utilizando únicamente los datos obtenidos a través de sus sensores [2]. Para ello, deben ser capaces de ejecutar con precisión las tareas esenciales de navegación, como la localización, la creación de mapas, la planificación de trayectorias y la detección de objetos. La correcta ejecución de estas permite al robot identificar obstáculos, modelar el entorno y elegir rutas seguras para evitar colisiones y desplazarse con precisión [3, 4]. Entre los sensores utilizados para estos fines destacan el sónar, el láser, el GPS y los sistemas de visión siendo estos últimos especialmente útiles por la cantidad de detalles que proporcionan, como color, forma y textura de los objetos. Esto es fundamental en aplicaciones avanzadas como la segmentación semántica y la detección de objetos, aspectos clave para mejorar la autonomía y la seguridad en la navegación.

Los sistemas de visión convencionales presentan, sin embargo, limitaciones en cuanto a su campo de visión, lo que puede afectar a su rendimiento en ciertas situaciones. Por ejemplo, un campo de visión limitado puede dificultar la detección de obstáculos, aumentando el riesgo de colisiones. Además, cuando el robot cambia de dirección o pierde de vista un objeto, debe detenerse y girar para volver a localizarlo. Asimismo, esta restricción puede generar dificultades al atravesar puertas o caminos estrechos, donde la visión reducida compromete la capacidad del robot para navegar con precisión [5].Para superar estas limitaciones, se emplean sistemas de visión omnidireccionales como las cámaras con lentes ojo de pez (*fisheye*), que ofrecen un campo de visión significativamente amplio y permiten al robot capturar una panorámica de su entorno, lo que es esencial para la detección de obstáculos y la navegación segura en entornos complejos. No obstante, tanto los sistemas de visión convencionales como los omnidireccionales presentan limitaciones adicionales en entornos interiores con cambios de iluminación, tales como sombras, luz artificial y áreas con baja y alta luminosidad, planteando desafíos adicionales que suelen afectar al rendimiento de estos sistemas de manera más notable que en exteriores.

La detección de objetos en imágenes ojo de pez representa un desafío particular. La gran distorsión radial característica de estas imágenes complica la localización y clasificación precisa de los objetos, ya que, a medida que los objetos se alejan del centro de la imagen, la distorsión aumenta, dificultando su correcta representación y localización. En cámaras convencionales de perspectiva estándar, los objetos suelen representarse mediante cajas delimitadoras (*bounding boxes*) rectangulares [114, 115]. Rashed et al. en [114] indican que en el caso de las imágenes captadas con las cámaras *fisheye*, esta representación presenta limitaciones, ya que las cajas rectangulares no se adaptan a las formas curvas de los objetos ubicados en los bordes de la imagen, lo que puede generar errores en la detección. Por ello, indican la necesidad de desarrollar métodos de procesamiento más sofisticados, como cajas curvadas, elipses o polígonos, que se ajusten mejor a los contornos distorsionados de los objetos.

Además, varios factores incrementan la complejidad de la detección de objetos en estas imágenes. Primero, la distorsión radial transforma las líneas rectas en curvas, impidiendo el uso de modelos convencionales de detección que requieren contornos rectilíneos. Segundo, los modelos adaptados a estas distorsiones requieren de procesamiento computacional intensivo y redes

neuronales convolucionales personalizadas, lo que aumenta el costo de procesamiento. Finalmente, la anotación de objetos en imágenes distorsionadas es costosa y exige curvatura de los objetos [114].

A pesar de estos desafíos, las cámaras ojo de pez presentan importantes ventajas en aplicaciones que requieren un campo de visión amplio, como la conducción autónoma, la vigilancia y la automatización industrial. En vehículos autónomos, por ejemplo, estas cámaras permiten una visión panorámica de 360 grados, esencial para tareas de detección de obstáculos y maniobras de seguridad, como el frenado de emergencia [114]. En la vigilancia, un solo dispositivo con lente ojo de pez puede cubrir grandes áreas, ideal para el monitoreo de personas y eventos en movimiento. En la industria, son clave en la automatización de procesos, permitiendo una manipulación y detección más precisa de objetos, especialmente en entornos con amplios espacios de trabajo [115].

1.2 OBJETIVOS

Este estudio tiene como finalidad determinar si YOLOv8 puede ser utilizado de manera confiable en un sistema de visión para un robot móvil, que podría navegar y operar en interiores de edificios.

Para lograr esto, se procederá a crear una base de datos de imágenes que capturen los objetos de interés en diferentes condiciones. Estas imágenes serán tomadas con una cámara Garmin VIRB, no solo en situaciones normales, sino también bajo condiciones de distorsión (imágenes capturadas con lentes ojo de pez), con el fin de poner a prueba la capacidad de YOLOv8 para reconocer objetos en situaciones que simulen posibles desafíos en un entorno real.

Posteriormente, se procederá al entrenamiento de la red neuronal utilizando estas imágenes, permitiendo analizar y medir la precisión y robustez del modelo en la identificación de los objetos. En este estudio se pretende determinar, en última instancia, la viabilidad de implementar YOLOv8 de manera efectiva en robots móviles, con la finalidad de facilitar la navegación autónoma en el interior de edificios.

1.3 APARTADOS

El presente trabajo se encuentra estructurado de la siguiente forma:

- ***Capítulo 2:*** En este capítulo se destacan los diferentes tipos de sensores utilizados por los robots móviles para obtener información del entorno, diferenciando entre sensores propioceptivos, que miden el estado interno del robot, y sensores exteroceptivos, que recogen datos del entorno exterior. Se describen en detalle aquellos sensores que proporcionan información de la escena útil para la detección e identificación de objetos, detallando sus ventajas, limitaciones y aplicaciones específicas en robótica. Además, se introduce el uso de cámaras de visión como sensores exteroceptivos, explicando sus

configuraciones, desde sistemas monoculares hasta cámaras omnidireccionales. Finalmente, se hace una breve mención a los métodos avanzados para la detección y reconocimiento de objetos en imágenes, fundamentales para mejorar la autonomía y precisión de los robots móviles.

- ***Capítulo 3:*** En este capítulo se detalla la implementación de las redes neuronales convolucionales (CNNs), con un enfoque particular en la arquitectura YOLO (You Only Look Once) para la detección en la detección de objetos. Además, se describe la evolución de las versiones de YOLO, destacando las mejoras en precisión y eficiencia. También, se explica el proceso de creación del *dataset* (conjunto de datos) de imágenes capturadas en interiores, tanto con distorsión (formato ojo de pez) como sin distorsión, utilizando la cámara Garmin VIRB 360. Además, se aborda la anotación y preparación de las imágenes para entrenar la red neuronal.

- ***Capítulo 4:*** Este capítulo se centra en la evaluación de la eficacia de la red neuronal YOLO aplicada a la detección de objetos en entornos interiores. Por tanto, se realizaron diversos experimentos utilizando diferentes modelos de YOLOv8 en imágenes convencionales y de ojo de pez, capturadas con la cámara Garmin VIRB. Se proporcionan detalles sobre la configuración del entorno de trabajo, los resultados de las pruebas y el modelo seleccionado para los experimentos posteriores. También se abordan aspectos relevantes de la librería Ultralytics y la metodología de entrenamiento utilizada, que incluye la selección de hiperparámetros y el ajuste del modelo.

  Además, se analizan los resultados obtenidos en diferentes configuraciones, como, por ejemplo, variando el número de épocas, tamaño de imágenes, etc. Se compararán los rendimientos de los experimentos en los distintos conjuntos de datos creados. Finalmente, se destacan los experimentos que lograron un equilibrio óptimo entre las métricas clave y se discuten los resultados más destacados.

- ***Capítulo 5:*** Se exponen las conclusiones extraídas, así como las posibles mejoras que se podrían realizar en un futuro.

CAPÍTULO 2. ESTADO DEL ARTE

2.1 ROBOTS MÓVILES

Los robots móviles son aquellos que tienen la capacidad de desplazarse en el entorno, a diferencia de los robots manipuladores que permanecen fijos. Esta movilidad ha permitido que los robots móviles se apliquen en diversos ámbitos, como la industria, la agricultura, el hogar y muchas otras áreas. A lo largo de los años, estos robots han sido utilizados para realizar tareas que antes eran llevadas a cabo por los seres humanos, en situaciones que podrían resultar peligrosas, como misiones de rescate [6, 7, 8], exploraciones en el espacio [9, 10] o en minas [11, 12]. Recientemente, su campo de aplicación se ha ampliado significativamente, permitiendo su uso en sectores como la entrega de paquetes [13, 14], apoyo en la medicina [15, 16], enseñanza [17], restaurantes [18, 19, 20], aeropuertos [21, 22, 23] y tareas cotidianas como la limpieza [24, 25] o la asistencia a personas [26, 27]. Para lograr que los robots móviles actúen de forma autónoma, es crucial el desarrollo de cuatro áreas tecnológicas: locomoción, percepción, cognición y navegación [28, 29].

2.2 NAVEGACIÓN AUTÓNOMA

La navegación autónoma es un desafío clave en muchas aplicaciones de robots móviles, ya que requiere que el robot se desplace en entornos impredecibles y dinámicos sin asistencia humana. Para una navegación autónoma exitosa, el robot debe localizarse correctamente en el entorno y ser capaz de crear un mapa mientras se mueve, proceso conocido como *Simultaneous Localization and Mapping* (SLAM) [30, 31, 32]. A partir de este mapa, el robot debe planificar la trayectoria [33, 34, 34, 3] y evitar obstáculos. La localización es un aspecto fundamental, ya que el robot debe conocer su posición y orientación para poder calcular la trayectoria correcta.

Un elemento clave para la autonomía es la adquisición de información a través de sensores, que pueden ser internos, midiendo el estado del robot, o externos, capturando datos del entorno, dependiendo del tipo de robot (terrestre, aéreo o acuático) y su entorno operativo [36]. En este contexto, las técnicas de visión, como *Visual-SLAM*, han ganado popularidad por su capacidad para proporcionar soluciones de localización y mapeo en entornos complejos, mejorando no solo la navegación autónoma en escenarios interiores y exteriores, sino también la detección y el reconocimiento de objetos, cruciales para aplicaciones como la conducción autónoma y la robótica colaborativa [37]. Este sistema permite a los robots mapear su entorno, localizarse y reconocer elementos dentro de él, utilizando imágenes capturadas para estimar su movimiento y características del entorno. Sin embargo, los errores acumulativos en la estimación de la trayectoria exigen la incorporación de técnicas adicionales para asegurar la precisión del sistema en escenarios dinámicos y complejos [38, 39, 40]. El avance de la inteligencia artificial ha potenciado estas técnicas mediante el aprendizaje

profundo, optimizando aspectos como la odometría visual, la detección de cierres de bucles y la creación de mapas detallados, permitiendo al sistema identificar y localizar objetos en el entorno. Además de, una navegación segura y precisa, así como la identificación de dinámicas clave en entornos no estructurados [41, 42, 43]. No obstante, la detección y el reconocimiento de objetos siguen siendo áreas de investigación activa dentro de la navegación autónoma. Por ejemplo, Crespo et al. en [44] abordan la incorporación de información semántica de alto nivel, como la identificación de objetos y las relaciones entre ellos, para optimizar la interpretación contextual del entorno y mejorar los sistemas de navegación autónoma.

2.3 SENSORES

Los robots móviles necesitan incorporar sistemas sensoriales que les permitan recolectar información del entorno, usualmente desconocido de antemano, con el fin de realizar de manera autónoma la tarea asignada. Estos sistemas sensoriales son esenciales para enfrentar desafíos de detección y reconocimiento de objetos que se presentan durante la navegación de los robots móviles [45].

Los sensores en robótica se pueden clasificar de diversas formas, siendo la más común la distinción entre sensores propioceptivos y exteroceptivos. Los sensores propioceptivos son aquellos que miden el estado interno del robot, como el desplazamiento, la orientación o el nivel de batería. Ejemplos típicos incluyen los *encoders*, utilizados en odometría para estimar la posición del robot respecto a un sistema de referencia inercial, así como giróscopos y acelerómetros, que proporcionan información sobre la orientación y aceleración. No obstante, estos sensores presentan el inconveniente de acumulación de errores con el tiempo, lo cual afecta a la precisión de las estimaciones. Por ejemplo, los *encoders* pueden generar errores debido al deslizamiento de las ruedas, mientras que los giróscopos y acelerómetros son susceptibles a desviaciones en las mediciones, las cuales se amplifican con el tiempo. Por esta razón, los sensores propioceptivos suelen combinarse con sensores exteroceptivos para corregir estos errores.

Los sensores exteroceptivos, en cambio, obtienen información del entorno que rodea al robot [46]. Entre los sensores más utilizados de esta categoría se encuentran el GPS, el sónar y el LIDAR. No obstante, dado que el GPS solo proporciona información de la ubicación del robot en tiempo preciso, en el presente trabajo se describirán únicamente el sónar y el LIDAR, que ofrecen información relacionada con el estudio que se está llevando a término.

El **sónar** (*Sound Navigation And Ranging*) es una tecnología que utiliza la propagación del sonido para medir distancias y detectar objetos, principalmente bajo el agua. Este sensor emite pulsos sonoros y mide el tiempo que tarda el eco en regresar tras reflejarse en un objeto, conocido como tiempo de vuelo (TOF, siglas en inglés *Time Of Flight*). Dado que la

velocidad del sonido en un medio es conocida, es posible calcular la distancia al objeto reflejado. Los sistemas de sónar se utilizan en aplicaciones donde el entorno puede ser difícil de evaluar mediante otros métodos, como en la navegación subacuática o en robots móviles que operan en entornos no estructurados.

Sin embargo, el sónar tiene limitaciones importantes, como la dependencia de las propiedades del material y la orientación de la superficie del objeto para dar reflexión al sonido. Además, los sensores de sónar pueden interferir entre sí si operan en frecuencias similares, lo que puede causar lecturas erróneas. También presentan una baja resolución angular y velocidad de escaneo comparadas con tecnologías más avanzadas como el láser.

Karimanzira et al. [47] indican que el sónar se emplea eficazmente para la detección de objetos en entornos submarinos debido a su independencia de la turbidez del agua. Sin embargo, el uso del sónar conlleva desafíos, como la resolución no homogénea, la intensidad no uniforme, el ruido moteado, las sombras acústicas y las reverberaciones. Además, las imágenes acústicas son proyecciones horizontales en 2D del entorno, lo que provoca ambigüedad en la localización de objetos a diferentes alturas, dificultando la precisión en su detección y calificación.

Shao et al. [48] señalan que las imágenes obtenidas mediante sonar suelen tener características de distorsión severa, bajo contraste y desenfoque. Esto dificulta la precisión y velocidad en la detección de objetos basada en características manuales. Para mejorar esta situación, proponen un método que utiliza la red YOLOv8 mejorada, incorporando convoluciones deformables para reducir el impacto de la distorsión en las imágenes sónar.

El **LIDAR** (acrónimo del inglés, *Light Detection And Ranging*) es una tecnología de teledetección que utiliza la emisión de un haz láser para medir distancias con alta precisión. Funciona enviando pulsos de luz láser hacia un objetivo y midiendo el tiempo que tarda la luz reflejada en regresar al sensor. Al igual que el sónar, este tiempo de vuelo permite calcular la distancia al objeto. Sin embargo, el LIDAR ofrece una resolución angular mucho mayor y una velocidad de escaneo más rápida que el sónar, lo que lo convierte en una herramienta esencial para la detección de obstáculos, el mapeo 3D y la captura de movimiento.

El LIDAR se utiliza extensamente en aplicaciones de navegación autónoma, incluyendo vehículos autónomos y drones, debido a su capacidad para generar mapas precisos del entorno en tiempo real. Sin embargo, presenta algunas desventajas, como su alto costo y la elevada demanda de procesamiento computacional para manejar los datos generados. Además, ciertos materiales, como el vidrio, pueden causar problemas de reflexión incorrecta, lo que introduce errores en la medición [45].

Muhammad y Kim [49] manifiestan que el sensor LIDAR es una herramienta clave para la percepción ambiental en entornos dinámicos, como las áreas urbanas complejas. Sin embargo, el LIDAR presenta el inconveniente de un gran número de detecciones de objetos inexactas. Para mitigar este problema, proponen un enfoque basado en redes neuronales para la detección visual, complementando el procesamiento de las nubes de puntos LIDAR. Indican que este método mejora la precisión en la detección y clasificación de objetos, reduciendo la carga computacional en el proceso de agrupamiento de datos.

McCrae y Zakhor [50] expresan que el LIDAR es fundamental para la detección precisa de objetos en la conducción autónoma, debido a su capacidad para capturar nubes de puntos tridimensionales del entorno. Sin embargo, mencionan que un desafío clave en el uso del LIDAR es la necesidad de procesar múltiples cuadros temporales, lo que puede generar problemas como la distorsión de objetos en movimiento. Para mitigar estos problemas, modificaron la arquitectura PointPillars, introduciendo una red recurrente que aprovecha datos temporales del LIDAR, mejorando la detección de peatones en un 8%, aunque con una ligera disminución en la detección de vehículos. La arquitectura PointPIllars convierte las nubes de puntos 3D capturadas por el LIDAR en pseudoimágenes 2D, permitiendo la detección eficiente de objetos en 3D mediante redes neuronales convolucionales.

2.3.1 SENSORES DE VISIÓN

Otro tipo de sensor exteroceptivo es el de visión. Los sensores de visión son dispositivos que capturan imágenes del entorno y las procesan para extraer información valiosa. Estos sensores son especialmente útiles para tareas como la extracción de características, la estimación de posición y el reconocimiento de patrones. A diferencia de otros tipos de sensores, las cámaras utilizadas en los sistemas de visión ofrecen una gran cantidad de información detallada sobre el entorno, lo que las hace fundamentales para aplicaciones avanzadas de robótica.

Una cámara funciona capturando la luz que se refleja en los objetos y proyectándola sobre una superficie sensible a la luz, como un sensor CMOS o CCD. La información visual obtenida se procesa posteriormente para permitir que el robot interprete y reaccione a su entorno. Las cámaras son económicas, ligeras y tienen un bajo consumo de energía, lo que las convierte en una solución atractiva para robots móviles autónomos.

Dependiendo del número de cámaras y el campo de visión necesario, los sistemas de visión pueden configurarse de diferentes formas:

Monoculares: Utilizan una sola cámara, lo que permite capturar imágenes desde un único punto de vista. Son la configuración más sencilla y económica, pero tienen limitaciones en la percepción de profundidad y la interpretación espacial [51].

Binoculares (par estéreo) [52, 53] y **Trinoculares** [54]: Estas configuraciones emplean dos o tres cámaras para proporcionar visión estéreo, lo que permite al sistema calcular la profundidad y la distancia de los objetos en el entorno. Aunque estas configuraciones de múltiples imágenes pueden incrementar el coste computacional.

Omnidireccionales: Este tipo de cámaras pueden estar compuestas por varias cámaras apuntando hacia distintas direcciones o compuestas por una única cámara y una superficie reflectante [55].

2.3.1.1 OMNIDIRECCIONALES

Las cámaras omnidireccionales son sistemas avanzados que ofrecen un campo de visión extremadamente amplio, permitiendo capturar una escena completa en una sola imagen. Son especialmente útiles en robótica móvil, ya que mantienen las características visuales estables incluso cuando el robot se mueve, proporcionando una descripción más completa y continua del entorno [56, 57].

Las cámaras omnidireccionales pueden clasificarse en centrales y no centrales dependiendo de si los rayos ópticos reflejados convergen en un único punto (central) o no lo hacen (no central). Este factor es crucial para el modelado y procesamiento de las imágenes obtenidas, afectando directamente a la complejidad de los algoritmos de visión por computadora [55].

2.3.1.1.1 Sistemas de visión catadióptricos

Los sistemas de visión catadióptricos combinan cámaras estándar con espejos convexos para capturar un amplio campo de visión. Un ejemplo común es el uso de un espejo hiperbólico junto con una cámara con lente de proyección ortográfica, que permite obtener imágenes panorámicas. Estos sistemas son versátiles y se utilizan en aplicaciones donde se requiere una visión 360 grados sin la necesidad de múltiples cámaras. En la Figura 2- 1 se muestra un sistema de visión catadióptrico, así como una imagen proporcionada por este tipo de sistema.

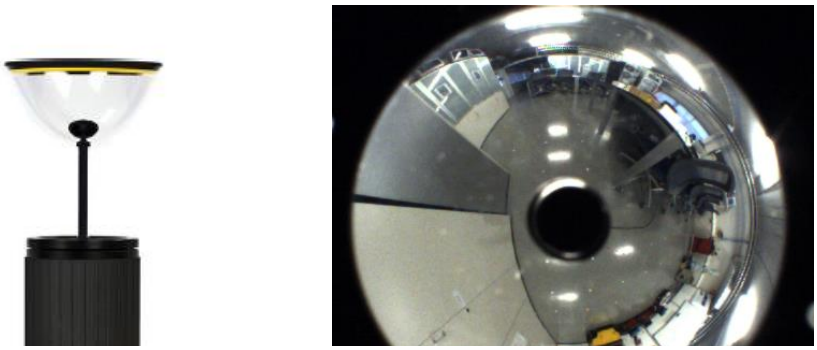


Figura 2- 1. Sistema catadióptrico omnidireccional e imagen que genera (sacada de la base de datos Omni DB *dataset* 2 – Laboratory Trajectory disponible en [58]).

2.3.1.1.2 Múltiples cámaras

Otra configuración para obtener una visión omnidireccional es la combinación de varias cámaras orientadas en diferentes direcciones de modo que sus campos de visión se superpongan. Un ejemplo es la cámara Ladybug (ver Figura 2- 2), que utiliza seis cámaras para capturar aproximadamente el 90 % del campo de visión del entorno. Otras cámaras comerciales con este tipo de configuración son la cámara Facebook Surround 360 (Figura 2- 3) y la Insta360 Titan (Figura 2- 4). Este tipo de sistema es ideal para aplicaciones que requieren una alta precisión en la estimación de trayectorias, como en vehículos autónomos y robots móviles avanzados [59].

Figura 2- 2. Cámara Ladybug6.

Figura 2- 3. Cámara Facebook surround 360.

Figura 2- 4. Cámara Insta360 Titan.

2.3.1.1.3 Cámaras con lente de ojo de pez

Las cámaras equipadas con lentes de ojo de pez, o *fisheye lenses*, son dispositivos ópticos diseñados para capturar un campo de visión (FOV, *Field Of View*) extremadamente amplio, generalmente entre 180 y 360 grados. Estas lentes permiten obtener una vista panorámica completa o casi completa, caracterizada por una distorsión notable que se asemeja a la visión de un ojo de pez. Esta distorsión es una de las características distintivas de las lentes ojo de pez permitiendo capturar una amplia área en una sola imagen.

Existen dos principales tipos de lentes ojo de pez. Las lentes ojo de pez circulares capturan una imagen en forma de círculo que llena el marco de la cámara, proporcionando una vista completa con un campo de visión de 180 grados en todas las direcciones. Este tipo de lente es especialmente útil para fotografía panorámica y efectos visuales que requieren una visión envolvente y completa. En contraste, las lentes de ojo de pez rectilíneas ofrecen un campo de visión de hasta 180 grados en un formato rectangular, corrigiendo parte de la distorsión inherente de las lentes ojo de pez. Estas lentes son ideales para aplicaciones donde se necesita un campo de visión amplio, pero con menor distorsión, como en la vigilancia y en la creación de panorámicas rectilíneas [60].

Además de estos tipos de lentes, existen cámaras de ojo de pez compuestas por dos lentes que permiten capturar imágenes en 360 grados. Cada lente se orienta en direcciones opuestas, abarcando la mitad del campo de visión. Posteriormente, las imágenes capturadas por ambas lentes se combinan para formar una vista esférica completa. Estas cámaras son particularmente útiles en aplicaciones de realidad virtual y vigilancia, ya que proporcionan una visión total del entorno sin dejar zonas ciegas. Al integrar dos lentes, estas cámaras logran cubrir un área de captura más extensa, facilitando una visualización inmersiva y panorámica. Un ejemplo de cámara compuesta por dos lentes ojo de pez es la Garmin VIRB 360 (Figura 2- 5).

Una de las principales ventajas de las cámaras con lentes ojo de pez es su capacidad para proporcionar un campo de visión amplio. Esto les permite capturar grandes áreas en una sola imagen, es decir, pueden adquirir una gran cantidad de información visual. Gracias a esta característica, presentan ventajas significativas en la recolección de datos visuales [61]. Por esta razón, son ampliamente utilizadas en aplicaciones de vigilancia inteligente [62], drones [63, 64], realidad virtual [65] [66, 67, 68, 69], robótica [70, 71, 72, 73], etc.

Sin embargo, estas lentes también tienen desventajas notables. La distorsión significativa que producen puede ser problemática en aplicaciones que requieren una representación precisa y sin curvaturas. Además, la calidad de la imagen puede verse reducida, especialmente en los bordes de la imagen, debido a la curvatura de la lente. Esta limitación puede ser un desafío en contextos profesionales donde se requiere alta precisión y resolución.

Degang Yang et al. [74] indican que las cámaras de ojo de pez satisfacen de manera eficaz la necesidad de un ángulo de visión amplio en las imágenes capturadas. Sin embargo, esto hace que las imágenes tengan distorsiones significativas, tamaños y distancias relativos exagerados de los objetos y produce la aparición de discontinuidad en los bordes de los objetos.

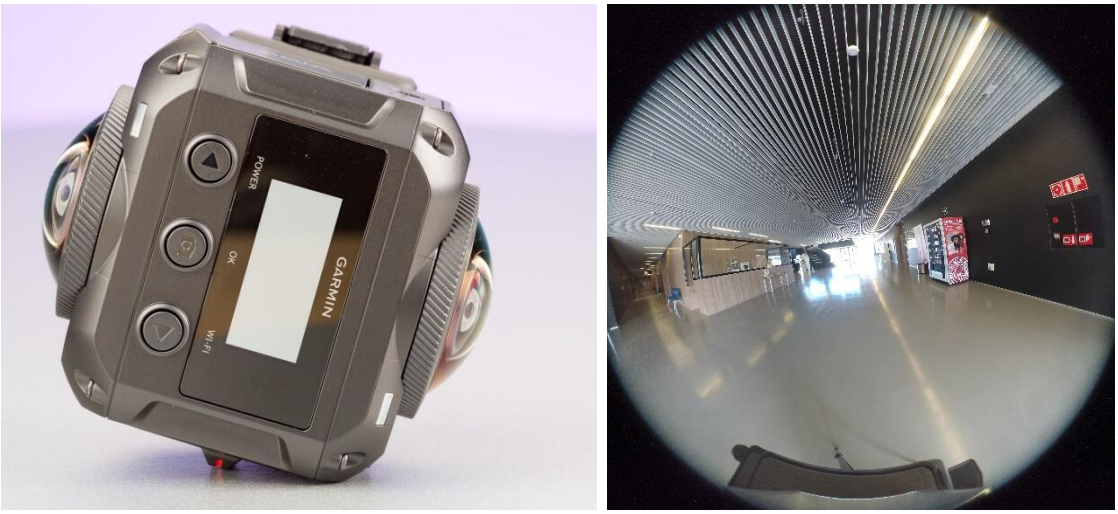


Figura 2- 5. Cámara de ojo de pez Garmin VIRB 360, junto con una de las imágenes ojo de pez generada por una de sus lentes (delantera o trasera).

2.4 MÉTODOS PARA LA DETECCIÓN Y RECONOCIMIENTO DE OBJETOS

La detección y reconocimiento de objetos son componentes cruciales en la visión por computadora y tienen una amplia gama de aplicaciones, desde la seguridad y la vigilancia [75, 76, 77] hasta la conducción autónoma [78, 70, 67, 68] y la realidad aumentada [66].

La detección de objetos en visión artificial enfrenta desafíos significativos en términos de precisión y eficiencia. Entre los problemas principales se encuentra la variación intraclase, que incluye diferencias en color, forma, tamaño, textura y pose dentro de una misma categoría de objetos, complicando su identificación precisa [81]. La detección multiescala es otro reto, ya que los objetos pueden variar mucho en tamaño, haciendo que los objetos pequeños y densos sean difíciles de detectar. Además, la detección en tiempo real es crucial para aplicaciones como los vehículos autónomos, donde se debe equilibrar la velocidad, precisión y recursos computacionales. La detección débilmente supervisada presenta dificultades cuando los modelos solo tienen etiquetas de imagen sin cuadros delimitadores, lo cual es problemático para detectar objetos raros o ausentes en los datos de entrenamiento. El desequilibrio de las muestras de entrenamiento también afecta el rendimiento de los modelos, requiriendo mejoras en el muestreo y ajustes en las funciones de perdida [79]. Finalmente, el manejo de un gran número de categorías de objetos demanda una gran cantidad de datos anotados y de alta calidad, lo que es un desafío considerable y limita la capacidad de los modelos para distinguir entre clases con diferencias sutiles [80]. Estos problemas reflejan la complejidad y los requerimientos elevados en la detección de objetos, a pesar de los avances en técnicas como las redes neuronales profundas.

Afif et al. [82] exponen que el reconocimiento de objetos en interiores presenta diversos desafíos debido a la complejidad y variabilidad de estos entornos, que suelen incluir una amplia gama de elementos y decoraciones.

La detección y reconocimiento de objetos son tareas fundamentales en el campo de la visión por ordenador. La detección se centra en identificar y localizar objetos dentro de una imagen o video, mientras que el reconocimiento va un paso más allá, no solo detectando la presencia de los objetos, sino también identificándolos y categorizándolos correctamente. A lo largo del tiempo, se han desarrollado diversos enfoques para abordar estas tareas, desde métodos tradicionales basados en características locales hasta técnicas más avanzadas como las redes neuronales convolucionales. Cada enfoque presenta sus propias características, ventajas y desventajas, permitiendo a las máquinas aprender y reconocer patrones visuales complejos de manera cada vez más eficiente.

Las Redes Neuronales Convolucionales (CNNs, siglas en inglés *Convolutional Neural Networks*) son cruciales para el reconocimiento de objetos debido a su capacidad para aprender características espaciales y jerárquicas a partir de datos de imagen. Modelos como AlexNet, VGG, GoogLeNet y ResNet han marcado un hito en términos de precisión y velocidad en tareas de reconocimiento

[83, 84, 85, 86]. AlexNet, introducido por Krzhevsky et al. en 2012 [83], revolucionó el campo con su arquitectura profunda y el uso de GPUs para entrenamiento. VGG, presentado por Simonyan y Zisserman [84], mejoró la precisión mediante el uso de redes más profundas con filtros convolucionales pequeños. GoogLeNet introdujo la *Inception Module*, que permite capturar múltiples escalas de información con una sola red, mientras que ResNet, con sus bloques de identidad, facilitó el entrenamiento de redes extremadamente profundas y mejoró significativamente la precisión [85, 86].

Los modelos de aprendizaje profundo se pueden dividir en dos partes principales: modelos basados en regiones y métodos de una sola etapa [82].

Yang et al. [74] clasifican los métodos de detección de objetos basados en aprendizaje profundo en dos categorías principales: algoritmos de dos etapas y de una sola etapa. Los algoritmos de detección de objetos de dos etapas generan regiones candidatas y luego realizan la clasificación sobre ellas. Como ejemplos de estos métodos incluyen R-CNN, Fast R-CNN y Mask R-CNN, que ofrecen alta precisión, pero con una velocidad limitada. Por otro lado, los algoritmos de una sola etapa, como SSD y YOLO, tratan la detección como un proceso directo, logrando una alta velocidad y precisión comparable a los métodos de dos etapas.

2.4.1 MÉTODOS BASADOS EN REGIONES (*Region-Based Methods)*

Los métodos que pertenecen a esta categoría se caracterizan por realizar el proceso en dos etapas. En la primera, dada una imagen de entrada, se obtienen una serie de *bounding boxes* que son aquellas regiones de la imagen en las que es posible que haya un objeto. Después, mediante una red neuronal profunda, se extraen características de cada una de estas regiones propuestas. En la segunda etapa, se utiliza una red de clasificación para, a partir de las características extraídas, predecir cuál es la clase del objeto en cada región, así como, refinar la posición del *bounding box* propuesto en la primera etapa. A continuación, se describen algunos de los algoritmos propuestos en la literatura que se encuentran dentro de esta categoría.

R-CNN (*Region-Based Convolutional Neural Network*): Este método, propuesto por Girshick et al. en 2014 [87], genera una serie de propuestas de regiones en una imagen que podrían contener objetos, utilizando un algoritmo de selección de regiones como *Selective Search*. Luego, se utiliza una red neuronal convolucional (CNN, *Convolutional Neural Network*) para extraer características de estas regiones propuestas, y una capa de clasificación adicional clasifica cada región como una categoría específica. Aunque este enfoque logra mejoras significativas en la precisión de la detección, el procesamiento es relativamente lento debido a la necesidad de aplicar la CNN a cada región propuesta de forma independiente.

Fast R-CNN: Girshick [88] propuso este método en 2015 con el propósito de mejorar el tiempo de procesamiento al utilizar una sola CNN para extraer características de toda la imagen en lugar de hacerlo para cada región propuesta

individualmente. Después de extraer las características, se aplican técnicas de agrupamiento de características (*Rol Pooling*) para obtener características fijas de cada región propuesta, que luego se usan para la clasificación y la regresión de cajas delimitadoras. Esta optimización reduce significativamente el tiempo de procesamiento en comparación con R-CNN, manteniendo una alta precisión.

Faster R-CNN: Presentado por Ren et al. en 2015 [89], Faster R-CNN introduce una Red de Propuesta de Regiones (RPN, *Region Proposal Network*) que genera regiones propuestas de manera más eficiente y con menos cómputo que el método de *Selective Search* utilizado en R-CNN y Fast R-CNN. La RPN es una red completamente convolucional que predice cajas delimitadoras y puntajes de objeto para cada posición en una imagen integrándose de manera eficiente con la red de detección. Esto mejora aún más la velocidad y precisión de la detección, combinando la propuesta de regiones y la clasificación en un único modelo *end-to-end*.

Mask R-CNN es una extensión de Faster R-CNN que no solo detecta objetos, sino que también realiza segmentación de instancias. Introduce una cabeza adicional en la red para predecir una máscara de segmentación para cada objeto detectado, lo que permite generar máscaras precisas y detalladas que delimitan cada objeto en la imagen. Esta capacidad de segmentación de instancias permite una detección más precisa y útil para tareas que requieren información detallada sobre los contornos de los objetos [90].

2.4.2 MÉTODOS DE DETECCIÓN EN UNA SOLA ETAPA (*Single-Stage Detection*)

Con el objetivo de conseguir un mejor rendimiento en tiempo real (menor coste computacional) y un diseño más simple con respecto a la categoría anterior, se han propuesto métodos en los que se realiza la detección en una única etapa. Por ello, los métodos que se encuentran dentro de esta categoría se caracterizan por emplear una única red neuronal a toda la imagen para determinar la posición de los *bounding boxes* y las clases de los objetos.

YOLO (*You Only Look Once*): Redmon et al. [91] proponen este método de detección en una sola etapa que divide la imagen en una cuadrícula y hace predicciones de clases y cajas delimitadoras directamente desde esta cuadrícula en una sola pasada. Este enfoque simplifica el proceso de detección al eliminar la necesidad de una etapa de propuesta de regiones, logrando una detección en tiempo real con una alta velocidad de procesamiento. YOLO ha pasado por varias versiones, cada una mejorando la precisión y velocidad. Una descripción más detallada de estas versiones y sus mejoras se presenta en el apartado 3.2 de este trabajo.

SSD (*Single Shot MultiBox Detector*): Liu et al. en 2016 [92], presentan otro método que se caracteriza por predecir las cajas delimitadoras y las probabilidades de clase en una sola pasada a través de la red. SSD utiliza una serie de capas convolucionales en diferentes escalas para manejar objetos de diferentes tamaños, y las predicciones se generan a partir de múltiples capas de

características. Este enfoque permite una detección rápida y eficiente, combinando precisión con velocidad.

RetinaNet: Lin et al. en 2017 [93], sugieren utilizar un enfoque basado en anclas y presenta una nueva función de pérdida llamada *Focal Loss* para abordar el desequilibrio de clases en la detección de objetos. En este método, una red de detección predice cajas delimitadoras y clasificaciones para múltiples anclas distribuidas a lo largo de la imagen. La *Focal Loss* ayuda a enfocar el entrenamiento en ejemplos difíciles y subrepresentados, mejorando la precisión en la detección.

Afif et al. [94] proponen un sistema de detección de objetos en interiores que utiliza como base la arquitectura de RetinaNet y Feature Pyramid Network (Red de Pirámide de Características) para mejorar la precisión en la detección de objetos de diferentes tamaños (pequeños, medianos y grandes). Además, la función de pérdida focal se centra en mejorar el rendimiento de muestras más complejas. Logrando así una detección más precisa, eficiente y adaptable en ambientes interiores.

CAPÍTULO 3. DETALLES DE LA IMPLEMENTACIÓN

En la actualidad, las redes neuronales convolucionales (CNN, por sus siglas en inglés) se destacan como una de las técnicas más utilizadas en el ámbito del aprendizaje profundo debido a sus notables resultados en diversas aplicaciones prácticas. Este tipo de red neuronal está especialmente diseñado para procesar datos con una estructura topológica predefinida, siendo su aplicación más común el análisis de imágenes, donde se emplean para tareas como clasificación y la detección de objetos.

Las CNN se caracterizan por conexiones locales entre neuronas y una organización jerárquica en la transformación de los datos. Estas redes están compuestas principalmente por tres tipos de capas: convolucionales (conv), de agrupación (*pooling*) y completamente conectadas -*fully connected* (fc)-. Cada capa transforma la entrada y produce una salida según los parámetros previamente definidos. Este proceso de conexión implica la transformación de información a través de todas las capas, finalizando en la última, que es una capa completamente conectada y que genera un vector de características unidimensional, el cual proporciona la predicción más probable.

No obstante, la construcción y el entrenamiento de una red desde cero requiere tanto experiencia con arquitecturas de redes como una gran cantidad de datos para el entrenamiento y, por tanto, un tiempo de computación importante. En este sentido, se propone utilizar una arquitectura de red muy popular para la tarea de detección de objetos, como es la red neuronal YOLO de la que se puede encontrar bastante información sobre ella en internet.

YOLO (*You Only Loo Once*) es una red neuronal convolucional desarrollada, originalmente por Joseph Redmon y su equipo, para la detección de objetos en tiempo real. A lo largo de sus diferentes versiones, YOLO ha evolucionado significativamente, incorporando mejoras en su arquitectura y rendimiento, lo que la ha convertido en una de las opciones más populares para tareas de visión por computadora. Treven et al. [95] presentan un análisis exhaustivo de la evolución de YOLO desde la versión inicial hasta YOLOv8, YOLO-NAS y YOLO con transformadores.

3.1 ARQUITECTURA DE YOLO

La arquitectura de YOLO es distinta de otros enfoques de detección de objetos como R-CNN. En lugar de generar propuestas de regiones y luego clasificar cada una, YOLO aplica una única red neuronal sobre la imagen completa, dividiendo la imagen en una cuadrícula *S x S*. Cada celda de la cuadrícula predice un número fijo de "*bounding boxes*" (rectángulos que delimitan los objetos) junto con sus probabilidades de clase. En YOLOv1, la imagen se dividía en una cuadrícula de 7x7, y cada celda predecía 2 *bounding boxes*, resultando en 98 *boxes* para la imagen completa. Cada *bounding box* tenía 5 parámetros: *x, y* (coordenadas del centro de la caja relativas a la celda), ancho, alto y una confianza que indicaba la probabilidad de que la caja contuviera un objeto y la precisión de la caja con

respecto a dicho objeto. Además, cada celda predecía las probabilidades para las distintas clases [91].

3.2 EVOLUCIÓN Y MEJORAS EN LAS VERSIONES DE YOLO

Desde su primera versión, YOLO ha experimentado una serie de mejoras significativas:

YOLOv2 (YOLO9000) introdujo la normalización por lotes (*Batch Normalization)* para estabilizar y acelerar el entrenamiento, además de "*anchor boxes*", que son tamaños de cajas predefinidos que mejoran la detección de objetos de diferentes tamaños y formas. También se incluyó una reducción de dimensiones mediante el uso de una capa convolucional que reduce las dimensiones espaciales antes de la salida final, permitiendo la predicción a múltiples escalas [96].

YOLOv3 mejoró la detección a múltiples escalas mediante *Feature Pyramid Networks* (FPN), agregando salidas en diferentes capas de la red para capturar mejor los objetos de diferentes tamaños. Además, se rediseñaron las capas de salida para incluir tres capas separadas, que permiten la detección de objetos pequeños, medianos y grandes [97].

YOLOv4 introdujo *Cross Stage Partial Networks* (CSPNet) para reducir la cantidad de parámetros y aumentar la eficiencia computacional sin perder precisión. Además, se incorporó *Path Aggregation Network* (PANet) para mejorar la información de la pirámide de características y *Mosaic Augmentation*, que aumenta la diversidad del conjunto de entrenamiento al combinar cuatro imágenes de manera aleatoria en una sola imagen de entrenamiento. También se mejoró la función de pérdida utilizando CIOU *Loss* (*Complete Intersection Over Union Loss*), que considera el área, el solapamiento y la distancia entre los centros de las cajas [98].

YOLOv5, aunque no fue desarrollado por los autores originales, se ha popularizado por su facilidad de uso y eficiencia. Introdujo técnicas como escalado automático del modelo (*Automated Model Scaling*) para ajustar la arquitectura en función del tamaño del modelo, mejoras en la optimización del entrenamiento y despliegue, y técnicas avanzadas de aumentación de datos como CutMix y Mosaic [99, 100].

YOLOv6 y YOLOv7 son modelos que continuaron con mejoras en la eficiencia y precisión utilizando técnicas avanzadas de arquitectura y regularización [101, 102, 95, 103].

Zhou et al. [61] indican que la arquitectura de YOLOv7 se caracteriza por su eficiencia, combinando velocidad, precisión y robustez sin incrementar los costes de inferencia. Adicionalmente, optimiza el uso de recursos al reducir los parámetros y la complejidad computacional. Por estos motivos la utilizan como base de su modelo e introducen un módulo de convolución deformable modulado y el módulo de transformación Swin para mejorar la capacidad de extracción de

características, lo que se traduce en una mejora significativa en la detección de objetos en comparación con la versión original.

YOLOv8, versión lanzada por Ultralytics en el año 2023, tiene un pipeline modular, optimización para diferentes plataformas y soporte para tareas adicionales como segmentación y clasificación [104, 101].

Yang et. al. [74] en su trabajo también optan por utilizar la arquitectura de YOLO empleando el modelo YOLOv8s para mejorar la detección de objetos en imágenes de ojo de pez. Este tipo de imágenes presentan características únicas como la distorsión y la discontinuidad. El método propuesto se basa en la implementación del módulo C2f, que mejora la precisión del modelo. Además, reemplazan el método *Anchor-Base* por *Anchor-Free*, lo que permite corregir las posiciones de los objetos de manera más precisa al predecir la distancia que tienen desde el centro hasta el cuadro delimitador.

En la Figura 3- 1se presenta una línea de tiempo que ilustra la evolución de las versiones de YOLO desde el año 2015, con el lanzamiento de YOLOv1, hasta el 2023, con la versión más reciente, YOLOv8. La figura destaca los hitos más relevantes en el desarrollo de este modelo, incluyendo versiones intermedias como YOLOv2 (YOLO900), YOLOv3 y YOLOv4, así como las variantes derivadas, como YOLOX y PP-YOLOv2, que han contribuido a la expansión y mejora del rendimiento de los modelos YOLO en términos de precisión y eficiencia. Esta visualización facilita la comprensión de cómo cada iteración ha incorporado avances técnicos para optimizar la detección de objetos.

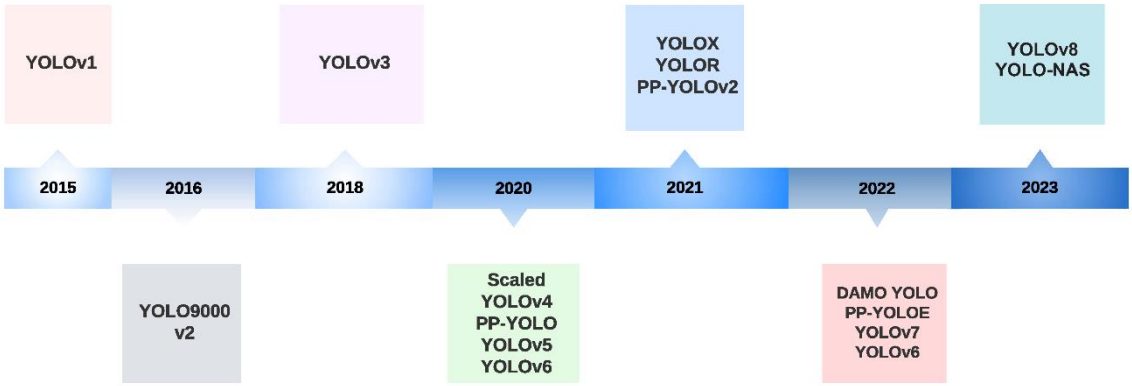


Figura 3- 1. Línea de tiempo de las versiones YOLO.

3.3 PARÁMETROS DEFINIDOS POR DEFECTO

En todas las versiones de YOLO, existen ciertos parámetros clave que suelen estar definidos por defecto. Entre estos se encuentra la tasa de aprendizaje (*Learning Rate*), que controla la magnitud de las actualizaciones en los pesos del modelo durante el entrenamiento; *Momentum y Weight Decay*, que son parámetros que ayudan en la convergencia del modelo; el tamaño de la cuadrícula (S x S), que varía según la versión y la resolución de la imagen de entrada, el número de clases,

que depende del conjunto de datos utilizado y los "*anchor boxes*", que se pueden ajustar durante el entrenamiento utilizando *K-means clustering* para los datos específicos del problema. Otros parámetros importantes incluyen el umbral de Intersección sobre Unión (IoU, siglas del inglés *Intersection over Union*), que define cuando una predicción se considera correcta en relación con la caja real, y el umbral de confianza que determina si la predicción se considera válida [101].

La versión de YOLOv8, lanzada por Ultralytics, representa la culminación de todas las mejoras anteriores, incorporando lo último en técnicas de redes neuronales. YOLOv8 es una versión más eficiente y flexible, diseñada para ser una solución *end-to-end* optimizada para diversas aplicaciones, desde servidores hasta dispositivos móviles.

La arquitectura de YOLOv8 incluye un *backbone* y una cabeza optimizada que mejoran la detección de objetos pequeños y difíciles. También se ha integrado soporte mejorado para despliegue en diferentes plataformas, aprovechando aceleraciones harware como TensorRT y CoreML. Además, YOLOv8 presenta un pipeline de entrenamiento más eficiente, que incluye optimización de hiperparámetros y mejoras en la precisión de las predicciones. Una de las características destacadas de YOLOv8 es un diseño modular, que permite un fácil ajuste de diferentes partes del modelo para tareas específicas, así como un soporte para diferentes tareas, como segmentación y clasificación, con modificaciones mínimas.

En términos de parámetros, YOLOv8 permite un ajuste flexible del tamaño de la cuadrícula, adaptándose mejor a las características específicas del conjunto de datos. Los *"anchor boxes"* se ajustan automáticamente durante el entrenamiento para optimizar la detección de objetos de diferentes tamaños, y se utilizan técnicas avanzadas de regularización como *batch normalization* y *dropout* para evitar el sobreajuste [101].

3.4 CREACIÓN DEL *DATASET*

El primer paso, fundamental para la creación de un *dataset* de imágenes, es reunir las imágenes que conformarán el conjunto de datos. Estas imágenes pueden provenir de diversas fuentes, incluyendo bases de datos públicas o colecciones privadas.

Aunque existen varias bases de datos conocidas, como KITTI u Oxford RobotCar, que son utilizadas en trabajos de investigación similares [74] no se ha encontrado una base de datos compuesta por imágenes ojo de pez y que hayan sido capturadas en un entorno interior, más concretamente un edificio público como una universidad, que es el tipo de entornos de interés para el presente trabajo.

Por este motivo y para cumplir con el objetivo de detectar objetos en imágenes distorsionadas, es indispensable capturar un conjunto de escenas utilizando una cámara con lentes ojo de pez. Dado que, este tipo de lente introduce una distorsión particular que permite simular escenarios donde los objetos aparecen deformados, reproduciendo así las condiciones que se desean estudiar

en entornos reales de navegación autónoma en interiores. La recopilación de imágenes bajo esta configuración óptica no solo amplía el campo de visión y facilita la captura de mayores detalles de la escena en una sola toma, sino que además proporciona al modelo YOLOv8 un conjunto de datos que contiene los retos únicos de la distorsión, esenciales para el entrenamiento y evaluación de su precisión en estas condiciones específicas.

3.4.1 SISTEMA DE VISIÓN UTILIZADO

En el presente trabajo como sistema de visión para la captura de imágenes, se emplea la cámara Garmin VIRB 360 [105] (Figura 3- 2). Este dispositivo cuenta con dos lentes tipo *fisheye* y dos sensores CMOS retroiluminados de 1/2.3". Gracias a sus lentes, cada una con un campo de visión (FOV, *Field of View*) de 201.8º, y a su disposición (con las lentes colocadas de manera opuesta), la Garmin VIRB 360 ofrece un campo de visión completo de 360 grados, abarcando una esfera completa. Las especificaciones técnicas [106] más relevantes de la cámara se detallan en la Tabla 3- 1. Además, la cámara permite seleccionar entre cuatro modos ópticos: 360º, solo delantera, solo trasera y RAW. La Tabla 3- 2 proporciona una descripción de cada modo, mientras que la Figura 3- 3 ilustra los tipos de imagen producidos por cada configuración.

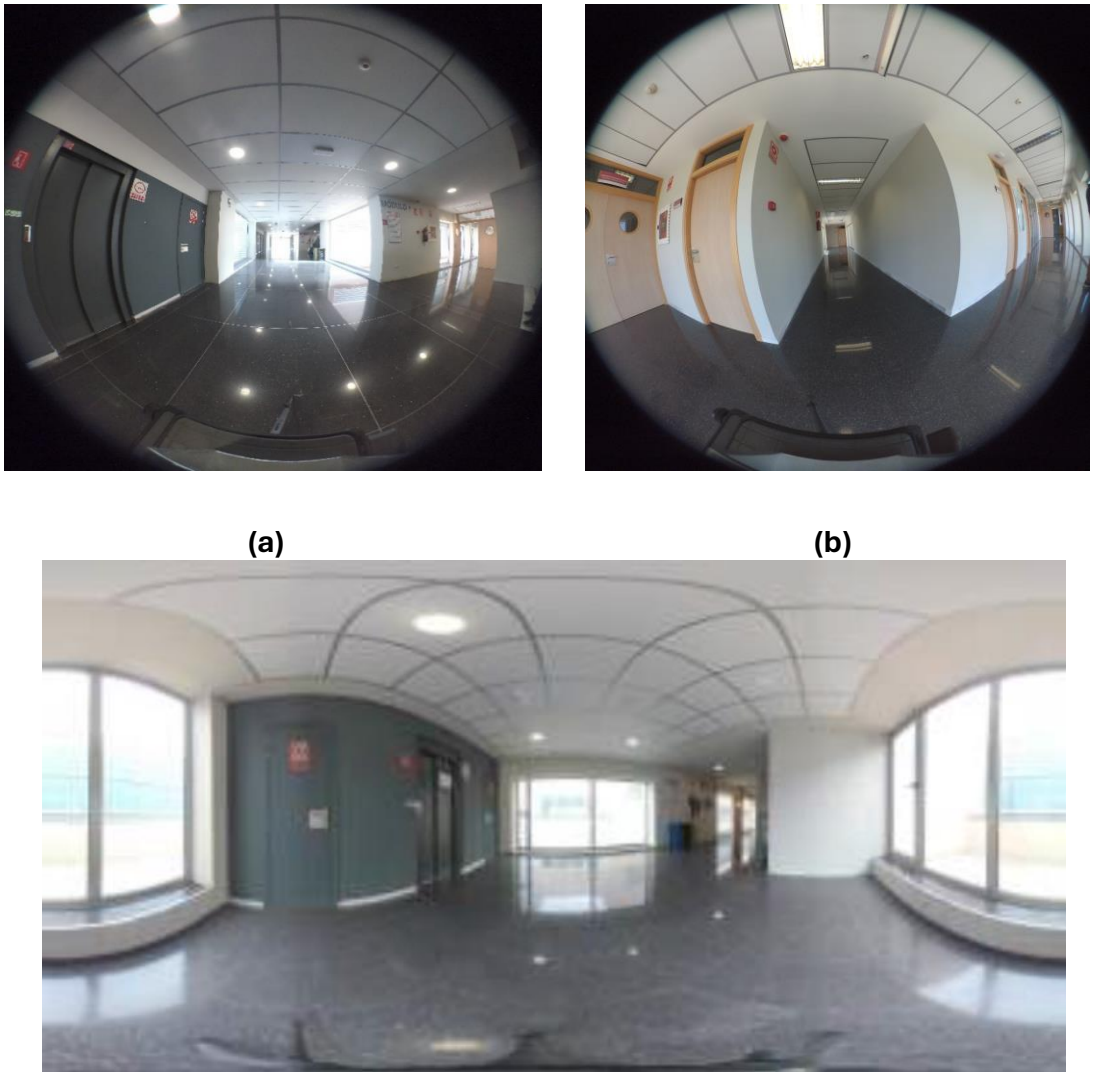
Figura 3- 2. Cámara Garmin VIRB 360.

Tabla 3- 1. Especificaciones técnicas principales cámara Garmin VIRB 360

Características físicas	
Peso:	160 g (con batería)
Tamaño (altura x anchura x profundidad):	39.0 x 59.3 x 69.8 mm
Características ópticas	
Sensor:	1/2.3” CMOS con iluminación trasera (2 sensores)
Número de lentes:	2
Distancia focal efectiva:	1.036 mm
Campo de visión (FOV):	201.8 grados (cada lente)

Tabla 3- 2. Configuraciones ópticas disponibles en la cámara Garmin VIRB 360 para fotografías y vídeos.

Modo	Descripción	Resolución fotografía
360	Con este modo se obtiene una imagen esférica completa en formato equirectangular. En la Figura 3- 3 (c) se puede ver un ejemplo de imagen obtenida configurando la cámara con este modo.	5640 x 2820 (15MP)
Solo delantera:	Este modo produce una imagen capturada por la lente trasera. En la Figura 3- 3 (d) se puede ver un ejemplo de imagen obtenida configurando la cámara con este modo.	1920 x 1440 (3MP)
Solo trasera:	Utilizando este modo, se genera una imagen en perspectiva a partir de la lente trasera. En la Figura 3- 3 (e) se puede ver un ejemplo de imagen obtenida configurando la cámara con este modo.	1920 x 1440 (3MP)
RAW:	Este modo toma una imagen sin procesar con cada lente, generando 2 archivos. En la Figura 3- 3 (a) y la Figura 3- 3 (b) se puede ver un ejemplo de imagen obtenida configurando la cámara con este modo.	3008 x 3000 (2 x 9MP)



(a) **(b)**

(c)

(d) **(e)**

Figura 3- 3. Tipos de imágenes capturadas por la Garmin VIRB 360 según el modo configurado (ver Tabla 3- 2). Se muestran: par de imágenes *fisheye*, **(a)** una delantera y otra **(b)** trasera, obtenidas simultáneamente en el modo RAW; **(c)** vista panorámica completa producida con el modo 360; **(d)** imagen en perspectiva tomada con la cámara delantera; e **(e)** imagen en perspectiva obtenida con la cámara trasera.

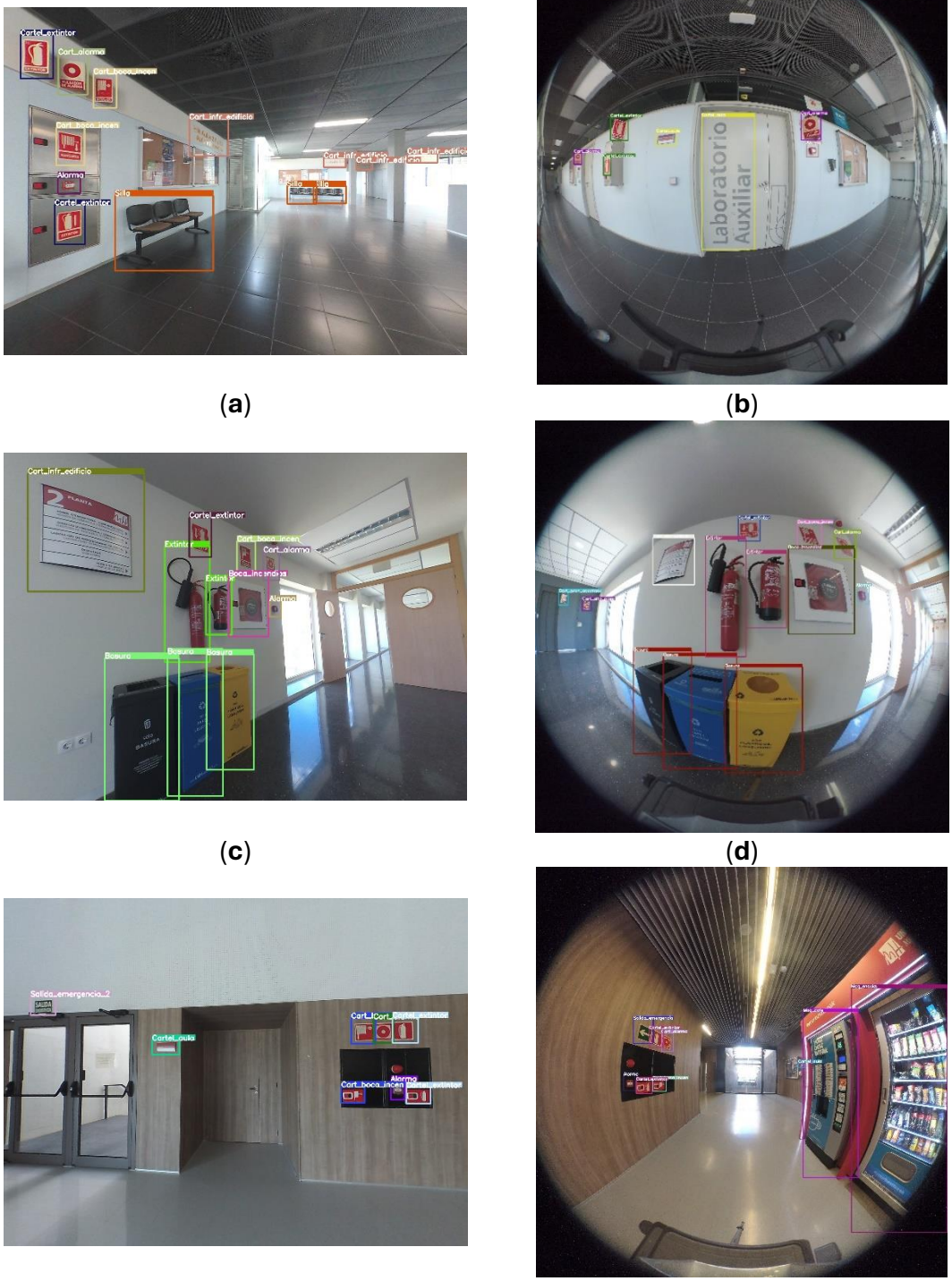
3.4.2 FORMA DE CAPTACIÓN IMÁGENES

En el presente trabajo las imágenes utilizadas fueron tomadas desde distintos puntos de vista y condiciones de iluminación con el objetivo de recrear diferentes situaciones. La cámara se montó en un trípode para así captar las imágenes desde una perspectiva cercana al suelo, y no desde una perspectiva superior (desde el techo), dado que este trabajo está enfocado a que la cámara esté dispuesta en un robot con movimiento terrestre. Se pretende que, mediante la identificación de los diferentes carteles y elementos característicos del interior del edificio, el robot sea capaz de reconocer los elementos representativos de los entornos interiores en los que se moverá. Aunque este estudio se centra exclusivamente en la detección de estos objetos, los resultados podrán servir de base para futuras tareas de *mapping* y localización, que no serán abordadas en el presente trabajo.

Cabe destacar que, aunque el presente trabajo se centra en estudiar la detección de objetos en imágenes con distorsión, se han capturado tanto imágenes de este tipo como sin distorsión. Las imágenes capturadas se pueden dividir en dos grupos: imágenes sin distorsión, que son las que se denominan normales en el presente proyecto, y las imágenes con distorsión, que son las capturadas en formato ojo de pez. Para obtener las imágenes sin distorsión, la cámara se configuró en modo solo delantera o solo trasera, generando imágenes de 1920 x 1440 (como se muestra en la Figura 3- 3 (d) y (e)). En el caso de las imágenes con distorsión, la cámara se configuró en modo RAW, capturando dos imágenes ojo de pez de 3008 x 3000 (ver Figura 3- 3 (a) y (b)), una con cada lente.

Después de la captación de las imágenes, el siguiente paso consiste en anotar las imágenes identificando y marcando con cuadros delimitadores rectangulares los objetos que se desean detectar.

En la actualidad existen diferentes programas, por ejemplo, robotflow, y páginas webs, por ejemplo, Make Sense, que permiten realizar el etiquetado de los objetos que aparecen en las imágenes y que se desean detectar. En este trabajo se optó por utilizar el programa Label-Studio que es una herramienta de etiquetado de datos de código abierto que admite múltiples proyectos, usuarios y tipos de datos en una sola plataforma. Además, permite exportar los datos etiquetados que se incluirán en las carpetas *images* y *labels* del *dataset*, y que contendrán las imágenes e información de ubicación de los objetos, la categoría de los objetos y las coordenadas superior izquierda e inferior derecha de los cuadros delimitadores rectangulares de los objetos, etc. respectivamente. En la Figura 3- 4 se muestran algunas imágenes sin distorsión (en la primera columna) y con distorsión (en la segunda columna) de la base de datos capturada tras el etiquetado de los objetos de interés del presente estudio. Por otro lado, en la Figura 3- 5 se presenta la interfaz de Label-Studio utilizada para el etiquetado de objetos, en la cual se visualizan distintos elementos, como señales de extintores, emergencia y boca de incendios, delimitados por cuadros de colores asignados a cada categoría de objeto. Esta herramienta facilita la anotación precisa y visual de los elementos de interés en las imágenes.



(**a**) (**b**)

(**c**) (**d**)

(**e**) (**f**)

Figura 3- 4. Imágenes del interior de los edificios de la Universidad Miguel Hernández mostrando las etiquetas de los objetos junto con sus *bounding boxes*. (**a**) Imagen sin distorsión interior edificio Altet UMH, (**b**) Imagen con distorsión interior edificio Altet UMH, (**c**) Imagen sin distorsión interior edificio Innova, (**d**) Imagen con distorsión interior edificio Innova, (**e**) Imagen sin distorsión interior edificio Arenals, (**f**) Imagen con distorsión interior edificio Arenals.

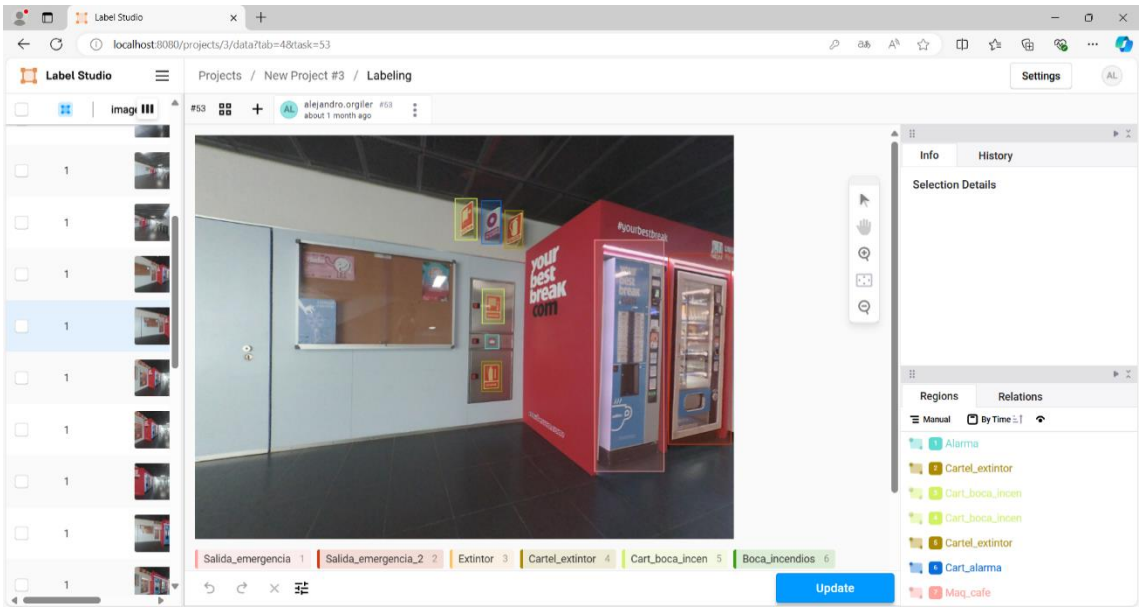


Figura 3- 5. La herramienta Label-Studio para etiquetado de objetos. En una imagen el cuadro de color es el cuadro delimitador para el etiquetado de cada objeto.

Una vez se tienen etiquetadas las imágenes, estas anotaciones se deben exportar a un formato compatible con la red neuronal que se utilice, en nuestro caso en formato YOLO (*.txt) compatible con la herramienta Ultralytics [101].

Posteriormente, el conjunto de datos debe de organizarse de acuerdo con una estructura de carpetas específica. En el directorio raíz del conjunto de datos, se deben crear dos subdirectorios principales: 'images/' y 'labels/'. Dentro de cada uno de estos subdirectorios, se deben incluir dos subcarpetas adicionales denominadas 'train/' (entrenamiento) y 'val/' (validación), quedando la estructura de la siguiente manera:

dataset/

├— images/

│ ├— train/

│ └— val/

└— labels/

├— train/

└— val/

Finalmente, en el directorio raíz del conjunto de datos, es crucial crear un archivo denominado 'data2.yaml'. Este archivo debe contener una descripción detallada del conjunto de datos, incluyendo las clases y cualquier otra información

necesaria para su uso. El 'data2.yaml' es esencial para garantizar que los datos se interpreten y utilicen correctamente durante el entrenamiento y evaluación de los modelos.

Es importante destacar que, si las anotaciones de las imágenes se han realizado en distintos *datasets* y se desea unificarlas en uno solo, las clases deben tener la misma estructura en todos los casos. Esto significa que las categorías y etiquetas utilizadas para anotar las imágenes deben ser consistentes en todos los *datasets*. De no ser así, se podrían presentar problemas con la identificación y el manejo de las clases durante el proceso de entrenamiento, lo cual afectaría negativamente el rendimiento y la precisión del modelo.

El conjunto de datos de este trabajo se compone de un total de 1008 imágenes capturadas en interior, que presentan diferentes condiciones de iluminación, suponiendo este hecho que el conjunto de datos sea más robusto. En el conjunto de datos se pueden encontrar imágenes con distorsión y sin distorsión, por tanto, el total de imágenes se puede dividir en dos carpetas. La carpeta denominada data_normal contiene un total de 456 imágenes capturadas en interior sin distorsión, y la carpeta llamada data_ojo_pez está compuesta por un total de 552 imágenes con distorsión. De las 456 imágenes sin distorsión, 198 fueron tomadas en el edificio Innova, 94 en el edificio Arenals y 164 en el edificio Altet. En cuanto a las imágenes con distorsión, 222 fueron tomadas en el edificio Altet, 104 en el edificio Arenals y 226 en el edificio Innova. Los edificios nombrados pertenecen a la Universidad Miguel Hernández de Elche. Todas las imágenes están en formato .jpg.

El conjunto de datos contiene 33 objetos emblemáticos a detectar que suelen estar presentes en entornos interiores (ver Tabla 3- 3 y Figura 3- 6).

Tabla 3- 3. Objetos que se desean detectar.

1. Alarma	12. Cartel información edificio	23. Máquina de refrescos
2. Basura	13. Cartel prohibido ascensor	24. Máquina de snacks
3. Boca incendios	14. Cartel publicaciones	25. Mesa
4. Botellas agua	15. Cartel aula	26. Papelera
5. Cartel prohibido el paso	16. Cartel extintor	27. Persona
6. Cartel ascensor	17. Escaleras	28. Planta decorativa
7. Cartel Alarma	18. Extintor	29. Salida de emergencia
8. Cartel alto el paso	19. Fuente de agua	30. Salida de emergencia 2
9. Cartel boca incendios	20. Impresora	31. Servicio
10. Cartel despacho	21. Máquina de agua	32. Silla
11. Cartel escaleras	22. Máquina de café	33. Videovigilancia

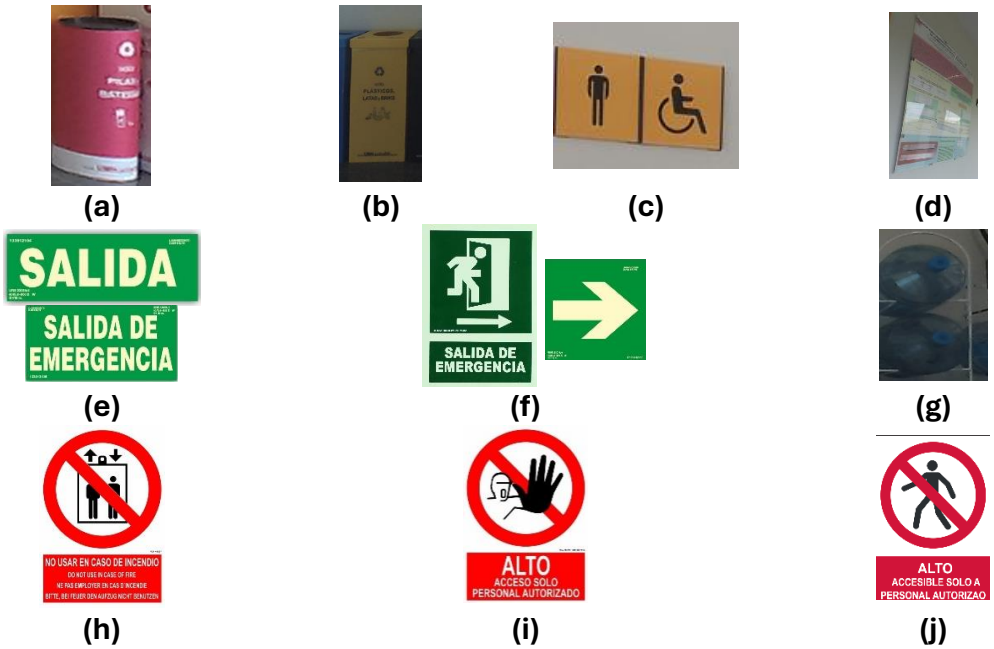


Figura 3- 6. Aclaración visual de ciertos objetos. **(a)** papelera, **(b)** basura, **(c)** servicio, **(d)** cartel publicaciones, **(e)** salida emergencia 2, **(f)** salida emergencia, **(g)** botellas de agua, **(h)** cartel prohibido ascensor, **(i)** cartel alto el paso, **(j)** cartel prohibido el paso.

La configuración utilizada para el entrenamiento, validación y prueba después de efectuar la anotación del conjunto de datos es la siguiente. Para el conjunto de datos de prueba, se reservaron un total de 7 imágenes con distorsión y un total de 7 imágenes sin distorsión, de manera aleatoria, y el resto se utilizó para incluirlo en la carpeta creada en el directorio raíz denominada *dataset*. En este directorio la distribución de imágenes y *labels* dentro de sus subcarpetas, *train* y val, se ha realizado de forma aleatoria, usando el 80 % de las imágenes y sus *labels* correspondientes en las carpetas 'train/'. Y un 20 % de las imágenes y sus respectivos *labels* se han incluido en las carpetas denominadas 'val/', para que sean usados estos datos en el proceso de validación. Esta es la mejor forma de distribuir las imágenes, para permitir que la red neuronal aprenda de forma más rápida y correcta.

Por consiguiente, el conjunto de datos denominado data_normal está organizado en dos subcarpetas dentro del directorio *images*: la subcarpeta 'train/', que contiene 360 imágenes, y la subcarpeta 'val/', que incluye 89 imágenes. Las subcarpetas correspondientes dentro del directorio *labels* ('train/' y 'val/') contienen las etiquetas asociadas a estas imágenes, distribuidas de manera que haya una correspondencia exacta entre cada imagen y su respectiva etiqueta. De manera análoga, en el conjunto de datos data_ojo_pez, la subcarpeta 'train/' contiene 436 imágenes y la subcarpeta 'val/' 109 imágenes, con sus correspondientes etiquetas distribuidas de forma coherente en las subcarpetas del directorio *labels*. Finalmente, el conjunto de datos denominado data_total es el resultado de la suma de ambos conjuntos de datos, integrando todas las imágenes y etiquetas previamente mencionadas.

CAPÍTULO 4. EXPERIMENTOS Y RESULTADOS

El objetivo de este capítulo es evaluar la eficacia de la red neuronal YOLO en la detección de objetos en entornos interiores. Para lograrlo, se han llevado a cabo una serie de experimentos, cuyos detalles y resultados se presentan a continuación.

4.1 SELECCIÓN DE MODELO YOLOv8

Lo primero de todo es seleccionar un modelo YOLOv8 dado que como se menciona en [107] el modelo cuenta con una amplia variedad de versiones, cada una optimizada para abordar tareas específicas de visión por ordenador. Estos modelos están diseñados para adaptarse a diversas necesidades, abarcando desde la detección de objetos hasta las tareas más avanzadas, como la segmentación de instancias, la detección de pose y puntos clave, la identificación de la orientación de los objetos y la clasificación.

Dado que el presente trabajo se centra en la detección de objetos en interiores, se procede a evaluar cuál de los cinco modelos disponibles en la serie YOLOv8 (ver Tabla 4- 1) ofrece un mejor rendimiento en este contexto. Para ello, se realizan pruebas tanto en imágenes convencionales como en imágenes capturadas en formato de ojo de pez, por la cámara Garmin VIRB, con el objetivo de determinar qué modelo es capaz de detectar los objetos deseados en las imágenes sin haber entrenado la red.

Tabla 4- 1. Modelos serie YOLOv8 para detección.

MODELOS YOLOv8		
Nano	n	yolov8n.pt
Small	s	yolov8s.pt
Medium	m	yolov8m.pt
Large	l	yolov8l.pt
XLarge	x	yolov8x.pt

El proceso de determinación del modelo se ha realizado utilizando el entorno de desarrollo PyCharm, empleando el código descrito en el ANEXO I y tres imágenes de ojo de pez capturadas en el interior de los edificios. Este código permite visualizar, al final de su ejecución, tanto en terminal como en una imagen guardada los resultados de los objetos detectados por la red neuronal, siendo una herramienta efectiva para evaluar los resultados obtenidos por cada modelo. Los resultados de esta prueba fueron los siguientes:

Al emplear el modelo nano, se obtuvo como resultado que en dos de las imágenes no se detectó ningún objeto (Figura 4- 1 a) y c)). Sin embargo, en una de las imágenes se detectó correctamente una persona, como se muestra en la Figura 4- 1 b).

En el experimento utilizando el modelo small, no se detectó ningún objeto en ninguna de las imágenes analizadas, como se muestra en la Figura 4- 1 d), e) y f).

En las pruebas realizadas con el modelo medium, se obtuvieron detecciones de objetos en las tres imágenes analizadas. En la imagen de la Figura 4- 2 g), se detectaron dos objetos: una silla y un objeto identificado como remoto; sin embargo, ninguno de ellos corresponde con los elementos reales presentes en la imagen. El objeto detectado como silla es en realidad el soporte donde está ubicada la cámara, y el objeto identificado como remoto corresponde a una luz de emergencia.

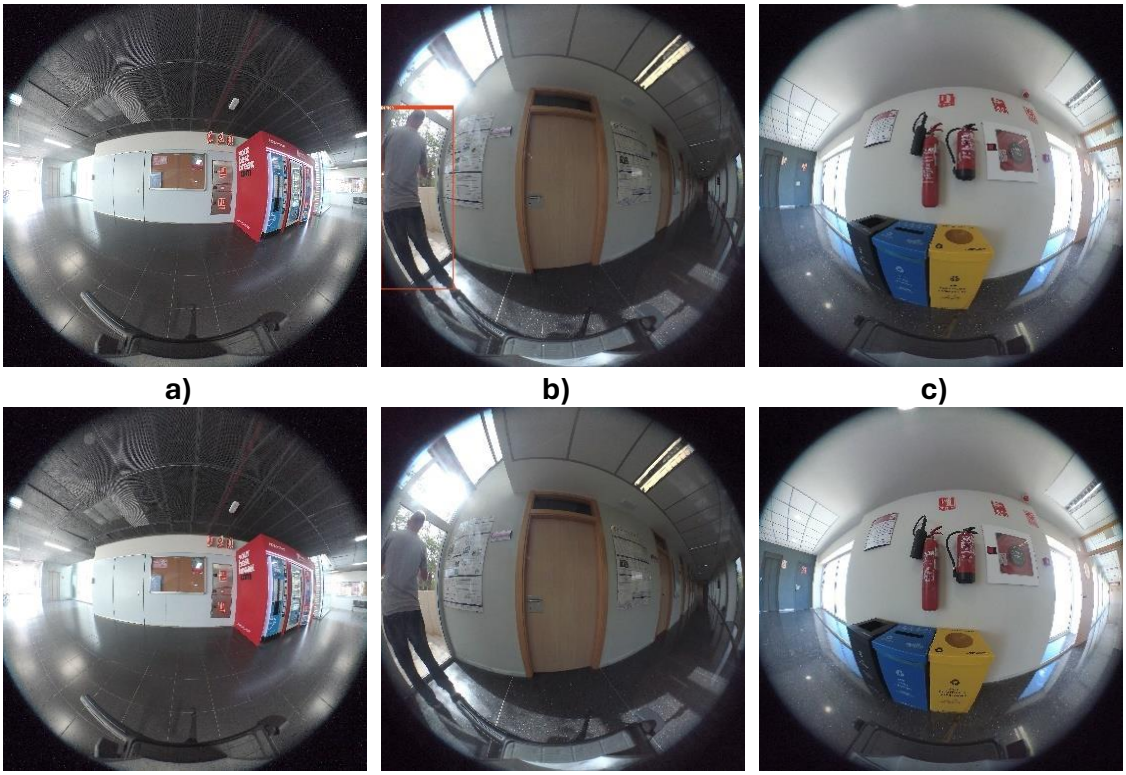
En la Figura 4- 2 h), el modelo detecta correctamente a la persona presente en la imagen, pero los objetos identificados como libro y baño son en realidad un cartel de despacho y la puerta de este, respectivamente.

La Figura 4- 2 i) muestra los objetos detectados en la imagen, donde el modelo identifica una silla, un microondas, un horno, un frigorífico y un libro. No obstante, estas identificaciones son incorrectas. El objeto identificado como silla es en realidad el soporte del trípode que sostiene la cámara, mientras que el supuesto microondas corresponde a tres basuras de reciclaje, una de las cuales también se identifica erróneamente como un horno. Además, el objeto identificado como libro es en realidad un cartel informativo del edificio.

Por último, el objeto detectado como frigorífico es en realidad una composición de varios elementos, incluyendo las basuras de reciclaje, extintores, una boca de incendios, una alarma y los carteles correspondientes a estos últimos, junto con el cartel informativo del edificio.

Al utilizar el modelo large, se observó que en una de las imágenes no se detectó ningún objeto (Figura 4- 2 j)). En la imagen de la Figura 4- 2 k), se detectó correctamente a una persona, sin identificar ningún otro elemento adicional. En la tercera imagen (Figura 4- 2 l)), se detectó únicamente un objeto, aunque este no es coherente con lo que realmente se observa. El modelo identificó una silla, cuando en realidad no hay ninguna en la imagen; el objeto detectado corresponde a la base del trípode en el que se encuentra la cámara fotográfica.

Por último, el análisis del modelo Xlarge muestra que la detección de una persona en la imagen de la Figura 4- 2 n) es correcta, aunque no se identificaron otros objetos. No obstante, en la Figura 4- 2 ñ), el modelo detecta un único objeto de manera incorrecta, ya que lo que identificado como silla corresponde en realidad a la base del trípode de la cámara. De igual forma, en la Figura 4- 2 m), los dos únicos elementos detectados, identificados como televisión y copa, son incorrectos, puesto que se trata de un cartel y una luz de emergencia.



a) **b)** **c)**

d) **e)** **f)**

Figura 4- 1. Imágenes obtenidas al aplicar los diferentes modelos de YOLOV8 para detección. Imágenes **a)**, **b)** y **c)** sacadas tras aplicar el modelo yolov8n.pt; imágenes **d)**, **e)** y **f)** obtenidas después de aplicar el modelo yolov8s.pt.

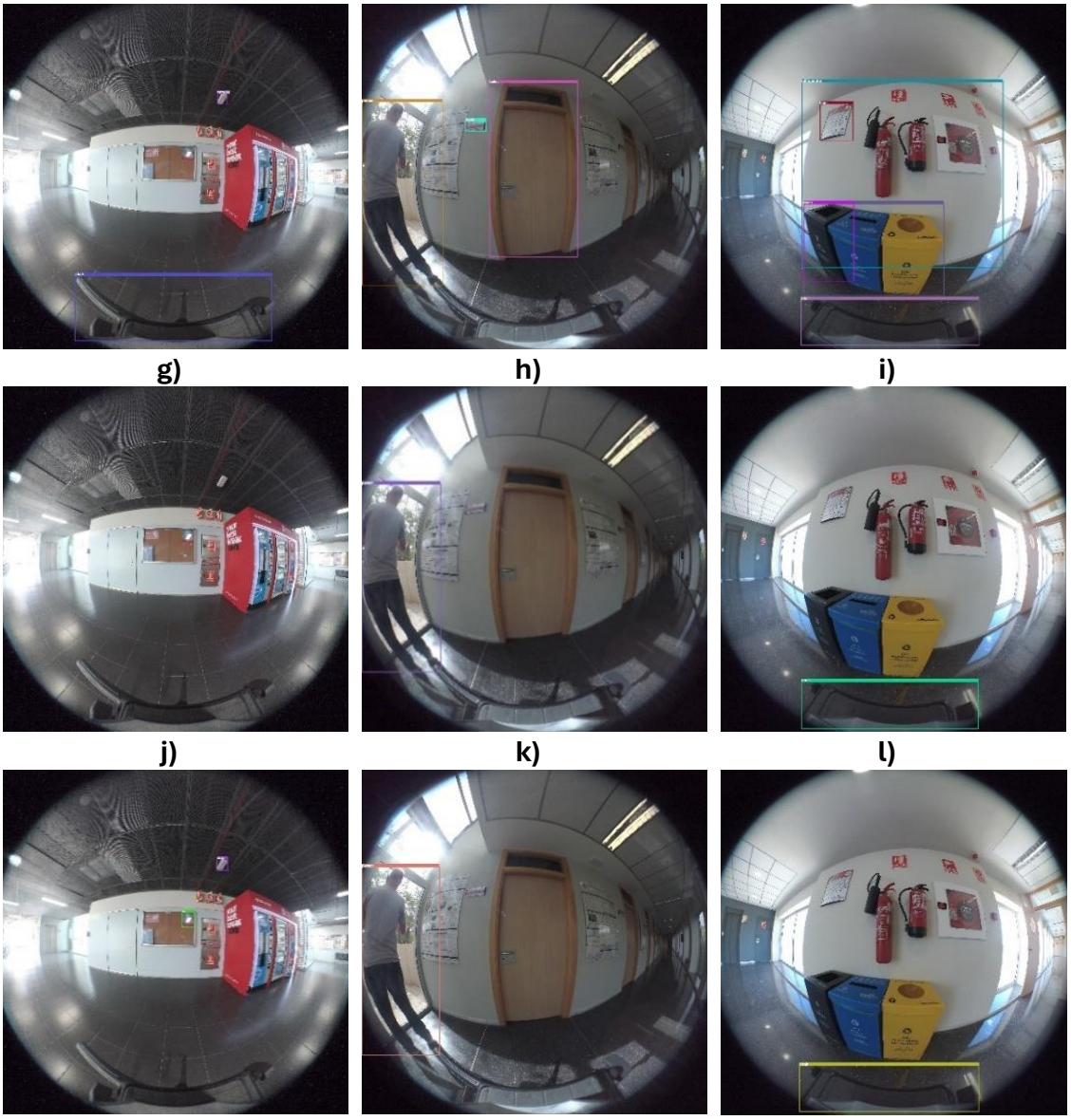


Figura 4- 2. Imágenes obtenidas al aplicar los diferentes modelos de YOLOV8 para detección. Imágenes **g)**, **h)** y **i)** obtenidas una vez aplicado el modelo yolov8m.pt; imágenes **j)**, **k)** y **l)** de salida al aplicar el modelo yolov8l.pt; imágenes **m)**, **n)** y **ñ)** sacadas de aplicar el modelo yolov8x.pt.

En base a las pruebas realizadas, los resultados obtenidos y considerando los diferentes aspectos mencionados en la web de Ultralytics, se ha decidido utilizar el modelo nano (yolov8n.pt). Aunque no es el modelo que detecta la mayor cantidad de objetos, pero hay que tener en cuenta que YOLO no ha sido entrenada para que detecte los objetos presentes, salvo personas. Así pues, considerando ambos formatos de imagen, es el que ofrece una mayor precisión en los resultados.

4.2 ASPECTOS DE ULTRALYTICS A TENER EN CONSIDERACIÓN

Una vez seleccionado el modelo que se va a utilizar en los entrenamientos, se procede a detallar los aspectos más relevantes de la librería Ultralytics que se deben de tener en consideración.

Es fundamental conocer que Ultralytics YOLOv8 dispone de varios modos para obtener un rendimiento maximizado del modelo:

- Modo entrenamiento: permite ajustar el modelo utilizando conjuntos de datos personalizados o preexistentes.
- Modo de validación (val): sirve como punto de control posterior al entrenamiento para evaluar el rendimiento del modelo.
- Modo de exportación: prepara el modelo para su despliegue en distintos formatos.
- Modo de seguimiento: extiende la funcionalidad del modelo de detección de objetos a aplicaciones de seguimiento en tiempo real.
- Modo *benchmark*: permite analizar la velocidad y precisión del modelo en diferentes entornos de despliegue [108].

En el presente trabajo, se ha utilizado el modo de entrenamiento para realizar los experimentos. La validación de los resultados se hace usando el método expuesto en apartado anterior y analizando los datos que ofrece el entrenamiento.

El modo entrenamiento (*train*) se emplea para entrenar el modelo YOLOv8 con un conjunto de datos personalizado. El modelo, en este modo, se optimiza utilizando los datos y los hiperparámetros especificados, con el fin de predecir con precisión las clases y ubicaciones de los objetos en una imagen. Entre las características más destacadas del modo de entrenamiento se encuentran la descarga automática de conjuntos de datos estándar, el soporte para entrenamiento con múltiples GPUs (*Graphics Processing Units*) para acelerar el proceso, la posibilidad de configuración de hiperparámetros y la visualización y monitorización en tiempo real, lo que permite un seguimiento continuo del entrenamiento y una comprensión detallada del proceso de aprendizaje.

El código empleado en PyCharm para la ejecución del entrenamiento es el que se muestra a continuación. Este código está basado en lo dispuesto en [100] y ha sido reformulado teniendo en cuenta los parámetros que se variarán en los

diferentes entrenamientos realizados, con el fin de analizar su impacto en el entrenamiento del modelo:

```
from ultralytics import YOLO
import numpy as np
import cv2
import os
import matplotlib as plt
from random import randint

def train():

model=YOLO("yolov8n.pt")
results_train = model.train(data="data2.yaml", device=0, epochs=300, batch=8, imgsz=1920)
```

Es importante señalar que el modelo de YOLOv8 que se emplea es previamente llamado y que previamente a la ejecución del código se debe de cargar la librería Ultralytics en PyCharm. Por lo que, se recomienda instalar la librería con el siguiente comando en la terminal de PyCharm:

```
pip install ultralytics
```

Además, se han tenido en consideración los ajustes de entrenamiento que permite Ultralytics, según los intereses de nuestro estudio. En el ANEXO II están contenidos todos los argumentos que se pueden imponer, como vienen definidos por defecto y la descripción de cada uno de ellos [101].

Uno de los aspectos más importantes de la red YOLOv8 es la posibilidad de ajustar los parámetros que permiten realizar un aumento de datos. Técnica fundamental para fortalecer la robustez y rendimiento del modelo, ya que añade variabilidad a los datos de entrenamiento, lo que facilita una mejor generalización de los datos no vistos. Además, resulta especialmente útil cuando el conjunto de datos disponible es insuficiente para entrenar el modelo de manera óptima y alcanzar los resultados deseados, dado que contar con un amplio conjunto de datos de entrenamiento es crucial para el rendimiento del modelo [109]. En el ANEXO III (información sacada de [101]) se describen cada uno de los argumentos que se pueden modificar y el impacto que tiene modificarlos.

De acuerdo con la información proporcionada por Ultralytics [100], YOLOv8 guarda automáticamente los resultados de los entrenamientos con TensorBoard y CSV en el directorio ***'runs/detect/train'***, generando un nuevo subdirectorio para cada sesión de entrenamiento, como ***'runs/detect/train2'***, ***'runs/detect/train3'***, etc.

Este subdirectorio incluye estadísticas de entrenamiento y validación, mosaicos, etiquetas, predicciones, mosaicos aumentados, así como métricas y

gráficos, entre los que se encuentran las curvas de precisión-recall (PR) y las matrices de confusión.

Además, al finalizar el entrenamiento se muestra en la terminal una validación de los datos contenidos en el archivo 'best.pt', que se encuentra dentro de una carpeta del subdirectorio de la sesión de entrenamiento llevada a cabo. Este archivo contiene información del nuevo modelo creado para cada una de las clases especificadas de los objetos suministrados. Los datos de validación que se exponen en la terminal son:

- Nombre de la clase del objeto.
- Número de imágenes de validación que contiene cada clase,
- Instancias, número de veces que aparece la clase en todas las imágenes del conjunto de validación.
- *Box* (P, R, mAP50, mAP50-90) que presenta los valores métricos del rendimiento del modelo en la detección de cada objeto. Significando P (*Precision*), R (*Recall*), mAP50 (precisión media calculada con un umbral de intersección sobre unión de 0.5) y mAP50-95 (precisión media calculada con distintos umbrales de intersección sobre unión que van de 0.5 a 0.95).

En la Figura 4- 3 se puede observar un ejemplo resumen de esta validación. La primera columna, *'Class'*, contiene el nombre de cada una de las clases de objetos detectados, como por ejemplo "Alarma", "Basura", "Botellas_agua", etc. La columna *'Images'* indica el número de imágenes de validación en las que aparece al menos un ejemplo de cada clase; por ejemplo, la clase "Alarma" figura en 14 imágenes. A continuación, la columna *'Instances'* muestra el número total de instancias en que aparece cada clase en todas las imágenes; por ejemplo, la clase "Alarma" se manifiesta 24 veces en total.

Los valores de rendimiento del modelo se encuentran en las columnas bajo el encabezado *'Box'*. Aquí se presentan métricas como:

- P (*Precision*): indica la fracción de predicciones correctas sobre todas las predicciones realizadas para una clase. Por ejemplo, la clase "Alarma" tiene una precisión de 0.892.
- R (*Recall*): refleja la capacidad del modelo de encontrar todas las instancias de una clase en las imágenes. Para la clase "Alarma", la recuperación es de 0.876.
- mAP50: este valor representa la precisión media utilizando un umbral de 0.5 en la intersección sobre unión (IoU), que para la clase "Alarma" es 0.907.
- mAP50-95: es la precisión media calculada utilizando diferentes umbrales de IoU entre 0.5 y 0.95, obteniendo para "Alarma" un valor de 0.516.

Al final del proceso, se indica que los resultados se guardaron en la ruta *'runs/detect/train92'*, con tiempos procesales promedio de 1.2 ms para preprocesamiento, 5.7 ms para inferencia, y 1.5 ms para posprocesamiento por imagen.

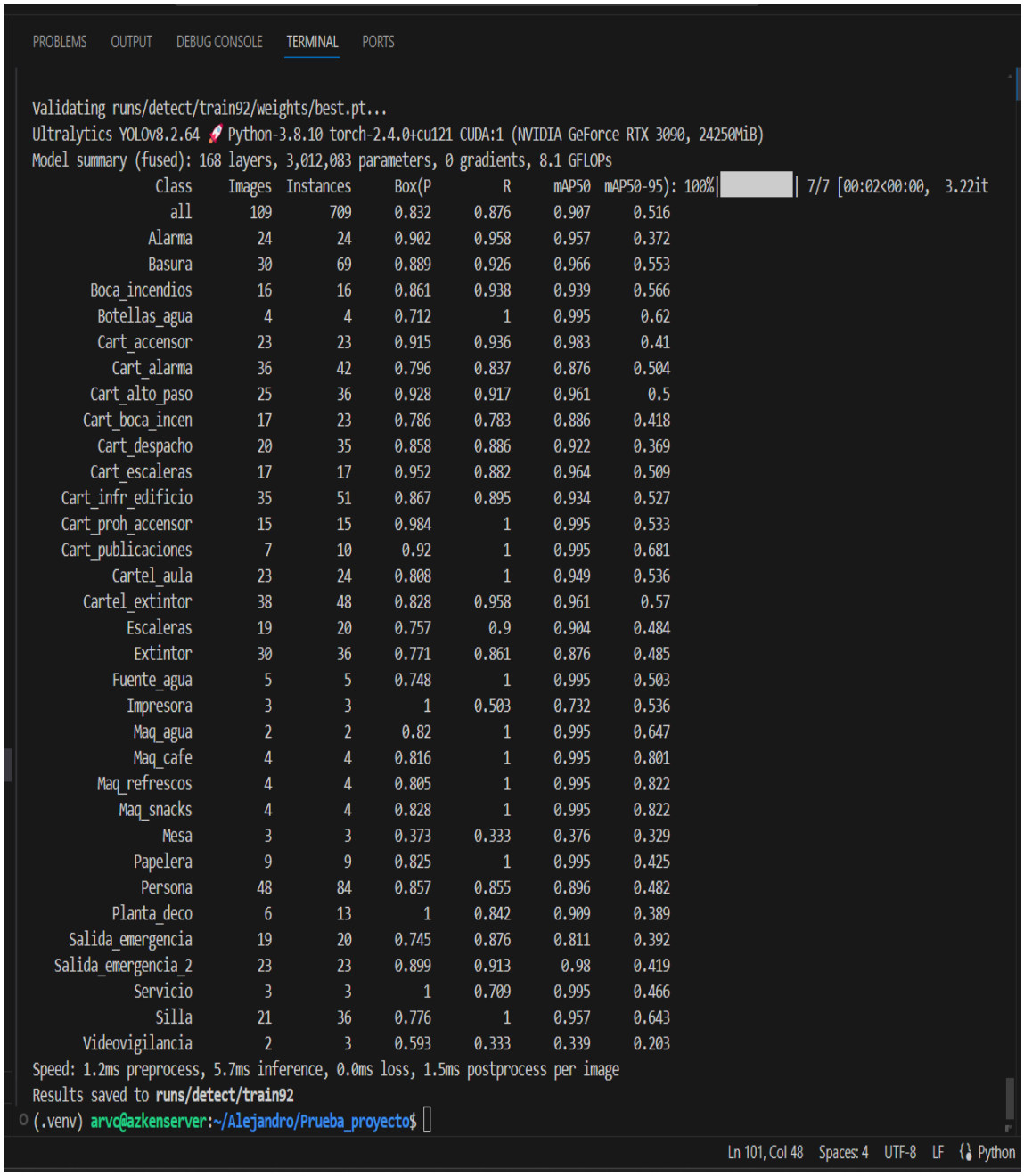


Figura 4- 3. Ejemplo de validación que se muestra en la terminal al final del entrenamiento.

4.3 METODOLOGÍA DE ENTRENAMIENTO

El proceso para entrenar la red YOLOv8 sigue el siguiente procedimiento:

1) Selección del modelo: Primero, se define el modelo a utilizar, el cual no se modifica y se utilizará en todos los experimentos.

2) Configuración del conjunto de datos: Luego, es necesario ajustar el archivo 'data2.yaml' para especificar las rutas de los archivos correspondientes al conjunto de datos de imágenes que se utilizarán tanto en el entrenamiento (*train*) como en la validación (val). Esto implica actualizar la ruta, si se usa un conjunto de datos diferente al experimento previo.

3) Ajuste de parámetros de entrenamiento: Finalmente, se configuran los parámetros de entrenamiento, incluyendo el número de *batch*, el tamaño de la imagen de entrada ('imgsz') y el dispositivo ('device') que se utilizará para llevar a cabo el entrenamiento. Es relevante señalar que YOLO incorpora, de manera predeterminada, un sistema de aumento de datos durante los entrenamientos, el cual no ha sido modificado en le presente estudio, manteniendo así la configuración estándar de este parámetro.

La Tabla 4- 2 resume las configuraciones de entrenamiento evaluadas, que varían en los parámetros de conjunto de datos, número de épocas, tamaño de *batch* y tamaño de imagen. Se han utilizado tres conjuntos de datos: data_normal, data_total y data_ojo_pez. El número de épocas utilizado en los entrenamientos fue de 150 o 300, y el tamaño de *batch* fue de 8 o 16. Las imágenes utilizadas tuvieron los tamaños de entrada de 640 x 640, 1280 x 1280, 1920 x 1920 y 2880 x 2880 píxeles.

En los entrenamientos con *batch* de 8, se realizaron pruebas con todos los tamaños de imagen. Sin embargo, en los entrenamientos con *batch* de 16, no fue posible utilizar imágenes de 2880 x 2880 píxeles debido a las limitaciones de memoria de la GPU (*Graphics Processing Unit*). Cabe mencionar que para estos experimentos se utilizó una NVIDIA GeForce RTX 3090, la cual, no pudo manejar el tamaño de imagen de entrada de 2880 píxeles con un *batch* de 16.

Tabla 4- 2. Configuraciones que se han estudiado que se diferencian por el tipo de *dataset* empleado (segunda columna), el número de épocas (tercera columna) y *batch* (cuarta columna), así como el tamaño de imagen de entrada (última columna) establecido.

Nº entrenamiento	***dataset***	***epochs***	***batch***	***Imgsz***
1	data_normal	300	8	640
2	data_normal	300	8	1280
3	data_normal	300	8	1920
4	data_normal	300	8	2880
5	data_normal	300	16	640
6	data_normal	300	16	1280
7	data_normal	300	16	1920
8	data_normal	150	8	640
9	data_normal	150	8	1280
10	data_normal	150	8	1920
11	data_normal	150	8	2880
12	data_normal	150	16	640
13	data_normal	150	16	1280
14	data_normal	150	16	1920
15	data_ojo_pez	300	8	640
16	data_ojo_pez	300	8	1280
17	data_ojo_pez	300	8	1920
18	data_ojo_pez	300	8	2880
19	data_ojo_pez	300	16	640
20	data_ojo_pez	300	16	1280
21	data_ojo_pez	300	16	1920
22	data_ojo_pez	150	8	640
23	data_ojo_pez	150	8	1280
24	data_ojo_pez	150	8	1920
25	data_ojo_pez	150	8	2880
26	data_ojo_pez	150	16	640
27	data_ojo_pez	150	16	1280
28	data_ojo_pez	150	16	1920
29	data_total	300	8	640
30	data_total	300	8	1280
31	data_total	300	8	1920
32	data_total	300	8	2880
33	data_total	300	16	640
34	data_total	300	16	1280
35	data_total	300	16	1920
36	data_total	150	8	640
37	data_total	150	8	1280
38	data_total	150	8	1920
39	data_total	150	8	2880
40	data_total	150	16	640
41	data_total	150	16	1280
42	data_total	150	16	1920

4.4 MÉTRICAS DE EVALUACIÓN

En este trabajo, se utilizan *F1 score* (puntuación F1), *Precision* (precisión), *Recall* (recuperación) y mAP@0.5 para evaluar el rendimiento de detección de los objetos de las imágenes contenidas en los conjuntos de datos: *data_normal, data_ojo_pez* y *data_total*. A continuación, se describen brevemente cada una de estas métricas y su relevancia en el contexto de la detección de objetos.

- *Precision* (precisión): La precisión mide la proporción de verdaderos positivos (objetos correctamente identificados) entre todas las predicciones positivas realizadas por el modelo. Es una métrica clave cuando el objetivo es minimizar los falsos positivos, es decir, cuando es crucial que las detecciones realizadas sean correctas. Un valor alto de precisión significa que la mayoría de las predicciones positivas del modelo son correctas.

- *Recall* (recuperación): La recuperación mide la proporción de verdaderos positivos (objetos correctamente identificados) entre todos los positivos reales del conjunto de datos. Es particularmente importante en aplicaciones donde es fundamental detectar todos los objetos presentes, aunque eso implique un mayor número de falsos positivos. Un alto valor de recuperación significa que el modelo identifica correctamente la mayoría de los objetos que están presentes en las imágenes.

- *F1 score* (puntuación F1): La *F1 score* es una métrica que combina *Precision* y *Recall* en una única medida armónica, permitiendo equilibrar estas dos métricas en casos donde una clase es mucho más frecuente que la otra. Es especialmente útil para evaluar modelos en situaciones donde tanto falsos positivos como los falsos negativos son críticos. Un valor alto de *F1 score* indica que el modelo tiene un buen rendimiento general, detectando correctamente los objetos con un equilibrio adecuado entre precisión y recuperación.

- mAP@0.5 (Mean Average Precision at IoU 0.5, precision media promedio en un umbral IoU de 0.5): El mAP (*Mean Average Precision*) es una métrica utilizada para evaluar el rendimiento de un modelo de detección de objetos, considerando tanto la precisión como la recuperación del modelo. El término "@0.5" se refiere a un umbral específico de *Intersection over Union* (IoU), intersección sobre unión, de 0.5, que es el mínimo solapamiento entre un cuadro delimitador predicho y un cuadro delimitador real para que una detección sea considerada correcta. Un valor alto de mAP@0.5 indica que el modelo es capaz de detectar correctamente los objetos con una alta precisión y recuperación en un umbral de IoU de 0.5.

4.5 EXPERIMENTOS CON MODELO SELECCIONADO

En este apartado se exponen los resultados obtenidos de los experimentos realizados utilizando el modelo seleccionado 'YOLOv8n.pt'. Estos resultados corresponden a los entrenamientos llevados a cabo con la red YOLOv8, empleando los distintos conjuntos de datos creados.

Antes de proceder a detallar los distintos entrenamientos realizados y discutir los resultados obtenidos, es importante señalar que se llevaron a cabo varias pruebas en la que se modificó el parámetro *'freeze'*, que permite bloquear ciertas capas convolucionales del modelo. Para la realización de estas pruebas se utilizó el conjunto de datos llamado *data_normal*, el cual, como se explicó en el capítulo 3, contiene 456 imágenes capturadas en interiores sin distorsión. De estas imágenes, se seleccionaron aleatoriamente 7, que fueron reservadas para la prueba de detección de objetos al finalizar el entrenamiento. El resto de las imágenes se distribuyó en dos carpetas: 360 imágenes en la carpeta *train* para el entrenamiento y 89 imágenes en la carpeta *val* para la validación. Además, en el código de ejecución se estableció para todas las pruebas un total de 150 *epochs*, con un *batch* de 8 y se dispuso que el tamaño de imagen de entrada fuese de 640. Asimismo, se especificó que el proceso de entrenamiento se realizaría utilizando la GPU (*Graphics Processing Unit)* NVIDIA GeForce RTX 3090 disponible en el ordenador.

Las pruebas realizadas tuvieron como objetivo evaluar si el tiempo de entrenamiento varía al modificar el valor del parámetro *freeze*. Las pruebas se efectuaron con los siguientes valores para *freeze*: 0 (sin bloquear capas), 10 (bloqueando 10 capas convolucionales), 22 (bloqueando 22 capas) y 24 (bloqueando todas las capas convolucionales).

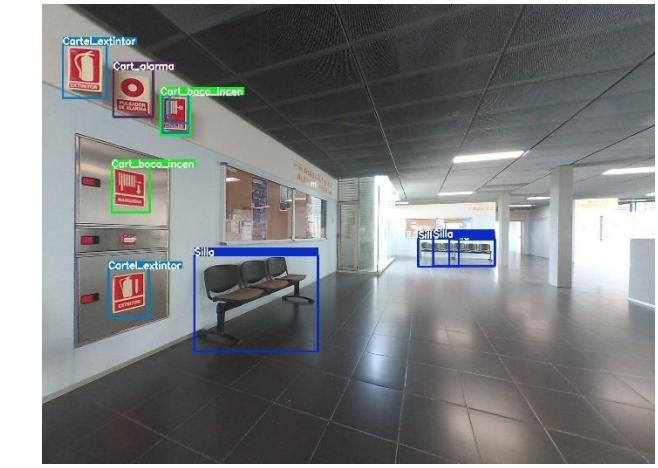
Al emplear el valor de *freeze*=24, se generó un mensaje en la terminal al inicio del proceso, advirtiendo que no era posible continuar debido a la falta de especificación de los pesos de activación necesarios para el modelo. Este mensaje indicaba que, sin proporcionar dichos pesos, el proceso no podía avanzar, ya que las capas convolucionales bloqueadas requieren valores predefinidos para realizar el aprendizaje correctamente.

En el caso del valor *freeze*=10, el tiempo total de entrenamiento con los parámetros establecidos fue de 0.129 horas. Para el valor *freeze*=22, el tiempo total fue de 0.114 horas. Finalmente, cuando se empleó *freeze*=0 (sin bloqueo de capas), el tiempo de entrenamiento fue de 0.149 horas. Los resultados indican que el bloqueo de capas convolucionales reduce el tiempo de entrenamiento.

Por último, se empleó una de las 7 imágenes reservadas para validar la detección de objetos, con el fin de verificar si los modelos obtenidos tras cada uno de los entrenamientos presentaban diferencias en la detección. Se evaluó también si el tiempo de entrenamiento influía en la eficiencia del modelo en la detección de objetos. Esto se expone en las siguientes líneas.

En la Figura 4- 4 se presentan los resultados de los objetos detectados utilizando el modelo obtenido tras entrenar la red con un valor de *freeze* igual a 22. En la figura se observa que la terminal indica la aparición de elementos repetidos en varias ocasiones. A partir de estos datos proporcionados por la terminal, junto con los cuadros delimitadores de los objetos detectados, los cuales se presentan en diferentes colores para cada clase, se puede notar que el cartel de boca de incendios se confunde en dos ocasiones, ya que en esos casos se identifica erróneamente como un cartel de extintor. Esto se observa en el cartel de boca de incendios que aparece en la parte superior de la imagen mostrada en la Figura 4- 4,

pues, además del *bounding box* verde (correspondiente a la clase correcta), se puede ver, en el mismo objeto, dos *bounding boxes* más del color correspondiente a la clase cartel extintor. Además, si se considera que el grupo de sillas debería interpretarse como una sola silla, se observa que la detección es incorrecta en las sillas del fondo de la imagen, ya que se identifica una silla adicional. Sin embargo, en este caso, no se está confundiendo con otro objeto.




```
80x640 1 Cart_alarma, 2 Cart_boca_incens, 4 Cartel_extintors, 4 Sillas, 175.3ms
 shape (1, 3, 480, 640)
LOPs
```

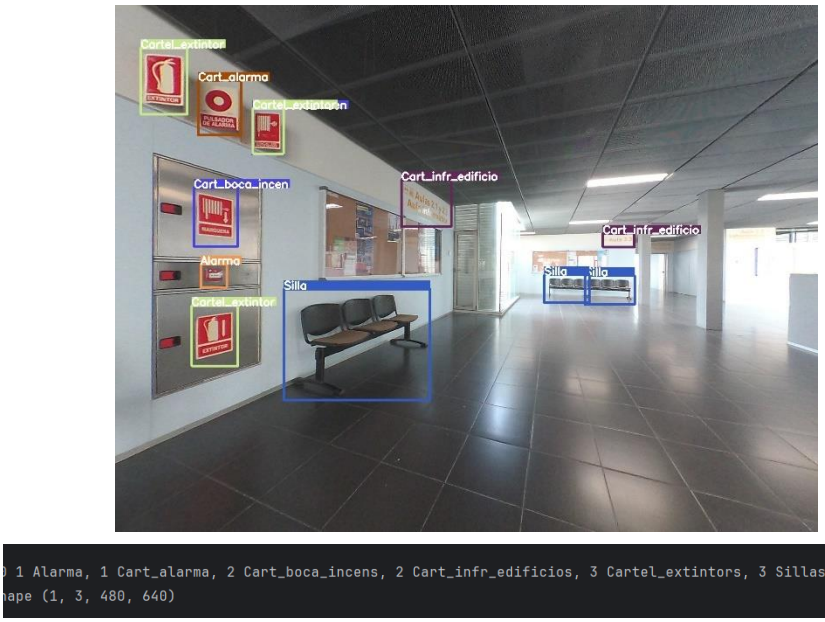
Figura 4- 4. Objetos detectados con freeze=22. En la parte superior de esta imagen con las clases y *bounding boxes* de los objetos detectados e identificados con esta configuración. Mientras que la parte inferior corresponde a una captura de la salida por terminal, donde se indica el tamaño de entrada de la imagen, el número de objetos detectados para cada clase y el tiempo.

Los resultados visuales de la detección de objetos en la imagen utilizada para la validación del modelo con un valor de *freeze* igual a 10 se presentan en la Figura 4- 5. En dicha figura, se muestran los elementos detectados por el modelo, tal como se indica en la terminal del programa PyCharm, y se puede observar que se corresponden con los objetos identificados en la imagen. No obstante, se presenta un error en la detección de uno de los carteles de extintor, ya que el modelo hace referencia incorrectamente a un cartel de boca de incendios.


```
 480x640 1 Alarma, 1 Cart_alarma, 2 Cart_boca_incens, 3 Cart_infr_edificios, 3 Cartel_extintors, 3 Sillas, 1730.3ms
ge at shape (1, 3, 480, 640)
GFLOPs
```

Figura 4- 5. Objetos detectados con freeze=10. En la parte superior de esta imagen con las clases y *bounding boxes* de los objetos detectados e identificados con esta configuración. Mientras que la parte inferior corresponde a una captura de la salida por terminal, donde se indica el tamaño de entrada de la imagen, el número de objetos detectados para cada clase y el tiempo.

En cuanto a los resultados obtenidos al utilizar un valor de *freeze* igual a 0, estos se pueden ver en la Figura 4- 6. En esta imagen, se observa que el modelo es capaz de detectar tanto las sillas como los carteles de información del edificio, que están más claramente visibles en la imagen. Aunque uno de los elementos está mal identificado, si nos fijamos en la imagen de la parte inferior de la Figura 4- 6, se han identificado tres carteles de extintores, cuando en realidad hay dos solamente presentes en la imagen. Este error se debe a la confusión del modelo entre un cartel de extintor y un cartel de boca de incendios, lo cual es evidente en la imagen (parte superior de la Figura 4- 6), donde se muestra el cuadro delimitador del objeto con el nombre de la clase detectada.




```
) 1 Alarma, 1 Cart_alarma, 2 Cart_boca_incens, 2 Cart_infr_edificios, 3 Cartel_extintors, 3 Sillas, 1697.7ms
hape (1, 3, 480, 640)
```

Figura 4- 6. Objetos detectados con freeze=0. En la parte superior de esta imagen con las clases y *bounding boxes* de los objetos detectados e identificados con esta configuración. Mientras que la parte inferior corresponde a una captura de la salida por terminal, donde se indica el tamaño de entrada de la imagen, el número de objetos detectados para cada clase y el tiempo.

Atendiendo a los resultados de las tres pruebas realizadas, se observó una ligera reducción en el tiempo de entrenamiento en los casos en los que se bloquearon las capas convolucionales, en comparación con el caso en el que no se aplicó ningún bloqueo. No obstante, la precisión final en la detección de objetos también se vio afectada negativamente. Por esta razón, y en concordancia con lo indicado en [101], donde se expone un experimento de ejemplo y se menciona que las métricas obtenidas por el modelo no son tan satisfactorias en comparación con los resultados obtenidos en la configuración por defecto (*freeze*=0), se decidio no modificar este parámetro en las pruebas experimentales realizadas en el presente trabajo.

4.5.1 RESULTADO DE LOS ENTRENAMIENTOS EMPLEANDO EL CONJUNTO DE DATOS: *data_normal*

En este punto se analiza el rendimiento del modelo, utilizando el conjunto de datos denominado *data_normal,* descrito previamente en el capítulo 3. Este conjunto contiene 456 imágenes capturadas en interiores sin distorsión, de las cuales 7 fueron seleccionadas aleatoriamente y reservadas para pruebas de detección de objetos una vez finalizado el entrenamiento. Las 449 imágenes restantes se dividieron en dos subconjuntos: 360 se asignaron a la carpeta *train* y 89 a la carpeta *val*. Considerando este conjunto de datos, se evaluó el rendimiento

del modelo mediante diversas métricas clave (*F1 scores, precision, recall* y mAP@0.5), realizando experimentos en los que se modificaron el número de épocas (*epoch*), el número de *batch* y el tamaño de entrada de las imágenes (imgsz). Los resultados obtenidos se presentan en la Tabla 4- 3 y, a continuación, se destacan los más relevantes según los números de épocas y números de *batchs* usados.

Tabla 4- 3. Resultados entrenamientos con conjunto datos *data_normal*.

ENT. Nº	*epochs*	*batchs*	Imgsz	F1 SCORE (%)	PRECISION (%)	RECALL (%)	mAP@0.5 (%)
1	300 (177)*	8	640	76,00	100,00	91,00	85,30
2	300	8	1280	82,00	100,00	95,00	91,60
3	300 (212)*	8	1920	82,00	99,00	96,00	90,80
4	300 (267)*	8	2880	83,00	100,00	92,00	87,10
5	300	16	640	81,00	100,00	92,00	89,00
6	300 (230)*	16	1280	79,00	100,00	95,00	90,50
7	300 (207)*	16	1920	77,00	100,00	94,00	88,70
8	150	8	640	74,00	99,00	91,00	84,60
9	150	8	1280	79,00	99,00	92,00	86,50
10	150	8	1920	79,00	100,00	93,00	85,30
11	150	8	2880	76,00	100,00	89,00	83,70
12	150	16	640	77,00	100,00	93,00	87,80
13	150	16	1280	78,00	100,00	95,00	90,30
14	150	16	1920	78,00	100,00	95,00	87,70

*Estos entrenamientos se detuvieron antes de completar las 300 épocas programadas debido a la activación del parámetro de detección temprana, el cual interrumpió el proceso al no observarse mejora en las últimas 100 épocas. En estos casos, se muestra el número de épocas alcanzadas entre paréntesis.

En primer lugar, analizando los resultados obtenidos para un total de 300 *epochs* y un tamaño de *batch* de 8, se realizaron cuatro entrenamientos con diferentes tamaños de imagen de entrada: 640, 1280, 1920 y 2880 píxeles. El mejor resultado se observó en el entrenamiento número 2, con un tamaño de imagen de entrada de 1280, donde se alcanzó un *F1 score* de 82,00 %, una precisión perfecta del 100,00 %, un *recall* del 95,00 % y un mAP@0.5 de 91,60 %. La elección de este entrenamiento se justifica porque presenta el mayor valor de mAP@0.5 entre todas las pruebas, lo que sugiere una capacidad superior para detectar objetos de manera precisa. Además, el equilibrio entre la precisión perfecta y un alto *recall* refuerza su posición como el mejor modelo dentro de este grupo, logrando un excelente rendimiento general.

El caso de los experimentos con 300 *epochs* y un *batch size* de 16, se consideraron tres tamaños de imagen de entrada: 640, 1280 y 1920 píxeles. El mejor resultado fue el obtenido en el entrenamiento número 5, donde se utilizó un tamaño de imagen de entrada de 640 píxeles. En este caso, se logró un *F1 score,* de 81,00 %, una precisión perfecta del 100,00 %, un *recall* del 92,00 % y un mAP@0.5 de 89,00 %. A pesar de que su mAP@0.5 es inferior al del entrenamiento número 2, este modelo destacó en esta configuración de *batch size* debido a su balance entre las métricas evaluadas, lo que lo posiciona como el más robusto para esta configuración.

En los experimentos con 150 *epochs* y un *batch size* de 8, se evaluaron los tamaños de imagen de entrada 640, 1280, 1920 y 2880 píxeles. El mejor entrenamiento dentro de esta configuración fue el número 9, donde se utilizó un tamaño de imagen de 1280 píxeles. Este modelo alcanzó un *F1 score* de 79,00 %, una precisión de 99,00 %, un *recall* de 92,00 % y un mAP@0.5 de 86,50 %. El entrenamiento fue seleccionado como el mejor debido a su equilibrio en todas las métricas, destacando especialmente en su mAP@0.5 y *F1 score* en comparación con los otros entrenamientos realizados con 150 *epochs*.

Por último, en los experimentos con 150 *epochs* y un *batch size* de 16, se llevaron a cabo tres entrenamientos con diferentes tamaños de imagen: 640, 1280 y 1920 píxeles. El entrenamiento más destacado fue el número 13, con un tamaño de imagen de entrada de 1280, en el cual se alcanzó un *F1 score* de 78,00 %, una precisión perfecta del 100,00 %, un *recall* del 95,00 % y un mAP@0.5 de 90,30 %. Este modelo fue seleccionado como el mejor por su excelente rendimiento en cuanto a mAP@0.5, así como por su buen balance entre *precision* y *recall*, lo que indica una alta capacidad para detectar objetos de manera precisa y consistente.

Cabe destacar que los entrenamientos 1, 3, 4, 6 y 7 se detuvieron antes de cumplir las 300 *epochs* previstas. Esto se debió a la activación del parámetro de detención temprana, el cual interrumpe el entrenamiento si no se observa ninguna mejora en las últimas 100 épocas. En el caso del entrenamiento 1, el proceso se detuvo en la *epoch* 177; en el entrenamiento 3, en la *epoch* 212; en el entrenamiento 4, en la *epoch* 267; en el entrenamiento 6, en la *epoch* 230; en el entrenamiento 7, en la *epoch* 207. Los mejores resultados de cada uno de estos entrenamientos fueron almacenados como best.pt.

Si se desea modificar este comportamiento de detección temprana, es posible ajustar el valor de paciencia (*patience*), o incluso desactivar la detención temprana por completo usando el parámetro *patience*=0. Cabe recordar que este parámetro ya fue explicado en capítulos anteriores.

Finalmente, al comparar los mejores entrenamientos de cada una de las configuraciones analizadas, se distingue que el entrenamiento número 2 (300 *epochs*, *batch* 8 y tamaño de imagen de entrada de 1280) es el más robusto de todos. Este modelo no solo logra un *F1 score* alto (82,00 %), sino que también mantiene una precisión perfecta del 100,00 % y un *recall* elevado del 95,00 %, junto con el mAP@0.5 más alto (91,60 %). Estos resultados demuestran que este entrenamiento ofrece el mejor rendimiento en la detección de objetos y se

consolida como el mejor modelo en términos generales, debido a su capacidad para mantener un equilibrio sobresaliente entre todas las métricas evaluadas.

La imagen de la Figura 4- 7 muestra una gráfica de una curva de *precision-recall,* perteneciente al entrenamiento número 2, utilizada para evaluar el rendimiento del modelo de clasificación. En este gráfico, el eje horizontal (eje X) representa el *recall* o recuperación, que indica la proporción de instancias positivas correctamente identificadas por el modelo. El eje vertical (eje Y) muestra la *precision*, que mide cuántas de las instancias predichas como positivas son realmente positivas.

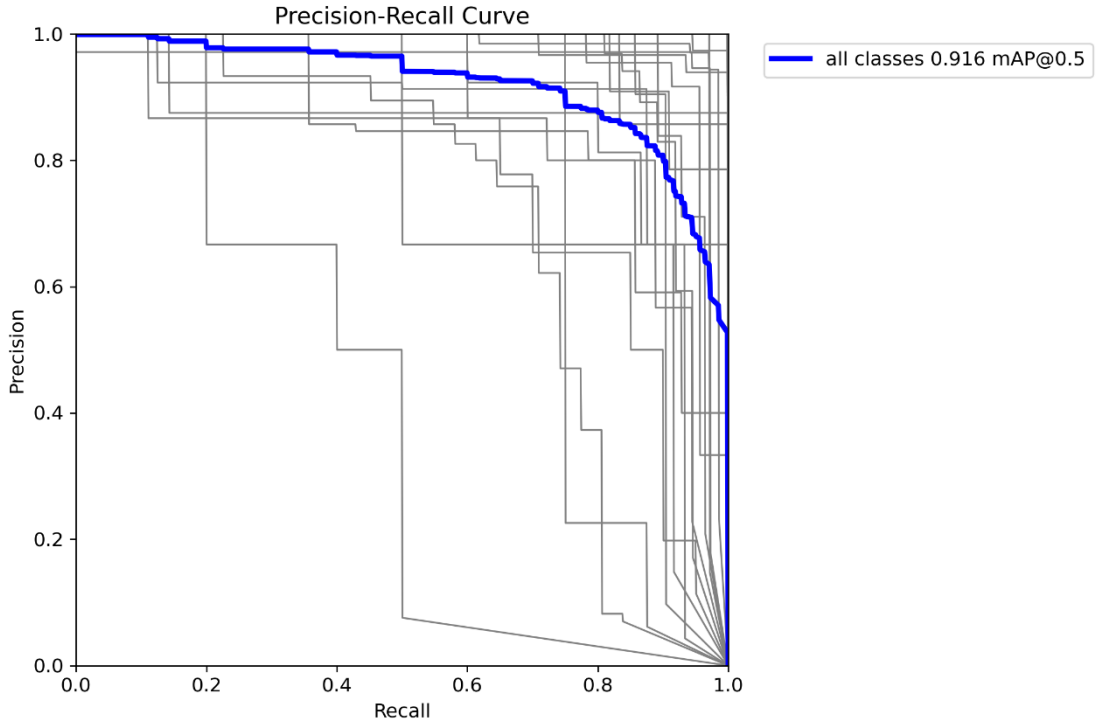


Figura 4- 7. Curva de *precision-recall* entrenamiento número 2 (300 *epochs*, 8 *batchs* y 1280 de tamaño en la imagen de entrada), que es el que ha obtenido mejores resultados con la base de datos *"data_normal"*.

En el gráfico destaca una curva azul gruesa, que representa el desempeño promedio del modelo para todas las clases. Según la leyenda en la parte superior derecha, esta curva corresponde a *all classes* (todas las clases) con un valor de mAP (*mean Average Precision*) de 0.916 a un umbral de IoU (*Intersection over Union*) de 0.5, un estándar común para evaluar modelos de detección de objetos. El valor mAP de 0.916 indica un buen equilibrio entre *precision* y *recall*, lo que sugiere un alto rendimiento del modelo.

Además, el gráfico incluye varias líneas girses delgadas, que corresponden a las curvas individuales de *precision-recall* para cada clase. Estas líneas muestran cómo varía el rendimiento del modelo a través de diferentes clases.

En conjunto, la imagen refleja un modelo con un rendimiento general sólido, con un mAP alto que indica una capacidad efectiva para hacer predicciones precisas y con buena cobertura en todas las clases evaluadas.

La Figura 4- 8 muestra una curva *F1 score-Confidence* que ilustra la relación entre la confianza de las predicciones de un modelo y el valor del *F1 score* para todas las clases en un problema de clasificación. En el eje X se representa el nivel de confianza, que varía de 0 (baja confianza) a 1 (alta confianza), mientras que en el eje Y se muestra el *F1score*, una métrica que combina la precisión y el *recall*, con valores entre 0 y 1, donde 1 representa el mejor desempeño posible.

La curva azul gruesa destaca el comportamiento del *F1 score* promedio para todas las clases conforme varía el umbral de confianza. El modelo alcanza un valor máximo de *F1 score* de 0.82 cuando la confianza es de 0.514, lo que indica que en ese punto el modelo tiene su rendimiento óptimo.

Además, las curvas grises representan el *F1 score* para cada clase individualmente, mostrando que hay variaciones significativas en el rendimiento entre clases para diferentes niveles de confianza. Algunas clases alcanzan un mayor *F1 score* a distintos niveles de confianza, lo que sugiere que el comportamiento del modelo puede no ser uniforme para todas las clases.

Esta visualización es útil para analizar cómo varía el rendimiento del modelo en términos del *F1 score* global, y por clase a medida que cambia el umbral de confianza, proporcionando información valiosa para ajustar y evaluar el modelo.

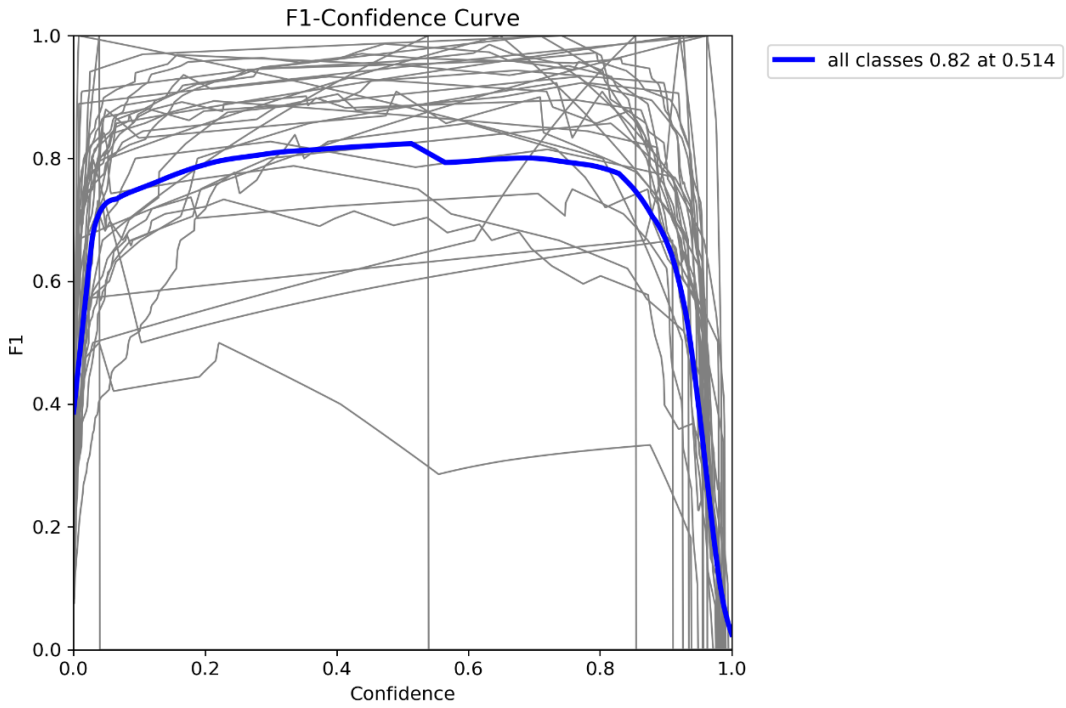


Figura 4- 8. Curva *F1 score-confidence* entrenamiento número 2 (300 *epochs*, 8 *batchs* y 1280 de tamaño en la imagen de entrada), que es el que ha obtenido mejores resultados con la base de datos *"data_normal"*.

4.5.2 RESULTADO DE LOS ENTRENAMIENTOS EMPLEANDO EL CONJUNTO DE DATOS: *data_ojo_pez*

Los resultados de los entrenamientos realizados se presentan en la Tabla 4-4. En primera instancia, se analizan los mejores resultados basados en la cantidad de épocas (*epochs*) y el número de lotes por iteración (*batch*), evaluando cuidadosamente las métricas asociadas a cada uno de los experimentos. A continuación, se identificó cuál de estos entrenamientos resultó ser el mejor para este conjunto de datos.

El rendimiento del modelo se evaluó utilizando las siguientes métricas: *F1 score*, precisión (*precision*), recuperación (*recall*) y el promedio de precisión bajo un umbral de 0.5 (mAP@0.5). Durante los experimentos, se variaron parámetros clave como el número de épocas (*epochs*), el tamaño de los lotes (*batch*) y el tamaño de entrada de las imágenes (imgsz). Esta metodología facilitó un análisis minucioso de las distintas configuraciones, con el fin de identificar el entrenamiento más eficiente. Dicho entrenamiento incorpora la configuración óptima, logrando el mejor equilibrio entre precisión, recuperación y el rendimiento general del modelo.

Para el desarrollo de estos experimentos, se empleó el conjunto de datos denominado *data_ojo_pez,* que, tal como se mencionó en el capítulo 3, consta de un total de 552 imágenes con distorsión. De este conjunto, 7 imágenes fueron seleccionadas aleatoriamente y reservadas exclusivamente para la fase de pruebas, con el propósito de llevar a cabo evaluaciones de detección de objetos una vez completado el proceso de entrenamiento. Estas 7 imágenes no fueron incluidas en las etapas de entrenamiento ni de validación. Las 545 imágenes restantes se dividieron en dos subconjuntos: 436 imágenes se destinaron a la fase de entrenamiento (*train*), mientras que 109 se utilizaron para la fase de validación (*val*).

Tabla 4- 4. Resultados entrenamientos con conjunto de datos *data_ojo_pez*.

ENT. Nº	*epochs*	*batchs*	Imgsz	F1 SCORE (%)	PRECISION (%)	RECALL (%)	mAP@0.5 (%)
15	300	8	640	77,00	100,00	88,00	83,40
16	300	8	1280	85,00	100,00	93,00	89,30
17	300 (235)*	8	1920	84,00	100,00	93,00	90,70
18	300	8	2880	83,00	100,00	91,00	89,10
19	300	16	640	76,00	100,00	89,00	82,60
20	300	16	1280	84,00	100,00	93,00	88,90
21	300 (261)*	16	1920	87,00	100,00	94,00	90,80
22	150	8	640	73,00	100,00	88,00	80,80
23	150	8	1280	79,00	100,00	93,00	87,80
24	150	8	1920	83,00	100,00	94,00	89,60
25	150	8	2880	84,00	100,00	91,00	87,00
26	150	16	640	72,00	100,00	88,00	78,70
27	150	16	1280	81,00	100,00	92,00	88,00
28	150	16	1920	83,00	100,00	93,00	88,40

*Estos entrenamientos se detuvieron antes de completar las 300 épocas programadas debido a la activación del parámetro de detección temprana, el cual interrumpió el proceso al no observarse mejora en las últimas 100 épocas. En estos casos, se muestra el número de épocas alcanzadas entre paréntesis.

En primera instancia, se analizan los resultados correspondientes a los experimentos realizados con 300 *epochs* y con un *batch* de 8, utilizando tamaños de imagen de entrada de 640, 1280, 1920 y 2880 píxeles. El mejor resultado dentro de esta configuración se observó en el entrenamiento número 17, donde se utilizó un tamaño de imagen de 1920 píxeles. En este caso, se alcanzó un *F1 score* de 84,00 %, una precisión perfecta del 100,00 %, un *recall* del 93,00 % y un mAP@0.5 de 90,70 %. La elección de este modelo se justifica por presentar el mayor mAP@0.5 dentro de este grupo de pruebas, lo que refleja una capacidad superior para detectar objetos de manera precisa. Además, el equilibrio entre la precisión perfecta y una alto *recall* refuerza la elección de este modelo como el mejor entre los entrenamientos con 300 *epochs* y *batch* 8. Es importante destacar que el entrenamiento número 17 se detuvo antes de cumplir las 300 *epochs* previstas, finalizando en la *epoch* 235. Esto ocurrió porque se activó el parámetro de detección temprana, que detiene el entrenamiento si no se observa ninguna mejora en las últimas 100 épocas. Guardando como best.pt el mejor modelo, que se observó en la época 135.

En el caso de los entrenamientos con 300 *epochs* y un *batch* de 16, se llevaron a cabo experimentos con tamaños de imagen de entrada de 640, 1280 y 1920 píxeles. El mejor resultado fue obtenido en el entrenamiento número 21, utilizando un tamaño de imagen de 1920 píxeles. Este modelo alcanzó un *F1 score* de 87,00 %, una precisión perfecta del 100,00 %, un *recall* del 94,00 % y un mAP@0.5 de 90,80 %. Es importante señalar que, al igual que en el caso del

entrenamiento 17, el entrenamiento número 21 se detuvo antes de completar las 300 *epcohs*, finalizando en la *epoch* 261 debido a la activación del parámetro de detección temprana, guardando como best.pt el modelo observado en la *epoch* 161. Este entrenamiento destacó claramente en comparación con los otros experimentos dentro de esta configuración debido a su combinación de altos valores en *F1 score* y mAP@0.5, lo que lo posiciona como el más robusto y preciso al hacer balance entre todas las métricas evaluadas.

A continuación, se evaluaron los resultados obtenidos con 150 *epochs* y un *batch* de 8, donde también se experimentó con tamaños de imagen de 640, 1280, 1920 y 2880 píxeles. El mejor desempeño en esta configuración se observó en el entrenamiento número 24, utilizando un tamaño de imagen de 1920 píxeles. Este modelo alcanzó un *F1 score* de 83,00 %, una precisión perfecta del 100,00 %, un *recall* del 94,00 % y un mAP@0.5 de 89,60 %. El entrenamiento fue seleccionado como el mejor de esta categoría debido a su sobresaliente rendimiento en términos de mAP@0.5 y su excelente equilibrio entre *precision* y *recall*, lo que sugiere una capacidad sólida para la detección de objetos.

Finalmente, en los entrenamientos realizados con 150 *epcohs* y un *batch* de 16, se consideraron tres tamaños de imagen de entrada: 640, 1280 y 1920 píxeles. El mejor resultado se observó en el entrenamiento número 28, donde se utilizó un tamaño de imagen de 1920 píxeles. En este caso, se logró un *F1 score* de 83,00 %, una precisión del 100,00 %, un *recall* del 93,00 % y un mAP@0.5 de 88,40 %. Este modelo fue seleccionado como el mejor en su configuración debido a su sólido rendimiento general, con especial énfasis en el equilibrio entre todas las métricas evaluadas.

Tras comparar los mejores resultados de cada una de las configuraciones analizadas se concluye que el entrenamiento número 21 (300 *epochs*, *batch* de 16 y tamaño de imagen de entrada de 1920 píxeles) es el más robusto de todos. Este modelo logró el *F1 score* más alto con 87,00 %, manteniendo una precisión del 100,00 % y un *recall* elevado de 94,00 %, acompañado por el mAP@0.5 más alto de 90,80 %. Estos resultados reflejan que este entrenamiento ofrece el mejor rendimiento global en la detección de objetos, destacándose por su capacidad para mantener un equilibrio sobresaliente entre todas las métricas evaluadas, consolidándose, así como el mejor modelo entre todas las configuraciones probadas.

La gráfica presentada en la Figura 4- 9 muestra la curva de precisión-recuperación (*Precision-Recall Curve*) generada tras el entrenamiento número 21. Esta curva ilustra la relación entre la precisión, representada en el eje vertical (eje Y), y el *recall*, en el eje horizontal (eje X). La curva azul más gruesa refleja el rendimiento general del modelo para todas las clases, mientras que las líneas grises más finas representan el resultado para cada clase individualmente.

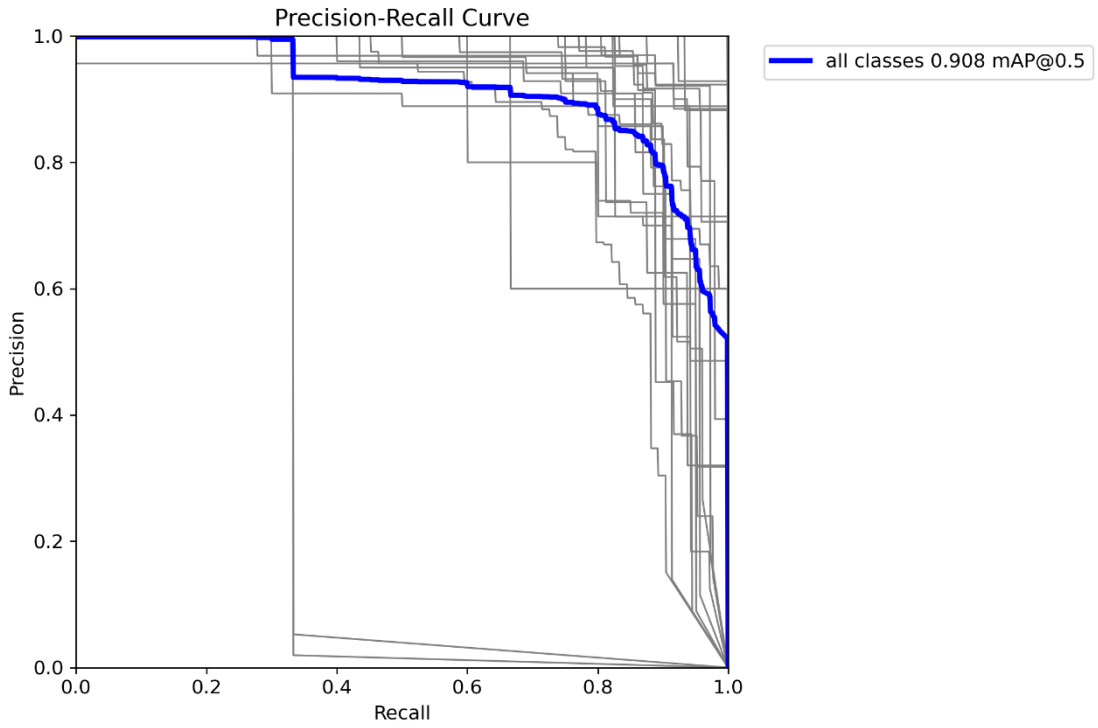


Figura 4- 9. Curva de *precision-recall* entrenamiento número 21 (300 *epochs*, 16 *batchs* y 1920 de tamaño en la imagen de entrada), que es el que ha obtenido mejores resultados con la base de datos *"data_ojo_pez"*.

La precisión se define como la fracción de predicciones correctas entre todas las realizadas por el modelo, mientras que el *recall*, o recuperación, es la fracción de verdaderos positivos sobre todos los positivos reales. Un aspecto importante de la gráfica es el valor de mAP@0.5, que se refiere al *mean Average Precision* con un umbral de Intersección sobre la Unión (IoU) de 0.5. En este caso, el mAP reportado es de 0.908, lo que indica un resultado elevado del modelo; un mAP cercano a 1.0 sugiere una alta precisión en la clasificación y detección de objetos.

Resumiendo, la curva azul demuestra que el entrenamiento posee una buena precisión y *recall*, manteniéndose cerca del valor 1 en ambos ejes, lo que significa que es capaz de identificar correctamente la mayoría de los objetos con alta precisión hasta cierto punto, antes de que se produzca una caída, un comportamiento normal en estas curvas.

La Figura 4- 10 muestra una gráfica de la curva de confianza F1 score, también conocida como la curva *F1-confidence Curve*. En el eje X se representa la confianza, que varía de 0 a1, y en el eje Y se muestra el F1 score, que también varía de 0 a 1.

Hay varias curvas en gris que corresponden a diferentes clases o modelos individuales, y una curva resaltada en azul más gruesa que representa el rendimiento promedio o general de todas las clases. En la leyenda, se indica que,

para todas las clases, el valor F1 score máximo es 0.87, obtenido a un nivel de confianza de 0.440.

Esto significa que, en promedio, el mejor rendimiento de clasificación (según el F1 score) para todas las clases ocurre cuando se realiza una confianza de 0.440 en el modelo de clasificación.

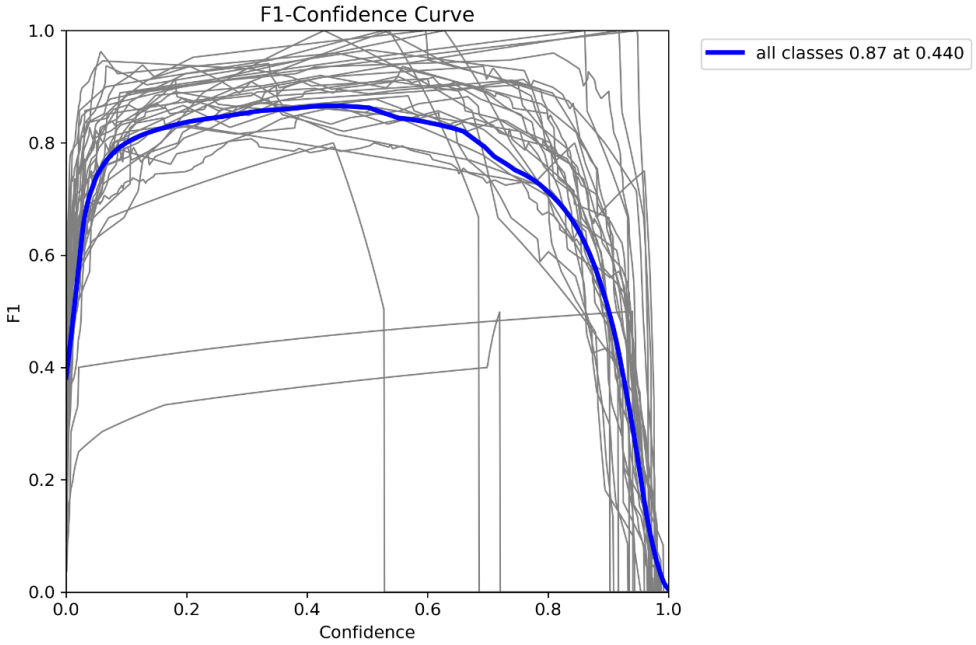


Figura 4- 10. Curva *F1 score-confidence* entrenamiento número 21 (300 *epochs*, 16 *batchs* y 1920 de tamaño de imagen de entrada), que es el que ha obtenido mejores resultados con la base de datos *"data_ojo_pez"*.

4.5.3 RESULTADO DE LOS ENTRENAMIENTOS EMPLEANDO EL CONJUNTO DE DATOS: *data_total*

El análisis de este nuevo conjunto de datos (*data_total*) se ha realizado siguiendo el mismo enfoque que en los casos anteriores. Identificando, en primer lugar, los resultados más destacados dependiendo del número de épocas (*epochs)* y tamaño de lotes (*batch)*. Los resultados pueden ser consultados en la Tabla 4- 5.

El rendimiento del modelo se evaluó utilizando las siguientes métricas: *F1 score*, precisión (*precision*), recuperación (*recall*) y el promedio de precisión bajo un umbral de 0.5 (mAP@0.5). Durante los experimentos, se variaron los parámetros correspondientes a: el número de épocas (*epochs*), el tamaño de los lotes (*batch*) y el tamaño de entrada de las imágenes (imgsz). Este procedimiento permitió un análisis exhaustivo de las distintas configuraciones, con el fin de determinar el entrenamiento más eficiente. Dicho entrenamiento incorpora la configuración óptima, logrando un equilibrio adecuado entre precisión, recuperación y el rendimiento general del modelo.

El conjunto de datos empleado en los experimentos es el denominado *data_total* que está compuesto por un total de 1008 imágenes con y sin distorsión, tal como se describió en el capítulo 3 del presente trabajo. De este conjunto, se seleccionaron aleatoriamente 14 imágenes que fueron reservadas exclusivamente para la fase de pruebas, y se usaron para evaluar la capacidad del modelo en la detección de objetos después del proceso de entrenamiento. Es importante destacar que estas 14 imágenes no fueron utilizadas en las fases de entrenamiento ni de validación. Las restantes 994 imágenes se dividieron en dos subconjuntos: 796 se utilizaron para la fase de entrenamiento (*train*) y 198 para la fase de validación (*val*).

Tabla 4- 5. Resultados entrenamientos con conjunto de datos *data_total*.

ENT. Nº	*epochs*	*batchs*	Imgsz	F1 SCORE (%)	PRECISION (%)	RECALL (%)	mAP@0.5 (%)
29	300	8	640	81,00	100,00	87,00	83,40
30	300 (238)*	8	1280	87,00	100,00	93,00	89,90
31	300 (211)*	8	1920	82,00	100,00	86,00	82,30
32	300 (268)*	8	2880	85,00	100,00	92,00	87,70
33	300	16	640	82,00	100,00	87,00	83,80
34	300	16	1280	86,00	100,00	93,00	89,50
35	300	16	1920	84,00	100,00	93,00	89,50
36	150	8	640	78,00	100,00	86,00	80,70
37	150	8	1280	86,00	100,00	93,00	89,80
38	150	8	1920	84,00	100,00	92,00	88,00
39	150	8	2880	84,00	100,00	92,00	87,40
40	150	16	640	79,00	100,00	87,00	81,50
41	150	16	1280	87,00	100,00	93,00	89,60
42	150	16	1920	85,00	100,00	93,00	89,20

*Estos entrenamientos se detuvieron antes de completar las 300 épocas programadas debido a la activación del parámetro de detección temprana, el cual interrumpió el proceso al no observarse mejora en las últimas 100 épocas. En estos casos, se muestra el número de épocas alcanzadas entre paréntesis.

Analizando los resultados obtenidos para un total de 300 *epochs* y un tamaño de *batch* de 8. En este grupo, se realizaron cuatro entrenamientos con diferentes tamaños de imagen de entrada: 640, 1280, 1920 y 2880 píxeles. El mejor resultado se observó en el entrenamiento número 30, que se detuvo a las 238 *epochs* debido a la implementación del parámetro de detención temprana, el cual se activa si no se observa ninguna mejora en las últimas 100 épocas. En este caso, se alcanzó un *F1 score* de 87,00 %, una precisión del 100,00 %, un *recall* del 93,00 % y un mAP@0.5 de 89,90 %. La elección de este entrenamiento se justifica por ser el que presenta el mayor valor de mAP@0.5 de 89,90 %. La elección de este entrenamiento se justifica por ser el que presenta el mayor valor de mAP@0.5

dentro de esta configuración, lo que indica una mejor capacidad para detectar objetos de manera precisa. Además, el balance entre la *precision* y un *recall* elevado refuerza su posición como el mejor modelo dentro de este grupo, logrando un excelente rendimiento general.

A continuación, en el caso de los experimentos con 300 *epochs* y un *batch* de 16, se evaluaron también diferentes tamaños de imagen: 640, 1280 y 1920 píxeles. El mejor resultado fue obtenido en el entrenamiento número 34, con un tamaño de imagen de 1280 píxeles. Este entrenamiento alcanzó un *F1 score* de 86,00 %, una precisión del 100,00 %, un *recall* del 93,00 % y un mAP@0.5 de 89,50 %. Aunque este valor de mAP@0.5 es ligeramente inferior al del entrenamiento número 30, este modelo fue seleccionado por mantener un alto equilibrio en todas las métricas evaluadas, lo que lo posiciona como el más robusto para esta configuración.

En el grupo de experimentos con 150 *epochs* y un *batch* de 8, se analizaron nuevamente cuatro tamaños de imagen: 640, 1280, 1920 y 2880 píxeles. El mejor resultado se obtuvo en el entrenamiento número 37, con un tamaño de imagen de entrada de 1280 píxeles. En este caso, el modelo alcanzó un F1 score de 86,00 %, una precisión del 100,00 %, un *recall* del 93,00 % y un mAP@0.5 de 89,80 %. Este entrenamiento se seleccionó por ofrecer el mayor valor de mAP@0.5 dentro de esta configuración, lo que sugiere una excelente capacidad para detectar objetos con gran precisión y un buen equilibrio en todas las métricas.

Por último, en los experimentos con 150 *epochs* y un *batch* de 16, se llevaron a cabo tres entrenamientos con tamaños de imagen de entrada de 640, 1280 y 1920 píxeles. El mejor resultado fue obtenido en el entrenamiento número 41, utilizando un tamaño de imagen de entrada de 1280 píxeles, donde se alcanzó un *F1 score* de 87,00 %, una precisión del 100,00 %, un *recall* del 93,00 % y un mAP@0.5 de 89,60 %. Este modelo se eligió debido a su excelente rendimiento en términos de mAP@0.5, y su buen equilibrio entre posición y *recall*, lo que evidencia una alta capacidad para detectar objetos de manera precisa y consistente.

Resaltar que los entrenamientos 30, 32 y 31 se detuvieron antes de cumplir las 300 *epochs* previstas. El entrenamiento 30 se detuvo en la *epoch* 238, el entrenamiento 32 se detuvo en la *epoch* 268, y el entrenamiento 31 se detuvo en la 211 *epoch*. Esto ocurrió porque se tiene activo el parámetro de detención temprana, que actúa si no se observa ninguna mejora en las últimas 100 épocas. Si se desea ajustar este parámetro para modificar la detención temprana, es posible pasar un nuevo valor de paciencia o indicar el uso de paciencia=0 para deshabilitar la detención temprana.

Al comparar los mejores entrenamientos de cada configuración analizada, se observa que el entrenamiento número 30 (300 *epochs*, *batch* 8 y tamaño de imagen de entrada de 1280) es el modelo más robusto de todos. Este modelo no solo alcanza el mayor valor de mAP@0.5 (89,90 %), sino que también logra un *F1 score* alto (87,00 %), una *precision* del 100,00 % y un *recall* elevado del 93,00 %. Estos resultados demuestran que este modelo ofrece el mejor rendimiento en la detección de objetos, consolidándose como el más completo y equilibrado en términos generales.

La gráfica mostrada en la Figura 4- 11 expone una curva de precisión y recuperación (*Precision-Recall Curve*) generada tras el entrenamiento número 30. En esta gráfica, el eje horizontal (eje X) representa la recuperación (*recall*), que mide la capacidad del modelo para identificar todas instancias positivas, mientras que el eje vertical (eje Y) representa la precisión (*precision*), que evalúa como de exactos son los resultados positivos predichos por el modelo.

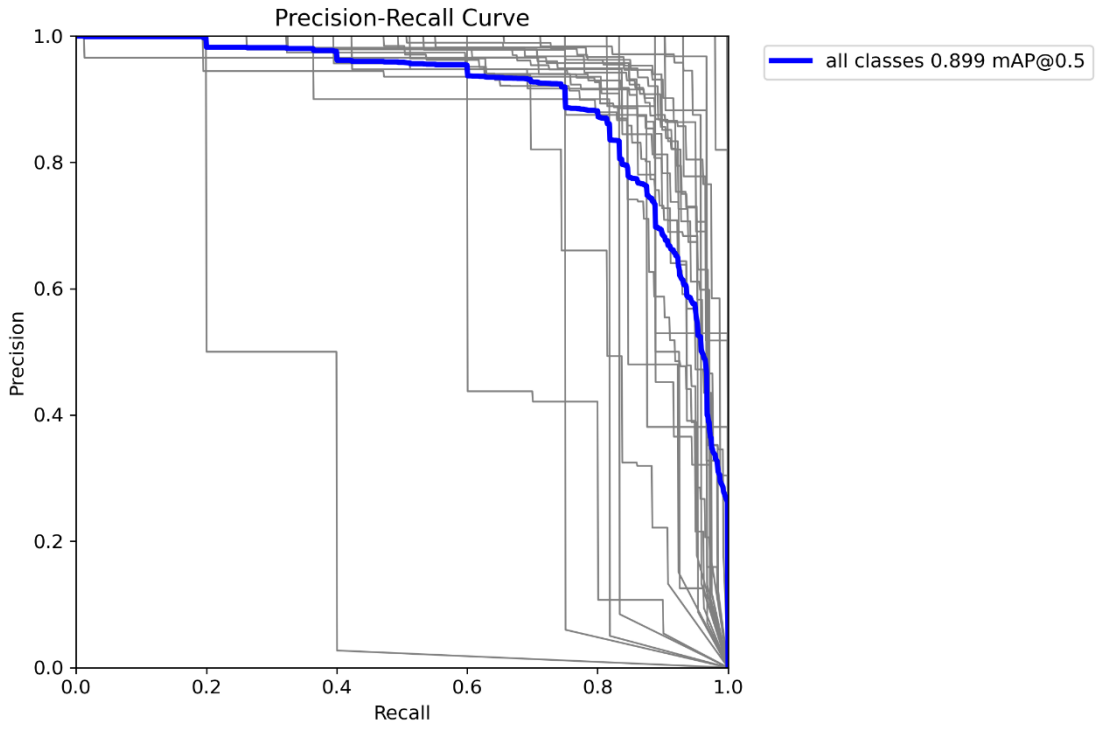


Figura 4- 11. Curva de *precision-recall* entrenamiento número 30 (300 *epochs*, 8 *batchs* y 1280 de tamaño en la imagen de entrada), que es el que ha obtenido mejores resultados con la base de datos *"data_total"*.

La línea azul gruesa que aparece en la gráfica representa el rendimiento promedio para todas las clases, y se destaca en la leyenda que el *mean Average Precision* (mAP) del modelo es de 0.899 a un umbral de 0.5 de IoU (*Intersection over Union*). Este valor indica que el modelo tiene un buen rendimiento general, logrando un equilibrio adecuado entre precisión y recuperación.

Por otro lado, las líneas grises más delgadas representan las curvas individuales de precisión y recuperación para diferentes clases específicas del entrenamiento. Estas líneas muestran cierta variabilidad, lo que sugiere que algunas clases tienen un rendimiento más inestable en comparación con el promedio general.

Atendiendo a la Figura 4- 12, en la que se muestra la curva de confianza de *F1 score*, perteneciente al entrenamiento número 30. En ella se grafican distintas

curvas correspondientes a varias clases o subconjuntos de datos, representadas por líneas grises, junto con una curva promedio resaltada en color azul.

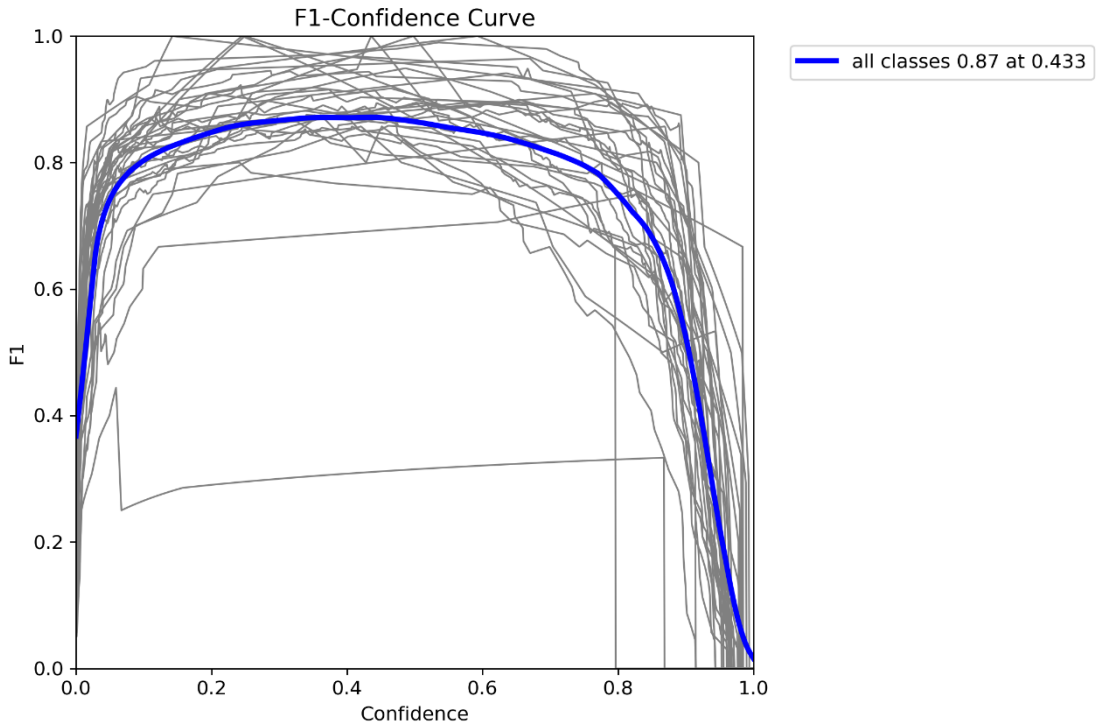


Figura 4- 12. Curva *F1 score-confidence* entrenamiento número 30 (300 *epochs*, 8 *batchs* y 1280 de tamaño de imagen de entrada), que es el que ha obtenido mejores resultados con la base de datos *"data_total"*.

El eje horizontal (eje X) representa el nivel de confianza, que varía entre 0 y 1, mientras que el eje vertical (eje Y) muestra la métrica *F1 score*, que también varía en el mismo rango. Las múltiples curvas grises reflejan el desempeño de diversas clases o modelos a diferentes niveles de confianza, evidenciando fluctuaciones en las puntuaciones *F1 score* a medida que aumenta la confianza.

La curva azul, que se destaca sobre el resto, muestra el promedio del comportamiento de todas las clases en términos de relación entre la confianza y la puntuación *F1 score*. Según la leyenda, esta curva promedio alcanza una puntuación *F1 score* de 0.87 cuando la confianza es de 0.433.

Este gráfico es útil para visualizar cómo varía el valor de *F1 score* según el nivel de confianza en un modelo de clasificación, destacando tanto el rendimiento global (curva azul) como el desempeño de las clases individuales (curvas grises).

4.6 ANÁLISIS VISUAL DE DETECCIÓN DE OBJETOS EN IMÁGENES OJO DE PEZ UTILIZANDO LOS MEJORES MODELOS OBTENIDOS EN LOS ENTRENAMIENTOS.

En el presente apartado, se lleva a cabo un análisis comparativo entre los tres mejores modelos entrenados, evaluando su capacidad para detectar objetos en imágenes capturadas con lentes de ojo de pez. Este tipo de lentes generan una distorsión esférica característica, que añade complejidad al proceso de detección de objetos. Para realizar esta evaluación, se emplean dos imágenes tomadas en el interior de los edificios de la Universidad Miguel Hernández de Elche: una en el edificio Altet (Figura 4- 13 a)) y otra en el edificio Arenals (Figura 4- 13 b)), cuyas características de captación permiten analizar la precisión de los modelos bajo diferentes condiciones de distorsión.

a) **b)**

Figura 4- 13. Imágenes ojo de pez para comprobación visual. **a)** Tomada en el interior edificio Arenals, **b)** Captada en el interior edificio Altet.

Inicialmente, se utiliza la imagen correspondiente a la Figura 4- 13 a) para evaluar visualmente el rendimiento de los tres modelos obtenidos en el entrenamiento 2 (imágenes sin distorsión, ver Apartado 4.5.1), 30 (imágenes con y sin distorsión, ver Apartado 4.5.3) y 21 (imágenes con distorsión, ver Apartado 4.5.2).

Al observar el resultado mostrado en la Figura 4- 14, correspondiente al modelo obtenido del entrenamiento 2, se puede apreciar que el modelo identifica varios objetos presentes en la imagen: un cartel de aula, una fuente de agua, una máquina de café, una máquina de refrescos, una máquina de *snacks* y dos personas. No obstante, al analizar detalladamente los "*bounding boxes*" (rectángulos que delimitan los objetos), se detecta un error significativo: el modelo duplica la detección de una misma persona, generando dos rectángulos superpuestos que la representan incorrectamente como dos individuos. Además,

los rectángulos no engloban completamente el cuerpo de la persona, lo que indica una falta de precisión en la delimitación del objeto. Estos resultados son claramente visibles en la Figura 4- 14, donde se evidencia esta inexactitud visualmente. Este error se debe, probablemente, a la distorsión esférica de la imagen, que este modelo no ha logrado manejar adecuadamente.

Figura 4- 14. Resultado detección objetos usando modelo entrenamiento 2 e imagen Figura 4- 13. a).

Por otro lado, la Figura 4- 15 muestra el resultado de aplicar el modelo correspondiente al entrenamiento 30. Aquí, se observa una mejora notable en la precisión de las detecciones, respecto al modelo anterior. Se detectan los mismos objetos que en el caso anterior: cartel de aula, fuente de agua, persona, máquinas de café, refrescos y *snacks*, de manera correcta. La persona presente en la imagen aparece ahora delimitada por un único *bounding box*, no teniendo el error observado en el modelo anterior. En la figura se puede ver que los rectángulos delimitadores abarcan de manera precisa y completa a los objetos detectados, lo que demuestra que este modelo maneja más eficazmente la distorsión esférica.

Figura 4- 15. Resultado detección objetos usando modelo entrenamiento 30 e imagen Figura 4- 13. a).

El modelo del entrenamiento 21, cuyos resultados se muestran en la Figura 4- 16, exhibe un rendimiento similar al del entrenamiento 30, logrando una detección correcta de todos los objetos presentes en la imagen, incluyendo la persona, que aparece nuevamente dentro de un único rectángulo delimitador. En la mencionada figura, se observa que los *bounding boxes* y las etiquetas generadas por este modelo son coherentes y precisas, identificando correctamente todos los elementos de la imagen. Por tanto, este modelo muestra una robustez comparable a la del entrenamiento 30, sin errores evidentes en la detección de objetos.

Figura 4- 16. Resultado detección objetos usando modelo entrenamiento 21 e imagen Figura 4- 13. a).

En resumen, los resultados mostrados en las Figuras: Figura 4- 14, Figura 4- 15 y Figura 4- 16 revelan que los modelos del entrenamiento 30 y 21 ofrecen una precisión superior en la detección de objetos frente al modelo del entrenamiento 2, especialmente en lo que respecta a la identificación correcta de personas y la delimitación precisa de los objetos. Cabe recordar que los entrenamientos 30 y 21 se han llevado a cabo con imágenes con distorsión (ojo de pez), por el contrario, en el entrenamiento 2, únicamente se ha utilizado imágenes sin distorsión. Sin embargo, entre los modelos 30 y 21, no se observa una diferencia visual significativa que permita afirmar que uno es claramente superior al otro en este entorno de imagen en particular.

A continuación, se analiza la imagen correspondiente a la Figura 4- 13 b), utilizando los mismos tres modelos, que para el caso de la imagen de la Figura 4- 13 a).

La Figura 4- 17, ilustra el resultado final de identificación de objetos al emplear el modelo del entrenamiento 2. En la figura se vislumbra que se detectan varios objetos, incluidos carteles de extintores, cartel de aula, alarma y su respectivo cartel, además de otros objetos como el cartel de alto el paso y el cartel de prohibido ascensor. Aunque estos objetos han sido identificados correctamente, el modelo presenta limitaciones al no detectar algunos elementos

situados en el lado derecho de la imagen, donde los objetos tienen menor nivel de detalle o no resaltan con suficiente claridad. A pesar de este defecto, el modelo muestra un rendimiento aceptable, considerando que el modelo se obtuvo entrenando la red con imágenes sin distorsión.

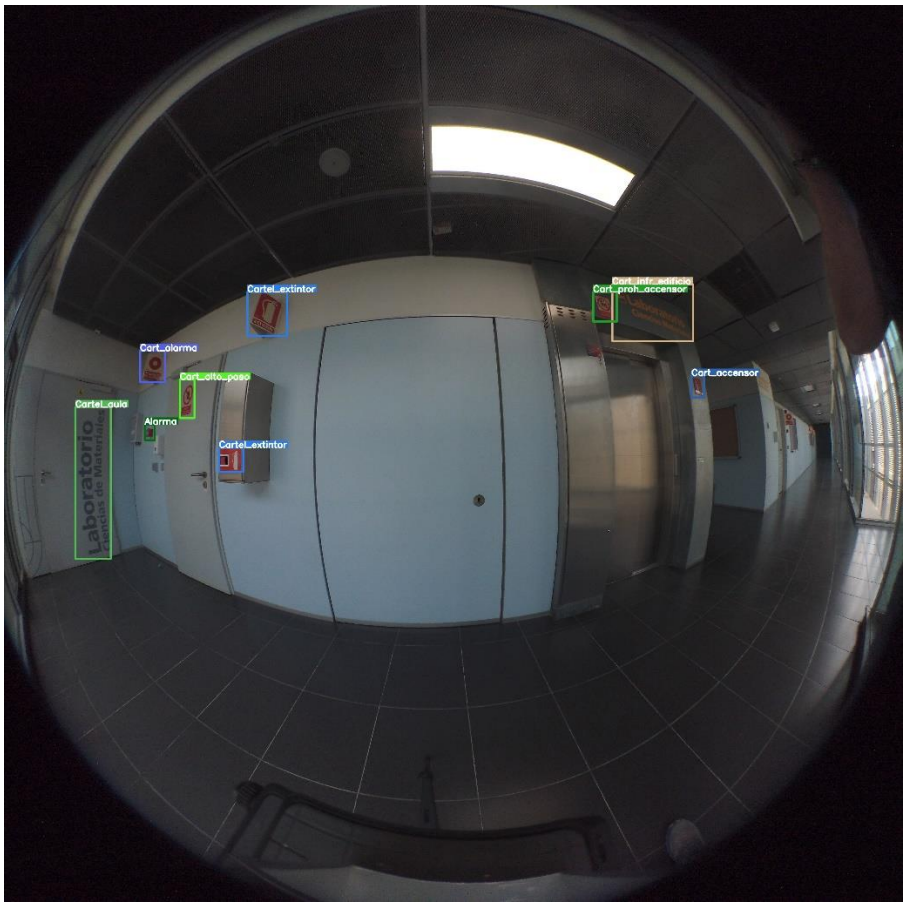


Figura 4- 17. Resultado detección objetos usando modelo entrenamiento 2 e imagen Figura 4- 13. b).

El modelo del entrenamiento 30, cuyos resultados se presentan en la Figura 4- 18, logra detectar correctamente todos los objetos mencionados en el párrafo anterior. Se aprecia que, a diferencia del modelo 2, este modelo es capaz de identificar objetos distorsionados situados en el lado derecho de la imagen, como el cartel de información del edificio y el cartel de alarma, tal como se muestra en la figura mencionada. Aun así, el modelo no logra identificar la alarma situada en el lado izquierdo de la imagen, que sí es detectada usando los otros modelos. Este comportamiento indica que, aunque mejora la detección en áreas con distorsión, el modelo aún presenta ciertas limitaciones.

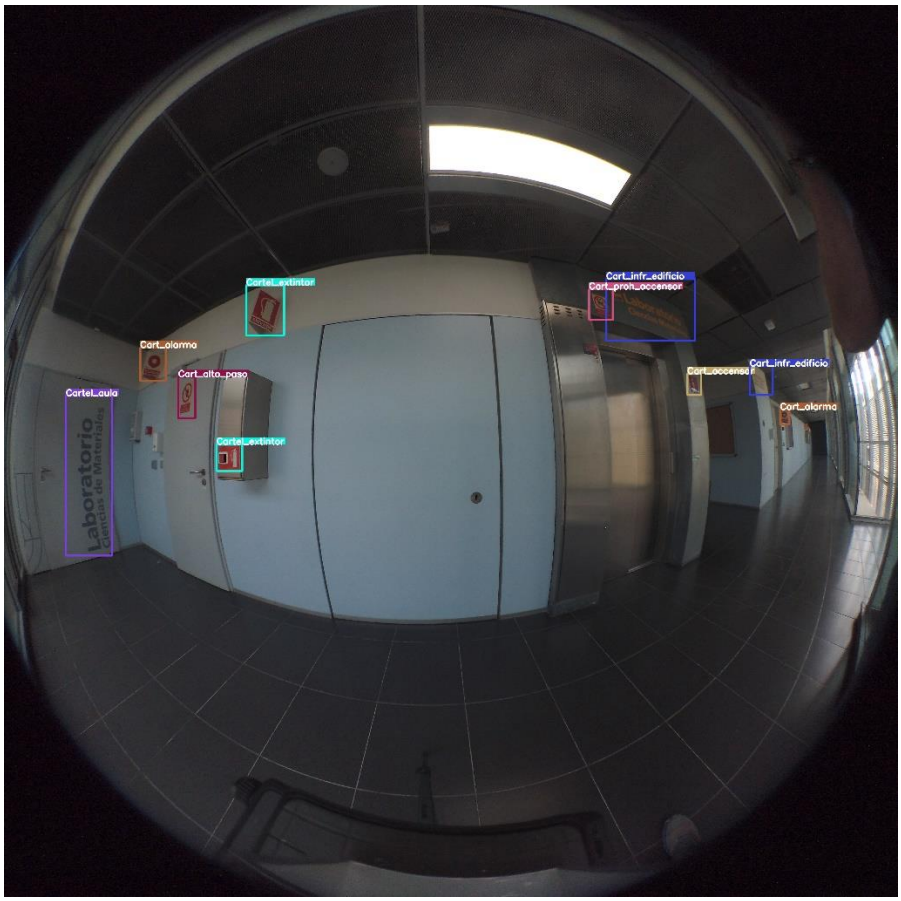


Figura 4- 18. Resultado detección objetos usando modelo entrenamiento 30 e imagen Figura 4- 13. b).

Finalmente, en la Figura 4- 19 se presenta el resultado obtenido al aplicar el modelo generado en el entrenamiento número 21. En esta imagen se identifican de manera correcta dos señales de alarma, dos carteles de información del edificio, dos señales de extintor, una señal de ascensor, una señal de prohibición de uso de ascensor, una alarma, una señal de alto el paso y un cartel de aula. Al comparar este modelo con los correspondientes a los entrenamientos 2 y 30, se observa claramente que el modelo 21 permite detectar un mayor número de objetos de manera precisa en la imagen.

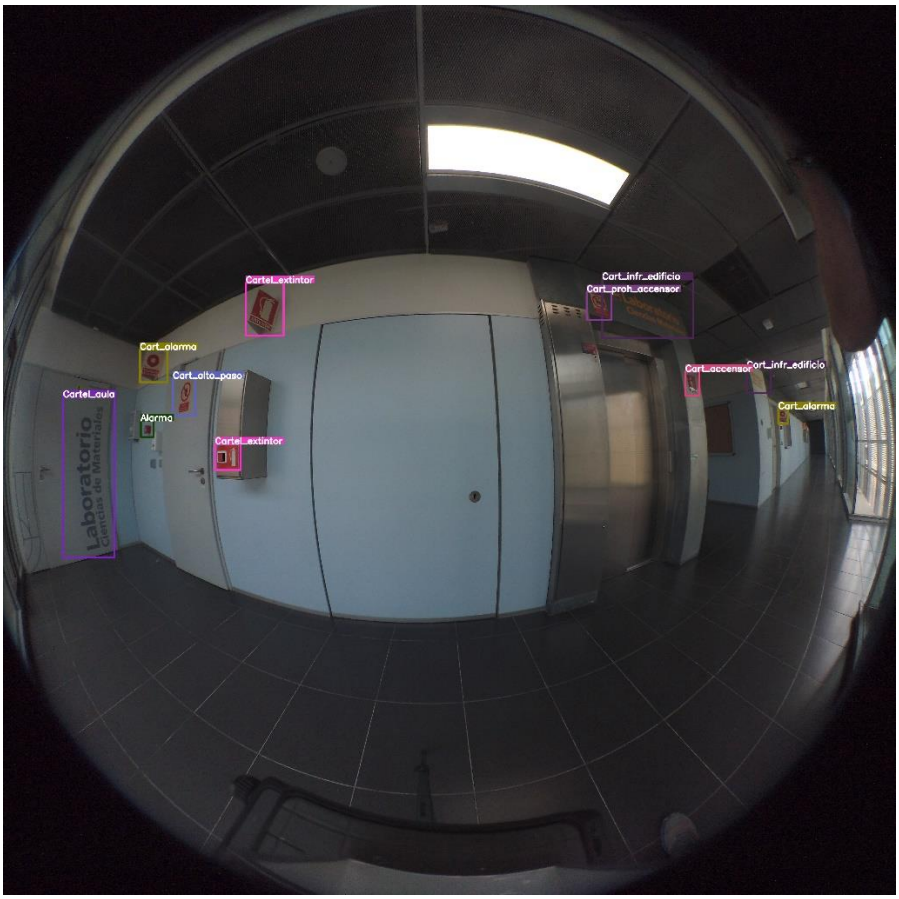


Figura 4- 19. Resultado detección objetos usando modelo entrenamiento 21 e imagen Figura 4- 13. b).

En conclusión, tras analizar los resultados visualizados en las Figuras: Figura 4- 14, Figura 4- 15, Figura 4- 16, Figura 4- 17, Figura 4- 18 y Figura 4- 19, se denota que el peor modelo es el obtenido entrenando solo con imágenes sin distorsión, y que la diferencia entre los otros dos modelos no es tan notable. Pudiendo determinar que, aunque los tres modelos logran detectar una amplia variedad de objetos en imágenes capturadas con lente de ojo de pez, el modelo del entrenamiento 21 es el más eficaz en términos de identificación y capacidad para manejar la distorsión. Los resultados obtenidos usando ambas imágenes, expuestas en la Figura 4- 13, evidencian que este modelo supera a los del entrenamiento 2 y 30, especialmente en la detección de objetos en imágenes de ojo de pez. Por tanto, el modelo del entrenamiento número 21 se recomienda como la mejor opción de los tres para la detección de objetos en imágenes distorsionadas por lentes de ojo de pez.

CAPÍTULO 5. CONCLUSIÓN

El presente trabajo ha abordado la evaluación de la red YOLOv8 aplicada a la detección de objetos en el interior de edificios, con un enfoque particular por la utilización de imágenes con distorsión capturadas con una cámara con dos lentes de ojo de pez. El objetivo principal de este trabajo es detectar e identificar correctamente aquellos objetos/personas cuando un robot móvil con una cámara ojo de pez a bordo navegue por un entorno interior público, como son los edificios de una universidad.

Inicialmente, se probó la capacidad de la red YOLOv8 sin entrenar, con el objetivo de determinar si era capaz de identificar los objetos presentes en las imágenes relevantes para este trabajo con distorsión y sin distorsión. Los resultados mostraron que la red no fue capaz de detectar correctamente la mayoría de estos objetos en ninguno de los escenarios evaluados. Esto supuso la creación de un *dataset* propio, uno de los principales aportes del estudio, compuesto por imágenes del interior de tres edificios de la Universidad Miguel Hernández. Este *dataset* incluye tanto imágenes sin distorsión como imágenes con distorsión, capturadas en diferentes condiciones de iluminación y perspectiva.

Posteriormente, se procedió al entrenamiento de la red neuronal utilizando los *datasets* creados. A lo largo del proceso experimental, se evaluaron distintas configuraciones del modelo, variando parámetros como el número de épocas, el tamaño de *batch* y el tamaño de entrada de las imágenes. Los resultados obtenidos revelaron que las configuraciones con un tamaño de entrada de imagen de 1280 y 1920 píxeles y un número de 300 épocas lograron un mejor rendimiento, alcanzando valores elevados de precisión y *recall*, con un F1 score que llegó al 87 % y un mAP@0.5 superior al 90 %. Estos resultados corroboran que el modelo YOLOv8 es eficaz para la detección de objetos, incluso en condiciones adversas como aquellas producidas por la distorsión de las lentes ojo de pez.

A pesar de los logros obtenidos, este estudio presenta limitaciones que merecen ser abordadas en futuras investigaciones. En primer lugar, la distorsión de las imágenes capturadas con lentes ojo de pez sigue representando un reto considerable, observándose algunos errores de detección, como la duplicación de objetos o la identificación incorrecta de elementos similares. Aunque, quizás una forma fácil de subsanar estos errores pueda ser ampliando el *dataset*.

Otra de las limitaciones es la necesidad de recursos computacionales avanzados, como GPUs. Asimismo, sería recomendable explorar otras arquitecturas de redes neuronales o combinaciones de algoritmos, podría llevar a una mayor robustez en la detección de objetos en imágenes con estas características.

Además, sería interesante explorar en futuros trabajos, la implementación del modelo YOLOv8 en un robot móvil capaz de capturar imágenes en tiempo real para identificar objetos y generar mapas que le permitan navegar de forma autónoma utilizando técnicas de SLAM. Además, sería interesante optimizar el rendimiento del modelo en condiciones de iluminación adversas mediante el

desarrollo de técnicas de preprocesamiento de imágenes, así como mejorar la detección en imágenes distorsionadas (como las capturadas por lentes ojo de pez) ajustando la arquitectura del modelo para adaptarse mejor a dichas condiciones ópticas.

BIBLIOGRAFÍA

[1] S. G. Tzafestas, "*Mobile robots: general concepts*.", Introduction to Mobile Robot Control, 2014.

[2] G. Cook y F. Zhang, *Mobile robots: navigation, control and sensing, surface robots and AUVs*, Second edition. Hoboken, New Jersey: Wiley, 2020. isbn: 9781119534785.

[3] T. T. Mac, C. Copot, D. T. Tran, y R. De Keyser, "Heuristic approaches in robot path planning: A survey", *Robotics and Autonomous Systems*, vol. 86, pp. 13-28, Dec. 2016, doi: 10.1016/j.robot.2016.08.001.

[4] R. Kala, *"Chapter2: Localization and mapping",* en Elsevier Science, 2023, págs. 635-663, isbn: 978-0-443-189098-1.

[5] S. Suzuki y R. Suda, "A vision system with wide field of view and collision alarms for teleoperation of mobile robots", *Robomech J*, vol. 1, no. 1, p. 8, Dec. 2014, doi: 10.1186/s40648-014-0008-5.

[6] D. Chatziparaschis, M. G. Lagoudakis, y P. Partsinevelos, "Aerial and ground robot collaboration for autonomous mapping in search and rescue missions", *Drones*, vol. 4, no. 4, p. 79, Dec. 2020, doi: 10.3390/drones4040079.

[7] M. P. Manuel, M. Faied, y M. Krishnan, "A Novel LoRa LPWAN-Based Communication Architecture for Search & Rescue Missions", *IEEE Access*, vol. 10, pp. 57596-57607, 2022, doi: 10.1109/ACCESS.2022.3178437.

[8] M. S. S. Tanjim, M. I. Rizvi, y A. N. Oishi, "Impact Analysis of Body Materials for Robo-Res 2.0 & 3.0: Part of Humanoid Rescue Robot", in *2021 IEEE International Conference on Robotics, Automation, Artificial-Intelligence and Internet-of-Things (RAAICON)*, Dhaka, Bangladesh: IEEE, Dec. 2021, pp. 94-97. doi: 10.1109/RAAICON54709.2021.9929845.

[9] Y. Zhang, J. Huang y L. Han, *Research status of planetary surface mobile exploration robots: Review*, Jan. de 2021. doi: 10.7527/S1000-6893.2020.23909.

[10] J. Arzberger, J. Zevering, F. Arzberger, y A. Nüchter, "Design and evaluation of an UI for astronauts to control mobile robots on planetary surfaces", in *2024 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Paredes de Coura, Portugal: IEEE, May 2024, pp. 74-81. doi: 10.1109/ICARSC61747.2024.10535929.

[11] C. Papachristos, S. Khattak, F. Mascarich, y K. Alexis, "Autonomous Navigation and Mapping in Underground Mines Using Aerial Robots", in *2019 IEEE Aerospace Conference*, Big Sky, MT, USA: IEEE, Mar. 2019, pp. 1-8. doi: 10.1109/AERO.2019.8741532.

[12] C. Tatsch, J. A. Bredu, D. Covell, I. B. Tulu, y Y. Gu, "Rhino: An Autonomous Robot for Mapping Underground Mine Environments", in *2023 IEEE/ASME*

International Conference on Advanced Intelligent Mechatronics (AIM), Seattle, WA, USA: IEEE, Jun. 2023, pp. 1166-1173. doi: 10.1109/AIM46323.2023.10196202.

[13] D. Lee, G. Kang, B. Kim, y D. H. Shim, "Assistive Delivery Robot Application for Real-World Postal Services", *IEEE Access*, vol. 9, pp. 141981-141998, 2021, doi: 10.1109/ACCESS.2021.3120618.

[14] J. Lee *et al.*, "ODS-Bot: Mobile Robot Navigation for Outdoor Delivery Services", *IEEE Access*, vol. 10, pp. 107250-107258, 2022, doi: 10.1109/ACCESS.2022.3212768.

[15] T. Tanioka, "Nursing and Rehabilitative Care of the Elderly Using Humanoid Robots", *J. Med. Invest.*, vol. 66, no. 1.2, pp. 19-23, Feb. 2019, doi: 10.2152/jmi.66.19.

[16] Y. Seo, E. Lee, S. Kwon, y W.-K. Song, "Real-time Virtual Coach using LSTM for Assisting Physical Therapists with End-effector-based Robot-assisted Gait Training", in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 4191-4196. doi: 10.1109/IROS45743.2020.9341523.

[17] F. Tanaka, K. Isshiki, F. Takahashi, M. Uekusa, R. Sei, y K. Hayashi, "Pepper learns together with children: Development of an educational application", en *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, Seoul, South Korea: IEEE, Nov. 2015, pp. 270-275. doi: 10.1109/HUMANOIDS.2015.7363546.

[18] K. Blöcher y R. Alt, "AI and robotics in the European restaurant sector: Assessing potentials for process innovation in a high-contact service industry", *Electron Markets*, vol. 31, no. 3, pp. 529-551, Sep. 2021, doi: 10.1007/s12525-020-00443-2.

[19] J. M. Garcia-Haro, E. D. Oña, J. Hernandez-Vicen, S. Martinez, y C. Balaguer, "Service Robots in Catering Applications: A Review and Future Challenges", *Electronics*, vol. 10, no. 1, p. 47, Dec. 2020, doi: 10.3390/electronics10010047.

[20] J. Zhang, Y. Ou, G. Jiang, y Y. Zhou, "An approach to restaurant service robot SLAM", in *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Qingdao, China: IEEE, Dec. 2016, pp. 2122-2127. doi: 10.1109/ROBIO.2016.7866643.

[21] H. T. Tran *et al.*, "A novel design of a smart interactive guiding robot for busy airports", *International Journal on Smart Sensing and Intelligent Systems*, vol. 15, no. 1, p. 20220017, Jan. 2022, doi: 10.2478/ijssis-2022-0017.

[22] R. Triebel *et al.*, "SPENCER: A Socially Aware Service Robot for Passenger Guidance and Help in Busy Airports", in *Field and Service Robotics*, vol. 113, D. S. Wettergreen y T. D. Barfoot, Eds., Cham: Springer International Publishing, 2016, pp. 607-622. doi: 10.1007/978-3-319-27702-8_40.

[23] A. Xavier, K. Alisha John, M. M. Akshara, L. M. Nair, y V. Rajan, “Navitrolley -An AI Integrated Path Finding Robot”, in *2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS)*, Kochi, India: IEEE, Jun. 2022, pp. 1-5. doi: 10.1109/IC3SIS54991.2022.9885507.

[24] A. K. Bordoloi, Md. F. Islam, J. Zaman, N. Phukan, y N. M. Kakoty, “A Floor Cleaning Robot for Domestic Environments”, in *Proceedings of the Advances in Robotics*, New Delhi India: ACM, Jun. 2017, pp. 1-5. doi: 10.1145/3132446.3134883.

[25] M. Arjun, C. A. Oliver, y A. M. Joseph, “Robot Vacuum Cleaner Using SLAM and ROS”, in *2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS)*, Chikkaballapur, India: IEEE, Apr. 2024, pp. 1-5. doi: 10.1109/ICKECS61492.2024.10616915.

[26] K. Yovchev, D. Chikurtev, N. Chivarov, y M. Grueva, “An Intelligent Control System for Service Robots”, *IFAC-PapersOnLine*, vol. 52, no. 25, pp. 327-332, 2019, doi: 10.1016/j.ifacol.2019.12.544.

[27] A. Eirale, M. Martini, L. Tagliavini, D. Gandini, M. Chiaberge, y G. Quaglia, “Marvin: An Innovative Omni-Directional Robotic Assistant for Domestic Environments”, *Sensors*, vol. 22, no.14, p. 5261, Jul. 2022, doi: 10.3390/s22145261.

[28] F. Rubio, F. Valero, y C. Llopis-Albert, “A review of mobile robots: Concepts, methods, theoretical framework, and applications”, *International Journal of Advanced Robotic Systems*, vol. 16, no.2, p. 172988141983959, Mar. 2019, doi: 10.1177/1729881419839596.

[29] Md. A. K. Niloy *et al.*, “Critical Design and Control Issues of Indoor Autonomous Mobile Robots: A Review”, *IEEE Access*, vol. 9, pp. 35338-35370, 2021, doi: 10.1109/ACCESS.2021.3062557.

[30] C. Cadena *et al.*, “Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age”, *IEEE Trans. Robot.*, vol. 32,no.6, pp. 1309-1332, Dec.2016, doi: 10.1109/TRO.2016.2624754.

[31] X. Zhou y R. Huang, “A State-of-the-Art Review on SLAM”, in *Intelligent Robotics and Applications*, H. Liu et al., eds., Cham: Springer International Publishing, 2022, págs. 240-251, isbn: 978-3-031-13835-5.

[32] A. Gil, O. M. Mozos, M. Ballesta, y O. Reinoso, “A comparative evaluation of interest point detectors and local descriptors for visual SLAM”, *Machine Vision and Applications*, vol. 21, no.6, pp. 905-920, Oct. 2010, doi: 10.1007/s00138-009-0195-x.

[33] N. Adzhar, Y. Yusof, y M. A. Ahmad, “A Review on Autonomous Mobile Robot Path Planning Algorithms”, *Adv. sci. technol. eng. syst. j.*, vol. 5,no.3, pp. 236-240, 2020, doi: 10.25046/aj050330.

[34] J. R. Sánchez-Ibáñez, C. J. Pérez-del-Pulgar, y A. García-Cerezo, “Path Planning for Autonomous Mobile Robots: A Review”, *Sensors*, vol. 21,no.23, p. 7898, Nov.2021, doi: 10.3390/s21237898.

[35] A. Gil, Ó. Reinoso, M. Ballesta, M. Juliá, y L. Payá, “Estimation of Visual Maps with a Robot Network Equipped with Vision Sensors”, *Sensors*, vol. 10,no.5, pp. 5209-5232, May 2010, doi: 10.3390/s100505209.

[36] I. S. Ramírez, P. J. Bernalte Sánchez, M. Papaelias, y F. P. G. Márquez, “Autonomous Underwater Vehicles and Field of View in Underwater Operations”, *JMSE*, vol. 9, no.3, p. 277, Mar. 2021, doi: 10.3390/jmse9030277.

[37] W. Chen *et al.*, “SLAM Overview: From Single Sensor to Heterogeneous Fusion”, *Remote Sensing*, vol. 14, no.23, p. 6033, Nov.2022, doi: 10.3390/rs14236033.

[38] G. Jia *et al.*, “Visual-SLAM Classical Framework and Key Techniques: A Review”, *Sensors*, vol. 22, no.12, p. 4582, Jun. 2022, doi: 10.3390/s22124582.

[39] A. Tourani, H. Bavle, J. L. Sanchez-Lopez, y H. Voos, “Visual SLAM: What Are the Current Trends and What to Expect?”, *Sensors*, vol. 22, no.23, p. 9297, Nov.2022, doi: 10.3390/s22239297.

[40] Scaramuzza, D., Fraundorfer, F., “Visual Odometry Part I: The First 30 Years and Fundamentals”, *IEEE ROBOTICS & AUTOMATION MAGAZINE*, 2011.

[41] Y. Zhang, Y. Wu, K. Tong, H. Chen, y Y. Yuan, “Review of Visual Simultaneous Localization and Mapping Based on Deep Learning”, *Remote Sensing*, vol. 15, no.11, p. 2740, May 2023, doi: 10.3390/rs15112740.

[42] Davison, A., “Real-time simultaneous localisation and mapping with a single camera.”, In: *Ninth IEEE International Conference on Computer Vision*, 2003, Proceedings, 2003, pp. 1403–1410.

[43] Y. Berenguer, L. Payá, M. Ballesta, L. Miguel Jiménez, S. Cebollada, y Ó. Reinoso, “Algoritmo de SLAM utilizando apariencia global de imágenes omnidireccionales”, in *XXXVIII Jornadas de Automática: Gijón, 6, 7, y 8 de septiembre de 2017*, Universidade da Coruña. Servizo de Publicacións, Aug.2020, pp. 956-963. doi: 10.17979/spudc.9788497497749.0956.

[44] J. Crespo, J. C. Castillo, O. M. Mozos, y R. Barber, “Semantic Information for Robot Navigation: A Survey”, *Applied Sciences*, vol. 10, no.2, p. 497, Jan. 2020, doi: 10.3390/app10020497.

[45] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, y N. Bt. Ismail, “Review of visual odometry: types, approaches, challenges, and applications”, *SpringerPlus*, vol. 5, no. 1, p. 1897, Dec. 2016, doi: 10.1186/s40064-016-3573-7.

[46] S. F. R. Alves, J. M., H. Ferasoli, L. K. A. Rincon, and R. A. T. Yamasaki, “Conceptual Bases of Robot Navigation Modeling, Control and Applications,” in *Advances in Robot Navigation*, A. Barrera, Ed., InTech, 2011. doi: 10.5772/20955.

[47] D. Karimanzira, H. Renkewitz, D. Shea, and J. Albiez, “Object Detection in Sonar Images,” *Electronics*, vol. 9, no. 7, p. 1180, Jul. 2020, doi: 10.3390/electronics9071180.

[48] S. Shao, C. Liu, J. Cheng, and J. Liu, “A Method Based on YOLOv8 for Sonar Image Object Detection,” in *2023 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, ZHENGZHOU, China: IEEE, Nov. 2023, pp. 1–5. doi: 10.1109/ICSPCC59353.2023.10400285.

[49] S. Muhammad and G.-W. Kim, “Visual Object Detection Based LiDAR Point Cloud Classification,” in *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Busan, Korea (South): IEEE, Feb. 2020, pp. 438–440. doi: 10.1109/BigComp48618.2020.00-32.

[50] S. McCrae and A. Zakhor, “3d Object Detection For Autonomous Driving Using Temporal Lidar Data,” in *2020 IEEE International Conference on Image Processing (ICIP)*, Abu Dhabi, United Arab Emirates: IEEE, Oct. 2020, pp. 2661–2665. doi: 10.1109/ICIP40778.2020.9191134.

[51] R. Sim y G. Dudek, “Effective exploration strategies for the construction of visual maps”, in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, Las Vegas, NV, USA: IEEE, 2003, pp. 3224-3231. doi: 10.1109/IROS.2003.1249653.

[52] D. Murray y C. Jennings, “Stereo vision based mapping and navigation for mobile robots”, in *Proceedings of International Conference on Robotics and Automation*, Albuquerque, NM, USA: IEEE, 1997, pp. 1694-1699. doi: 10.1109/ROBOT.1997.614387.

[53] R Sim and J. J. Little, "Autonomous vision-based exploration and mapping using hybrid maps and Rao-Blackwellised particle filters," *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006, pp. 2082-2089, doi: 10.1109/IROS.2006.282485.

[54] N. Ayache y F. Lustman, “Trinocular stereo vision for robotics”, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, no.1, pp. 73-85, Jan. 1991, doi: 10.1109/34.67633.

[55] L. Payá, A. Gil, y O. Reinoso, “A State-of-the-Art Review on Mapping and Localization of Mobile Robots Using Omnidirectional Vision Sensors”, *Journal of Sensors*, vol. 2017, pp. 1-20, 2017, doi: 10.1155/2017/3497650.

[56] “APPEARANCE-BASED DENSE MAPS CREATION - Comparison of Compression Techniques with Panoramic Images”, in *Proceedings of the 6th International Conference on Informatics in Control, Automation and Robotics*, Milan, Italy: SciTePress - Science and and Technology Publications, 2009, pp. 250-255. doi: 10.5220/0002210502500255.

[57] N. Winters, J. Gaspar, G. Lacey, y J. Santos-Victor, “Omni-directional vision for robot navigation”, in *Proceedings IEEE Workshop on Omnidirectional*

Vision (Cat. No.PR00704), Hilton Head Island, SC, USA: IEEE Comput. Soc, 2000, pp. 21-28. doi: 10.1109/OMNVIS.2000.853799.

[58] "Omnidirectional Images Database". Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://arvc.umh.es/db/images/trajectory/

[59] J.-P. Tardif, Y. Pavlidis and K. Daniilidis, "Monocular visual odometry in urban environments using an omnidirectional camera," *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, 2008, pp. 2531-2538, doi: 10.1109/IROS.2008.4651205

[60] "Objetivo ojo de pez", *Wikipedia, la enciclopedia libre*. 26 de septiembre de 2024. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Objetivo_ojo_de_pez&oldid=162677707

[61] J. Zhou, D. Yang, T. Song, Y. Ye, X. Zhang, y Y. Song, "Improved YOLOv7 models based on modulated deformable convolution and swin transformer for object detection in fisheye images", *Image and Vision Computing*, vol. 144, p. 104966, Apr. 2024, doi: 10.1016/j.imavis.2024.104966.

[62] L. Yang, G. Hu, Y. Song, G. Li, y L. Xie, "Intelligent video analysis: A Pedestrian trajectory extraction method for the whole indoor space without blind areas", *Computer Vision and Image Understanding*, vol. 196, p. 102968, Jul. 2020, doi: 10.1016/j.cviu.2020.102968.

[63] C. Luo *et al.*, "Autonomous detection of damage to multiple steel surfaces from 360° panoramas using deep neural networks", *Computer aided Civil Eng*, vol. 36, no.12, pp. 1585-1599, Dec.2021, doi: 10.1111/mice.12686.

[64] P. Barmpoutis, T. Stathaki, K. Dimitropoulos, y N. Grammalidis, "Early Fire Detection Based on Aerial 360-Degree Sensors, Deep Convolution Neural Networks and Exploitation of Fire Dynamic Textures", *Remote Sensing*, vol. 12,no.19, p. 3177, Sep. 2020, doi: 10.3390/rs12193177.

[65] T. Bertel, M. Yuan, R. Lindroos, y C. Richardt, "OmniPhotos: casual 360° VR photography", *ACM Trans. Graph.*, vol. 39, no.6, pp. 1-12, Dec.2020, doi: 10.1145/3414685.3417770.

[66] Y. Zhou, L. Tian, C. Zhu, X. Jin, y Y. Sun, "Video Coding Optimization for Virtual Reality 360-Degree Source", *IEEE J. Sel. Top. Signal Process.*, vol. 14, no.1, pp. 118-129, Jan. 2020, doi: 10.1109/JSTSP.2019.2957952.

[67] V. Ravi Kumar *et al.*, "OmniDet: Surround View Cameras Based Multi-Task Visual Perception Network for Autonomous Driving", *IEEE Robot. Autom. Lett.*, vol. 6,no.2, pp. 2830-2837, Apr. 2021, doi: 10.1109/LRA.2021.3062324.

[68] Z. Cui, L. Heng, Y. C. Yeo, A. Geiger, M. Pollefeys, y T. Sattler, "Real-Time Dense Mapping for Self-Driving Vehicles using Fisheye Cameras", in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada: IEEE, May 2019, pp. 6087-6093. doi: 10.1109/ICRA.2019.8793884.

[69] C. Häne *et al.*, "3D visual perception for self-driving cars using a multi-camera system: Calibration, mapping, localization, and obstacle detection", *Image and Vision Computing*, vol. 68, pp. 14-27, Dec.2017, doi: 10.1016/j.imavis.2017.07.003.

[70] W. Gao, K. Wang, W. Ding, F. Gao, T. Qin, y S. Shen, "Autonomous aerial robot using dual-fisheye cameras", *Journal of Field Robotics*, vol. 37,no.4, pp. 497-514, Jun. 2020, doi: 10.1002/rob.21946.

[71] G. Billings y M. Johnson-Roberson, "SilhoNet-Fisheye: Adaptation of A ROI-Based Object Pose Estimation Network to Monocular Fisheye Images", *IEEE Robot. Autom. Lett.*, pp. 1-1, 2020, doi: 10.1109/LRA.2020.2994036.

[72] M. Roxas y T. Oishi, "Variational Fisheye Stereo", *IEEE Robot. Autom. Lett.*, vol. 5,no.2, pp. 1303-1310, Apr. 2020, doi: 10.1109/LRA.2020.2967657.

[73] H.-E. Benseddik, F. Morbidi, y G. Caron, "PanoraMIS: An ultra-wide field of view image dataset for vision-based robot-motion estimation", *The International Journal of Robotics Research*, vol. 39, no.9, pp. 1037-1051, Aug.2020, doi: 10.1177/0278364920915248.

[74] D. Yang, J. Zhou, T. Song, X. Zhang, y Y. Song, "PGDS-YOLOv8s: An Improved YOLOv8s Model for Object Detection in Fisheye Images", *Applied Sciences*, vol. 14, no.1, p. 44, Dec.2023, doi: 10.3390/app14010044.

[75] S. Joy, R. L. Paulraj, P. M, S. M, S. Goudar, y R. S, "A Raspberry Pi based Smart Security Patrol Robot", en *2023 7th International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India: IEEE, feb. 2023, pp. 1140-1145. doi: 10.1109/ICCMC56507.2023.10083908.

[76] G. Marcu, I. Lin, B. Williams, L. P. Robert, y F. Schaub, "Would I Feel More Secure With a Robot?": Understanding Perceptions of Security Robots in Public Spaces", *Proc. ACM Hum.-Comput. Interact.*, vol. 7, no.CSCW2, pp. 1-34, Sep. 2023, doi: 10.1145/3610171.

[77] R. M. Stock y M. Merkle, "A service Robot Acceptance Model: User acceptance of humanoid robots during service encounters", in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Kona, HI: IEEE, Mar. 2017, pp. 339-344. doi: 10.1109/PERCOMW.2017.7917585.

[78] T. Yang *et al.*, "Hybrid Camera Array-Based UAV Auto-Landing on Moving UGV in GPS-Denied Environment", *Remote Sensing*, vol. 10, no.11, p. 1829, Nov.2018, doi: 10.3390/rs10111829.

[79] E. Arkin, N. Yadikar, X. Xu, A. Aysa, y K. Ubul, "A survey: object detection methods from CNN to transformer", *Multimed Tools Appl*, vol. 82, no.14, pp. 21353-21383, Jun. 2023, doi: 10.1007/s11042-022-13801-3.

[80] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, y B. Lee, "A survey of modern deep learning based object detection models", *Digital Signal Processing*, vol. 126, p. 103514, Jun. 2022, doi: 10.1016/j.dsp.2022.103514.

[81] L. Liu *et al.*, “Deep Learning for Generic Object Detection: A Survey”, *Int J Comput Vis*, vol. 128, no.2, pp. 261-318, feb. 2020, doi: 10.1007/s11263-019-01247-4.

[82] M. Afif, R. Ayachi, E. Pissaloux, Y. Said, y M. Atri, “Indoor objects detection and recognition for an ICT mobility assistance of visually impaired people”, *Multimed Tools Appl*, vol. 79, no.41-42, pp. 31645-31662, Nov.2020, doi: 10.1007/s11042-020-09662-3.

[83] A. Krizhevsky, I. Sutskever, y G. E. Hinton, “ImageNet classification with deep convolutional neural networks”, *Commun. ACM*, vol. 60, no.6, pp. 84-90, May 2017, doi: 10.1145/3065386.

[84] K. Simonyan y A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, 2014, *arXiv*. doi: 10.48550/ARXIV.1409.1556.

[85] C. Szegedy *et al.*, “Going deeper with convolutions”, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, Jun. 2015, pp. 1-9. doi: 10.1109/CVPR.2015.7298594.

[86] K. He, X. Zhang, S. Ren, y J. Sun, “Deep Residual Learning for Image Recognition”, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770-778. doi: 10.1109/CVPR.2016.90.

[87] R. Girshick, J. Donahue, T. Darrell, y J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”, en *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA: IEEE, Jun. 2014, pp. 580-587. doi: 10.1109/CVPR.2014.81.

[88] R. Girshick, “Fast R-CNN”, in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile: IEEE, Dec.2015, pp. 1440-1448. doi: 10.1109/ICCV.2015.169.

[89] S. Ren, K. He, R. Girshick, y J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no.6, pp. 1137-1149, Jun. 2017, doi: 10.1109/TPAMI.2016.2577031.

[90] K. He, G. Gkioxari, P. Dollar, and R. Girshick, “Mask R-CNN,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice: IEEE, Oct. 2017, pp. 2980–2988. doi: 10.1109/ICCV.2017.322.

[91] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 779-788. doi: 10.1109/CVPR.2016.91.

[92] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector”, en *Computer Vision – ECCV 2016*, vol. 9905, B. Leibe, J. Matas, N. Sebe, y M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 21-37. doi: 10.1007/978-3-319-46448-0_2.

[93] T.-Y. Lin, P. Goyal, R. Girshick, K. He, y P. Dollar, “Focal Loss for Dense Object Detection”, in *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice: IEEE, Oct. 2017, pp. 2999-3007. doi: 10.1109/ICCV.2017.324.

[94] M. Afif, R. Ayachi, Y. Said, E. Pissaloux, y M. Atri, “An efficient object detection system for indoor assistance navigation using deep learning techniques”, *Multimed Tools Appl*, vol. 81, no.12, pp. 16601-16618, May 2022, doi: 10.1007/s11042-022-12577-w.

[95] J. Terven, D.-M. Córdova-Esparza, y J.-A. Romero-González, “A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS”, *MAKE*, vol. 5, no.4, pp. 1680-1716, Nov.2023, doi: 10.3390/make5040083.

[96] J. Redmon y A. Farhadi, “YOLO9000: Better, Faster, Stronger”, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI: IEEE, Jul. 2017, pp. 6517-6525. doi: 10.1109/CVPR.2017.690.

[97] J. Redmon y A. Farhadi, “YOLOv3: An Incremental Improvement”, 8 de abril de 2018, *arXiv*: arXiv:1804.02767. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: http://arxiv.org/abs/1804.02767

[98] A. Bochkovskiy, C.-Y. Wang, y H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection”, 22 de abril de 2020, *arXiv*: arXiv:2004.10934. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: http://arxiv.org/abs/2004.10934

[99] Ultralytics, “Comprehensive Guide to Ultralytics YOLOv5”. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/yolov5

[100] “Train Custom Data”, GitHub. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

[101] G. Jocher, A. Chaurasia, y J. Qiu, *Ultralytics YOLO*. (enero de 2023). Python. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://github.com/ultralytics/ultralytics

[102] C. Li *et al.*, “YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications”, 2022, *arXiv*. doi: 10.48550/ARXIV.2209.02976.

[103] C.-Y. Wang, A. Bochkovskiy, y H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors”, 2022, *arXiv*. doi: 10.48550/ARXIV.2207.02696.

[104] Ultralytics, “YOLOv8”. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/models/yolov8/

[105] Garmin y G. L. or its subsidiaries, “VIRB 360”, Garmin. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://www.garmin.com/es-ES/p/562010

[106] Garmin, VIRB 360, especificaciones. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: www8.garmin.com/automotive/pdfs/VIRB360-specs.pdf

[107] Ultralytics, “Ultralytics YOLOv8 Modes”. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/models/yolov8/#supported-tasks-and-modes

[108] Ultralytics, “Ultralytics YOLOv8 Introduction”. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/modes/#introduction

[109] J. J. Cabrera, S. Cebollada, M. Ballesta, L. M. Jiménez, L. Payá, y Ó. Reinoso, “Entrenamiento, optimización y validación de una CNN para localización jerárquica mediante imágenes omnidireccionales.”, in *XLII JORNADAS DE AUTOMÁTICA: LIBRO DE ACTAS*, Servizo de Publicacións da UDC, 2021, pp. 640-647. doi: 10.17979/spudc.9788497498043.640.

[110] “Robot explorador Perseverence”. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://www.nationalgeographic.com.es/ciencia/objetivos-rover-perseverance-marte_16363

[111] “Robot Bellabot”. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://www.pudurobotics.com/es/products/bellabot

[112] “Robot Roomba”. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://www.irobot.es/es_ES/essential-series/Y0MASTER.html

[113] “Robot RB-WATCHER”. Accedido: 27 de septiembre de 2024. [En línea]. Disponible en: https://robotnik.eu/es/la-era-de-la-vigilancia-movil-el-potencial-de-los-robots-de-seguridad/

[114] H. Rashed *et al.*, “Generalized Object Detection on Fisheye Cameras for Autonomous Driving: Dataset, Representations and Baseline,” in *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Waikoloa, HI, USA: IEEE, Jan. 2021, pp. 2271–2279. doi: 10.1109/WACV48630.2021.00232.

[115] Z. Balgabekova, M. Alaran, and H. A. Varol, “A Data-Centric Approach for Object Recognition in Hemispherical Camera Images,” in *IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society*, Singapore, Singapore: IEEE, Oct. 2023, pp. 1–6. doi: 10.1109/IECON51785.2023.10311869.

ANEXO I

Código para visualizar los objetos detectados en una imagen.

 # Nombre del archivo: verificar_deteccion.py

```
from ultralytics import YOLO
import numpy as np
import cv2
import os
import matplotlib as plt
from random import randint

# Clase para gestionar los colores asignados a las etiquetas de detección.
class color:
  def __init__(self, classes=0, RGB=(255, 255, 0)):
    self.dic_colors = {}  # Diccionario para almacenar colores únicos para cada categoría.
    self.RGB = RGB

  def getColor(self, category=''):

    # Genera un color aleatorio para una categoría nueva.
    if category != '':
      if category not in self.dic_colors:
        self.dic_colors[category] = (randint(0, 255), randint(0, 255), randint(0, 255))

      self.RGB = self.dic_colors[category]     # Asigna el color correspondiente.



# Función para mostrar las detecciones en una imagen.

def displayDetections(img_display, items, show=False, name='',
font=cv2.FONT_HERSHEY_SIMPLEX, font_scale=2,
              font_thickness=4):

  bboxcolor = color()    # Objeto para gestionar colores de etiquetas.
  # for item in items:
  for i in range(items['labels'].__len__()):
    label = items['labels'][i]
    bbox = items['bboxes'][i]
    text_size, _ = cv2.getTextSize(label, font, font_scale, font_thickness)
    text_w, text_h = text_size
    x, y = (bbox[0][0], bbox[0][1] - text_h)

   # Ajusta la posición de la etiqueta para no exceeder los límites del bounding box.
```

```
      if x + text_w > bbox[1][0]:
        x_label = x + text_w
      else:
        x_label = bbox[1][0]

      bboxcolor.getColor(label)   # Asigna un color a cada etiqueta de detección.
      # Dibuja el rectángulo de detección y la etiqueta en la imagen (Bounding box).
      img_display = cv2.rectangle(img_display, bbox[0], bbox[1], bboxcolor.RGB,
thickness=5)
      # Display label
      img_display = cv2.rectangle(img_display, (x, y), (x_label, y + text_h),
bboxcolor.RGB, -1)
      img_display = cv2.putText(img_display, label, (x, y + text_h + font_scale - 1),
font, font_scale,
                        (255, 255, 255),
                        font_thickness)
  if show:
      plt.imshow(cv2.cvtColor(img_display, cv2.COLOR_BGR2RGB))  # Muestra la
imagen con detecciones.
      plt.show()
  if name != '':
      #if not os.path.exists(os.path.dirname(name)):
       #   os.makedirs(os.path.dirname(name))
      cv2.imwrite(name, img_display)  # Guarda la imagen con detecciones si se
proporciona un nombre.


# (Load a model), carga un modelo preentrenado YOLO para detección de objetos.
# model = YOLO("yolov8n.yaml")  # construir un nuevo modelo desde cero.
model = YOLO("best.pt")  # cargar un modelo preentrenado (recomendado para el
entrenamiento)
# "best.pt contiene los pesos entrenados.


# Imagen de entrada.
img='eb2d496b-V7330957.JPG'
results = model(img)   # Realiza las detecciones en la imagen.
labels=[]
bboxes=[]
scores=[]
objects={}


# Extrae y organiza la información de detección.
for result in results:
      boxes = result.boxes
      for box in boxes:
```

```
        cls = result.names[int(box.cls)]
        bbox = np.array(box.xyxy).squeeze()
        left, top, right, bottom = bbox
        conf = box.conf.tolist()
        labels.append(cls)
        bboxes.append([(int(left), int(top)), (int(right), int(bottom))])
        scores.append(conf[0])

# Estructura de los resultados de las detecciones para su visualización.
objects['labels'] = labels
objects['bboxes'] = bboxes
objects['scores'] = scores


# Carga la imagen original y muestra las detecciones de ella.
img2 =cv2.imread(img)
displayDetections(img2, objects, font_scale=1,
            name='objetos.jpg')

# Muestra información sobre el modelo YOLO (Display model information).
model.info()
```

ANEXO II

AJUSTES PERMITIDOS EN LA RED YOLOv8 PARA ENTRENAMIENTO

ARGUMENTO	VALOR POR DEFECTO	DESCRIPCIÓN
model	ninguno	Especifica el archivo del modelo para el entrenamiento permitiendo la inclusión de una ruta hacia un modelo preentrenado en formato ".pt" o un archivo de configuración ".yaml". Esto resulta fundamental para definir la estructura del modelo o para inicializar los pesos.
data	Ninguno	Ruta al archivo de configuración del conjunto de datos. Este archivo incluye parámetros específicos del conjunto de datos, como las rutas para los datos de entrenamiento (*train*) y validación (val), los nombres de las clases y el número de clases.
epoch	100	Número total de épocas (*epoch*) de entrenamiento. Cada época corresponde a una pasada completa por todo el conjunto de datos. Modificar este valor puede influir en la duración del entrenamiento y el rendimiento del modelo.
time	Ninguno	Tiempo máximo de entrenamiento en horas. Al establecer este parámetro, se sobrescribe el argumento de épocas, permitiendo que el entrenamiento se detenga automáticamente una vez transcurrido el tiempo especificado. Esto resulta útil para escenarios de entrenamientos con limitaciones temporales.
patience	100	Cantidad de épocas (*epoch*) que se deben esperar sin mejoras en las métricas de validación antes de detener el entrenamiento. Esto ayuda a evitar el sobreajuste al parar el entrenamiento cuando el rendimiento se mantiene constante.
batch	16	Tamaño de lote (*batch*), configurable de tres maneras: como un valor entero específico (por ejemplo, *batch*=16), en modo automático para utilizar el 60% de la memoria de la GPU (*batch*=-1), o en modo automático con una fracción de memoria determinada (*batch*=0.70).

ARGUMENTO	VALOR POR DEFECTO	DESCRIPCIÓN
imgsz	640	Tamaño de imagen deseado para el entrenamiento. Las imágenes se ajustan a estas dimensiones antes de ser ingresadas al modelo. Este tamaño impacta tanto en la precisión del modelo como en la complejidad computacional.
save	True	Facilita el guardado de los puntos de control del entrenamiento y los pesos finales del modelo. Esto es útil para reanudar el entrenamiento en el futuro o para implementar el modelo.
save_period	-1	Frecuencia con la que se guardan los puntos de control del modelo, especificada en *epoch*. Un valor de -1 desactiva esta función. Es útil para guardar modelos intermedios durante sesiones de entrenamiento prolongados.
cache	False	Habilita el almacenamiento en caché de las imágenes del conjunto de datos en la memoria (True/ram), en el disco (disk), o desactivado (False). Esto puede acelerar el entrenamiento al reducir el acceso al disco, aunque con un mayor consumo de memoria.
device	Ninguno	Define el dispositivo o dispositivos de cálculo para el entrenamiento: una sola GPU (*device*=0), varias GPUs (*device*=0,1), CPU (*device*=cpu) o MPS para Apple Silicon (*device*=mps).
workers	8	Cantidad de subprocesos para la carga de datos (por RANK si el entrenamiento es Multi-GPU). Influye en la velocidad de preprocesamiento de datos y alimentación del modelo, especialmente útil en configuraciones Multi-GPU.
project	Ninguno	Nombre del directorio del proyecto destinado a almacenar los resultados del entrenamiento. Facilita la organización y el almacenamiento de los distintos experimentos.
name	Ninguno	Nombre del entrenamiento. Este nombre se emplea para generar un subdirectorio dentro de la carpeta del proyecto, que alberga los registros y resultados del entrenamiento.

ARGUMENTO	VALOR POR DEFECTO	DESCRIPCIÓN
exist_ok	False	Si es True, habilita la sobreescritura de un directorio de proyecto con nombre ya existente. Esto es útil para realizar experimentos iterativos sin tener que eliminar manualmente los resultados anteriores.
pretrained	True	Define si el entrenamiento comienza a partir de un modelo reentrenado. Puede ser un valor booleano o una ruta que apunte a un modelo específico desde el cual cargar los pesos. Esto optimiza la eficiencia del entrenamiento y el rendimiento del modelo.
optimizer	'auto'	Selección del optimizador para el entrenamiento. Se puede elegir entre opciones como SGD, Adam, AdamW, NAdam, RAdam, RMSProp, entre otros o seleccionar 'auto' para que el sistema elija automáticamente según la configuración del modelo. Esto influye en la rapidez de convergencia y en la estabilidad del entrenamiento.
verbose	False	Habilita la salida detallada durante el entrenamiento, generando registros exhaustivos y actualizaciones del progreso. Esta opción es valiosa para la depuración y para el monitoreo minucioso del proceso de entrenamiento.
seed	0	Define la semilla aleatoria para el entrenamiento, garantizando la reproducibilidad de los resultados en ejecuciones con las mismas configuraciones.
deterministic	True	Fuerza el uso de algoritmos deterministas, asegurando la reproducibilidad de los resultados. Sin embargo, esto puede impactar negativamente en el rendimiento y la velocidad al restringir el uso de algoritmos no determinísticos.
single_cls	False	Considera todas las clases de los conjuntos de datos multiclase como una sola clase durante el entrenamiento. Esta configuración es útil para tareas de clasificación binaria o cuando el enfoque está en la detección de la presencia de objetos más que en la clasificación específica.
rect	False	Habilita el entrenamiento rectangular, optimizando la composición del lote para minimizar el relleno. Esto puede mejorar la eficiencia y la velocidad del entrenamiento, aunque podría tener un impacto en la precisión del modelo.

ARGUMENTO	VALOR POR DEFECTO	DESCRIPCIÓN
Ccs_lr	False	Emplea un programador de tasa de aprendizaje basado en una función coseno, que ajusta la tasa de aprendizaje según la curva coseno a lo largo de las *epoch*. Esto facilita la gestión de la tasa de aprendizaje para lograr una mejor convergencia.
close_mosaic	10	Desactiva el aumento de datos en mosaico en las últimas N *epoch* para estabilizar el entrenamiento antes de su finalización. El valor 0 desactiva esta función.
resume	False	Reanuda el entrenamiento a partir del último punto de control guardado. Cargando automáticamente los pesos del modelo, el estado de optimizador y el recuento de *epoch*, permitiendo continuar el entrenamiento sin inconvenientes.
amp	True	Habilita el entrenamiento con Precisión Mixta Automática (AMP), lo que reduce el consumo de memoria y puede acelerar el proceso de entrenamiento con un impacto mínimo en la precisión.
fraction	1.0	Define la fracción del conjunto de datos que se empleará para el entrenamiento. Esto permite realizar el entrenamiento en un subconjunto del conjunto de datos total, lo cual es útil para experimentos o cuando los recursos son limitados.
profile	False	Habilita el perfilado de las velocidades de ONNX y TensorRT durante el entrenamiento, lo cual es útil para optimizar la implementación del modelo.
freeze	Ninguno	Congela las primeras N capas del modelo o las capas indicadas por su índice, disminuyendo el número de parámetros que se pueden entrenar. Esto es útil para el ajuste fino o el aprendizaje por transferencia.
lr0	0.01	Tasa de aprendizaje inicial. El ajuste de este valor es esencial para el proceso de optimización, ya que afecta la velocidad con la que se actualizan los pesos del modelo.
lrf	0.01	Tasa de aprendizaje final como fracción de la tasa inicial = (lr0*lrf), que se usa junto con los programadores para ajustar la tasa de aprendizaje a lo largo del tiempo.
momentum	0.937	Factor de impulso para SGD o beta1para optimizadores Adam, que afecta a la influencia de los gradientes pasados en la actualización actual.

ARGUMENTO	VALOR POR DEFECTO	DESCRIPCIÓN
weight_decay	0.0005	Término de regularización L2, que impone una penalización a los pesos grandes para prevenir el sobreajuste.
warmup_epochs	3.0	Número de *epoch* dedicadas al calentamiento de la tasa de aprendizaje, durante las cuales la tasa se incrementa progresivamente desde un valor bajo hasta alcanzar la tasa de aprendizaje inicial, con el fin de estabilizar el entrenamiento desde el inicio.
warmup_momentum	0.8	Valor inicial del momento para la fase de calentamiento, que se ajusta gradualmente al momento fijado durante el periodo de calentamiento.
warmup_bias_lr	0.1	Tasa de aprendizaje para los parámetros de sesgo durante la fase de calentamiento, que contribuye a estabilizar el entrenamiento del modelo en las primeras *epoch*.
box	7.5	Peso del componente de pérdida de caja (*box loss*) en la función de pérdida, que determina el grado de importancia asignado a la precisión en la predicción de las coordenadas de la caja delimitadora.
cls	0.5	Peso de la pérdida de clasificación (*classification loss*) en la función de pérdida total (*total loss*), que determina la relevancia de obtener una predicción correcta de la clase en comparación con otros componentes.
dfl	1.5	Peso de la pérdida focal (*focal loss*) de distribución, empleado en ciertas variantes de YOLO para mejorar la precisión de la clasificación.
pose	12.0	Peso de la pérdida de pose (*pose loss*) en los modelos diseñados para la estimación de la pose, que determina la relevancia de la precisión en la predicción de los puntos clave de la pose.
kobj	2.0	El peso de la pérdida de objetividad de los puntos clave en modelos de estimación de la pose, que ajusta el equilibrio entre la confianza en la detección y la precisión en la estimación de la pose.
label_smoothing	0.0	Aplica suavizado de etiquetas (*labels*), que transforma las etiquetas duras en una combinación de la etiqueta objetivo y una distribución uniforme entre las etiquetas. Esto puede mejorar la capacidad de generalización del modelo.
nbs	64	Tamaño de lote nominal para la normalización de la pérdida.

ARGUMENTO	VALOR POR DEFECTO	DESCRIPCIÓN
overlap_mask	True	Define si las máscaras de segmentación deben superponerse durante el entrenamiento, aplicable a tareas de segmentación de instancias.
mask_ratio	4	Proporción de reducción de la muestra para las máscaras de segmentación, que influye en la resolución de las máscaras empleadas durante el entrenamiento.
dropout	0.0	Tasa de abandono para la regularización en tareas de clasificación, que ayuda a prevenir el sobreajuste al desactivar aleatoriamente unidades durante el entrenamiento.
val	True	Activa la validación durante el entrenamiento, lo que posibilita la evaluación periódica del rendimiento del modelo en un conjunto de datos separado.
plots	False	Crea y almacena gráficos de métricas de entrenamiento y validación, junto con ejemplos de predicción, ofreciendo una representación visual del rendimiento del modelo y del avance del aprendizaje.

ANEXO III

PARÁMETROS PARA AUMENTO DE DATOS

ARGUMENTO	TIPO	VALOR POR DEFECTO	RANGO	DESCRIPCIÓN
hsv_h	Numérico (*float*)	0.015	0.0 - 1.0	Modifica el tono de la imagen en una porción de la rueda de color, agregando variabilidad cromática. Esto contribuye a que el modelo generalice mejor ante distintas condiciones de iluminación.
hsv_s	Numérico	0.7	0.0 – 1.0	Modifica la saturación de la imagen en un determinado porcentaje, impactando la intensidad de los colores. Es útil para replicar distintas condiciones ambientales.
hsv_v	Numérico	0.4	0.0 – 1.0	Ajusta el valor de brillo de la imagen en un cierto porcentaje, mejorando la capacidad del modelo para adaptarse a diferentes condiciones de iluminación.
degrees	Numérico	0.0	-180 - +180	Gira la imagen de manera aleatoria dentro de un rango de grados determinado, lo que mejora la habilidad del modelo para identificar objetos en distintas orientaciones.
translate	Numérico	0.1	0.0 – 1.0	Desplaza la imagen en dirección horizontal y vertical por una fracción de su tamaño, facilitando el aprendizaje del modelo para identificar objetos parcialmente visibles.
scale	Numérico	0.5	≥ 0.0	Escala la imagen mediante un factor de amplificación, simulando objetos ubicados a distintas distancias de la cámara.
shear	Numérico	0.0	-180 - +180	Recorta la imagen en un ángulo determinado, replicando el efecto de observar objetos desde diversas perspectivas.

ARGUMENTO	TIPO	VALOR POR DEFECTO	RANGO	DESCRIPCIÓN
perspective	Numérico	0.0	0.0 – 0.001	Aplica una transformación de perspectiva aleatoria a la imagen, lo que mejora la habilidad del modelo para interpretar objetos en un entorno tridimensional.
flipud	Numérico	0.0	0.0 – 1.0	Gira la imagen con la probabilidad indicada, incrementado la variabilidad de los datos sin modificar las características del objeto.
fliplr	Numérico	0.5	0.0 – 1.0	Voltea la imagen horizontalmente con la probabilidad indicada, lo que es útil para enseñar al modelo a reconocer objetos simétricos y para ampliar la diversidad del conjunto de datos.
bgr	Numérico	0.0	0.0 – 1.0	Cambia el orden de los canales de la imagen de RGB a BGR con la probabilidad indicada, lo cual ayuda a mejorar la robustez del modelo frente a una disposición incorrecta de los canales.
mosaic	Numérico	1.0	0.0 – 1.0	Une cuatro imágenes de entrenamiento en una sola, imitando diversas composiciones de escena e interacciones entre objetos. Esta técnica es altamente eficaz para la comprensión de escenas complejas.
mixup	Numérico	0.0	0.0 – 1.0	Fusiona dos imágenes y sus respectivas etiquetas para generar una imagen compuesta. Esto mejora la capacidad de generalización del modelo al introducir variabilidad visual y ruido en las etiquetas.
copy_paste	Numérico	0.0	0.0 – 1.0	Transfiere objetos de una imagen a otra, lo que es útil para incrementar el número de instancias de objetos y para enseñar al modelo a manejar la oclusión de objetos.

ARGUMENTO	TIPO	VALOR POR DEFECTO	RANGO	DESCRIPCIÓN
auto_augment	Texto (str)	randaugment		Aplica de manera automática una política de aumento predefinida (como RandAugment, AutoAugment o AugMix), optimizando las tareas de clasificación al diversificar las características visuales.
erasing	Numérico	0.4	0.0 – 0.9	Elimina aleatoriamente una porción de la imagen durante el entrenamiento de clasificación, incentivando al modelo a enfocarse en características menos evidentes para el reconocimiento.
crop_fraction	Numérico	1.0	0.1 – 1.0	Recorta la imagen de clasificación a una fracción de su tamaño para resaltar las características principales y ajustarse a las escalas de los objetos, minimizando las distracciones del fondo.