BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

RECONOCIMIENTO Y UBICACIÓN ESPACIAL DE OBJETOS EN UN LABORATORIO DE SÍNTESIS DE MATERIALES USANDO REDES NEURONALES CONVOLUCIONALES

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

ALBERTO ESTEBAN REYES PERALTA

Director de Tesis:

DR. DAVID EDUARDO PINTO AVENDAÑO

Co-Director:

M.C. FRANCISCO JOSÉ LÓPEZ CORTÉS

HEROICA PUEBLA DE ZARAGOZA, PUEBLA, MÉXICO.

SEPTIEMBRE 2024

Resumen

Este trabajo de tesis aborda estos desafíos mediante el desarrollo e implementación de un sistema de visión por computadora diseñado para operar en un laboratorio de síntesis de materiales. El objetivo principal es equipar a un robot colaborativo con la capacidad de interactuar de manera autónoma y segura con los objetos del entorno, facilitando así la automatización de tareas repetitivas y potencialmente peligrosas.

Se ha seleccionado la arquitectura YOLO (You Only Look Once) para el reconocimiento de objetos debido a su eficiencia y precisión en la detección de objetos en tiempo real. Para abordar el problema de la estimación de la profundidad, se ha utilizado una técnica de triangulación basada en imágenes de referencia, lo que permite obtener una estimación inicial de la distancia a los objetos.

El proceso de desarrollo del sistema incluyó la recopilación y preprocesamiento de un conjunto de datos de imágenes etiquetadas, la selección y entrenamiento de la arquitectura YOLO, y la implementación de algoritmos de postprocesamiento para mejorar la precisión de la detección y localización. Los resultados obtenidos muestran un desempeño prometedor en términos de precisión y velocidad de detección, aunque se identificaron áreas de mejora, como la robustez ante cambios en las condiciones de iluminación y la precisión de la estimación de la profundidad en casos de oclusiones parciales.

En conclusión, este trabajo presenta una solución viable para la implementación de sistemas de visión por computadora en entornos de laboratorio. Los resultados obtenidos demuestran el potencial de esta tecnología para automatizar tareas y mejorar la eficiencia de los procesos de síntesis. Sin embargo, se requiere de investigación adicional para abordar los desafíos pendientes y lograr una mayor generalización del sistema.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La visión por computadora se enfoca en entrenar a las computadoras para adquirir, procesar y analizar imágenes digitales para extraer información significativa y realizar una amplia gama de tareas [40], como clasificación de imágenes, reconocimiento de objetos, seguimiento de objetos, ubicación de objetos, segmentación de imágenes, recuperación de imágenes, reconocimiento de patrones y estimación de distancia, por mencionar algunas.

Para llevar a cabo estas tareas, se pueden emplear diversos métodos. Entre ellos destacan las redes neuronales convolucionales (CNN), que han demostrado un gran éxito en el reconocimiento de objetos. Su capacidad para capturar patrones y características relevantes a diferentes escalas espaciales ha sido la clave de su éxito y aplicabilidad en diversas áreas[26].

En el reconocimiento de objetos, uno de los principales desafíos es la escasez de datos etiquetados. La obtención de conjuntos de datos grandes puede ser costosa y laboriosa. Además, los objetos pueden cambiar en términos de tamaño, forma, orientación, iluminación y fondo, lo que dificulta la tarea de los modelos de reconocimiento. Los modelos deben entrenarse en conjuntos de datos que sean diversos y representativos de situaciones del mundo real, lo cual puede ser difícil de obtener.

Una tarea particular en la visión por computadora es la estimación de la distancia a objetos en una imagen 2D, la cual, presenta diversos desafíos que deben considerarse para lograr un desempeño confiable. Entre los desafíos se encuentran la falta de información de profundidad, la variabilidad de las condiciones de captura, y la escala y tamaño de los objetos. Las técnicas más comunes para estimar la distancia en imágenes las mas comunes son la triangulación, la estereos-

cópica y el aprendizaje automático. La elección de la técnica adecuada depende de la aplicación y las condiciones de captura.

Tanto la estimación de la distancia como el reconocimiento de objetos pueden implementarse en sistemas que se combinen con robots colaborativos (*cobots*). Un *cobot* puede interactuar de forma segura en estrecha proximidad espacial y temporal con humanos en tareas compartidas [16]. Sus áreas de aplicación más comunes son la logística, la manufactura, la salud y la agricultura.

Una de las áreas en las que se puede aplicar el reconocimiento y la ubicación espacial de objetos son los laboratorios de síntesis de materiales, que son centros de investigación donde se exploran y crean materiales. En estos laboratorios, la eficiencia y la precisión son esenciales para el éxito de los experimentos. En este contexto, tener un sistema de identificación y ubicación espacial de objetos puede ser de gran ayuda para una variedad de propósitos, por ejemplo, para automatizar procesos de síntesis de materiales identificando las herramientas necesarias para realizar alguna tarea o verificar si los materiales se han sintetizado correctamente.

1.1. Planteamiento del problema

El proyecto *"Automatización de Procesos de Síntesis y Caracterización Espectroscópica de Compuestos Orgánicos. Mediante el Uso de un Robot Colaborativo"*, es un proyecto de ciencia de frontera de CONAHCyT. Este proyecto se realiza en el laboratorio de síntesis de materiales ubicado en el eco campus de la Benemérita Universidad Autónoma de Puebla.

Para dicho proyecto es necesario realizar en primer lugar un sistema de reconocimiento y ubicación espacial de objetos. Por tal motivo en este trabajo de tesis, se planteó desarrollar un sistema de reconocimiento y ubicación espacial de objetos en un laboratorio de síntesis de materiales.

1.2. Objetivos

1.2.1. Objetivo general

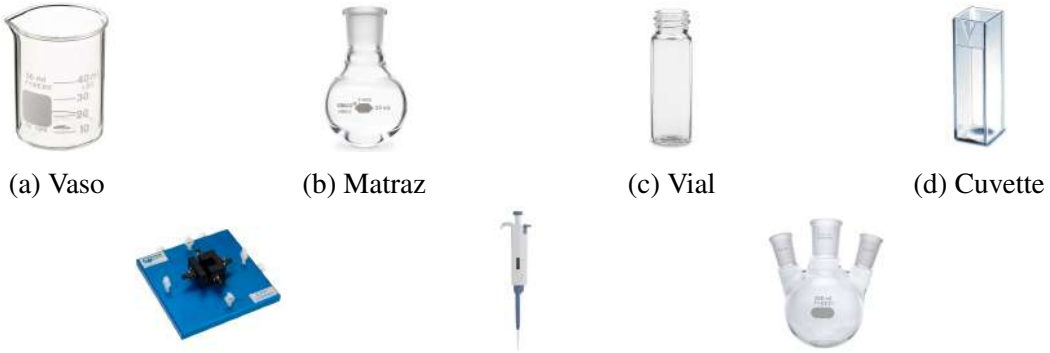
Desarrollar un sistema basado en una red neuronal convolucional para el reconocimiento y ubicación espacial de objetos en un laboratorio de síntesis de materiales.

1.2.2. Objetivos específicos

- Revisar el estado del arte de las técnicas de visión por computadora y redes neuronales convolucionales aplicadas al reconocimiento y ubicación espacial de objetos utilizando cámaras 2D.
- Realizar un caso de estudio para la identificación y ubicación espacial de objetos en un laboratorio de síntesis de materiales.
- Adaptar una arquitectura de red neuronal convolucional existente para que sea adecuada para el reconocimiento y ubicación de objetos en un laboratorio de sintesis de materiales.
- Recopilar y etiquetar un conjunto de datos representativo que incluya objetos relevantes para el laboratorio de síntesis de materiales, con el fin de entrenar y evaluar la red neuronal convolucional.
- Implementar el sistema de visión por computadora basado en la red neuronal convolucional para el reconocimiento y ubicación de objetos en un laboratorio de síntesis de materiales.
- Evaluar el rendimiento y la precisión del sistema propuesto en términos de reconocimiento y ubicación espacial de objetos, utilizando métricas apropiadas y realizando pruebas en diferentes escenarios de laboratorio.

1.3. Justificación

Un laboratorio de síntesis de materiales cuenta con equipo y material altamente especializado, por lo que la escasez de datos etiquetados limita la capacidad para entrenar modelos que ayuden dentro del laboratorio. En este proyecto se seleccionó un conjunto de 7 objetos particulares de un laboratorio de síntesis de materiales los cuales son: vaso de precipitado, matraz, vial, cuvette, base para muestras, pipeta de precisión y matraz de 3 bocas como se observa en la figura 1.1. Los cuales no son encontrados en su totalidad en los conjuntos revisados en la literatura como se muestra en la tabla 1.1. Por lo cual se requiere realizar un conjunto de datos personalizado para que la detección y ubicación espacial de objetos dentro del laboratorio sea posible.



(a) Vaso (b) Matraz (c) Vial (d) Cuvette

(e) Base (f) Pipeta (g) Matraz 3B

Figura 1.1: Instrumentos de laboratorio de síntesis de materiales seleccionados para el desarrollo del conjunto de imágenes personalizado.

Conjunto de imágenes	**Objetos seleccionados de un laboratorio de síntesis de materiales**						
	Matraz	Vial	Soporte de muestras	Pipeta de precisión	Cuvette	Vaso de precipitado	Matraz de 3 bocas
COCO [24]	✗	✗	✗	✗	✗	✗	✗
Objects365 [38]	✗	✗	✗	✗	✗	✗	✗
Roboflow 100 [3]	✗	✗	✗	✗	✗	✗	✗
Imagenet [7]	✗	✗	✗	✗	✗	✓	✗
LabPics V2 [36]	✓	✓	✗	✗	✗	✓	✗
CLAD [8]	✓	✗	✗	✗	✗	✓	✓

Tabla 1.1: Objetos que son reconocidos por los conjuntos de imágenes encontrados en la literatura

1.4. Metodología

El proyecto se desarrolló en cuatro etapas. En la primera, se realizó una revisión exhaustiva de las contribuciones previas, lo que permitió identificar las principales tendencias y desafíos en el campo de la detección de objetos en entornos de laboratorio. Esto estableció las bases para la implementación, garantizando que nuestra solución abordara las necesidades específicas de este dominio. Durante esta etapa, no se encontró un modelo preentrenado que detecte los objetos de laboratorio, ni una base de datos que los contenga a todos, por lo que se generó una base de datos personalizada y anotada manualmente.

Posteriormente, se seleccionó la ubicación de la cámara estratégicamente para maximizar la cobertura del área de trabajo y minimizar las interferencias de sombras y reflejos. Una vez ubicada la cámara, se diseñó una solución con dos módulos principales: el módulo de identificación de objetos, que trata de entrenar a un modelo con redes neuronales convolucionales. Para lograr esto, se realizaron las siguientes acciones: recopilación de imágenes bajo diversas condiciones de iluminación y desde diferentes ángulos, etiquetado de imágenes utilizando herramientas de anotación, aplicación de diversas técnicas de aumento de datos (rotación, escalado, volteo horizontal, ruido gaussiano, etc.) y reentrenamiento de la CNN (YOLO-NAS). El segundo módulo calculará la distancia entre la cámara y el objeto usando la similitud del triángulo, por lo que es necesario la captura de imágenes de referencia con marcas de calibración y el cálculo de la distancia focal utilizando técnicas de calibración de cámara estándar.

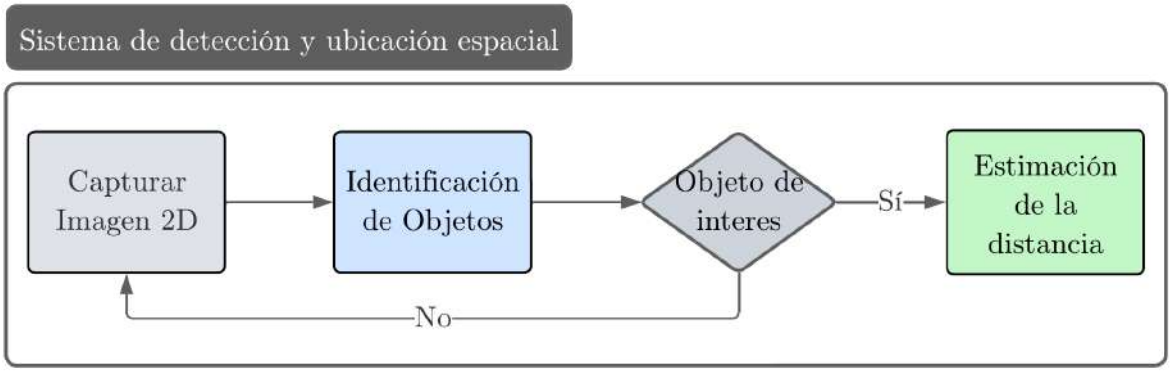


Figura 1.2: Diagrama general del sistema de detección y ubicación espacial

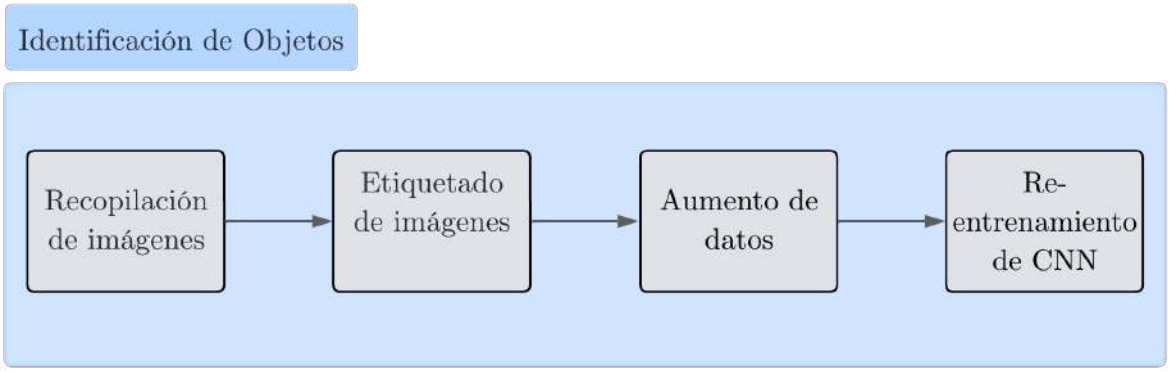


Figura 1.3: Etapas realizadas para el desarrollo del modulo de Identificación de objetos

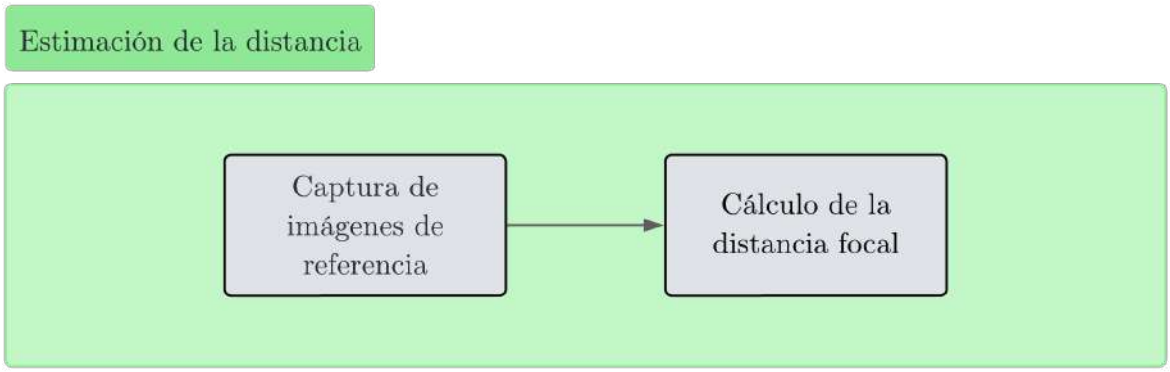


Figura 1.4: Etapas realizadas para poder estimar la distancia entre el objeto y la cámara

Para probar la eficacia del sistema, se evaluó en un ambiente controlado con dimensiones similares al área de trabajo que existe en el laboratorio de síntesis de materiales, evaluando su desempeño utilizando la métrica mAP (Mean Average Precision). Los resultados obtenidos superaron nuestras expectativas iniciales, demostrando la viabilidad de la solución propuesta. En particular, se obtuvo un mAP del 86 % en el conjunto de prueba, lo que indica una alta precisión en la detección y localización de los objetos de interés.

Finalmente, se documentaron los resultados obtenidos, alcanzando los objetivos planteados y proporcionando una base sólida para futuras investigaciones. Los detalles técnicos se presentan con mayor profundidad en el capítulo 4.

1.5. Hipótesis planteada

Tomando en cuenta lo expuesto, se formula la siguiente hipótesis a comprobar:

Se puede desarrollar sistema de identificación y ubicación espacial de objetos basado en redes neuronales convoluciones que sea lo suficientemente preciso para identificar y ubicar objetos dentro de un laboratorio de sintesis de materiales.

Capítulo 2

Estado del Arte

El reconocimiento y la ubicación espacial de objetos son tareas cruciales en una variedad de aplicaciones, como seguridad, vigilancia, robótica, agricultura, navegación autónoma y la realidad aumentada por mencionar algunas. En los últimos años, el uso de las redes neuronales profundas en especial las redes neuronales convolucionales (CNN) han demostrado tener un potencial significativo para abordar estos desafíos de manera precisa y eficiente.

2.1. Arquitecturas

Dentro de la literatura se pueden encontrar diversas arquitecturas de CNN. Arquitecturas como Faster R-CNN [35] y YOLO [33] han demostrado ser efectivas para la detección en tiempo real y la ubicación espacial precisa en imágenes y vídeos, además de ser base para nuevas arquitecturas que se han generado. Por otro lado se han desarrollado arquitecturas como SSD [25] y nuevas versiones de YOLO [42] que optimizan la velocidad de procesamiento para la detección en tiempo real. También hay arquitecturas que han logrado avances significativos en el ámbito de la precisión y la segmentación como es el caso de RetinaNet [23], mientras que Mask R-CNN [17] ha demostrado ser una herramienta poderosa para la segmentación.

Más recientemente, se han desarrollado arquitecturas como MobileNet [18] que ha sido diseñada para dispositivos móviles con menor potencia de procesamiento, y EfficentDet [41] que ofrece un balance entre precisión y eficiencia. Actualmente dentro del campo de la detección de objetos se encuentra YOLO-NAS [42] la cual aprovecha la búsqueda de arquitectura neuronal para me-

jorar su rendimiento, es decir la arquitectura se optimiza automáticamente utilizando algoritmos avanzados. También se esta investigando el aprendizaje por refuerzo profundo (RL) para la detección de objetos. El RL es un paradigma de aprendizaje automático en el que un agente aprende a tomar decisiones en un entorno para maximizar una recompensa [37]. En la tabla 2.1 se muestra algunas de las características importantes de cada arquitectura y en la figura 2.1 la comparación del rendimiento de las arquitecturas citadas.

Comparación de rendimiento de las arquitecturas encontradas en la literatura				
Arquitectura	**Año**	**Características**	**Precisión**	**Velocidad**
YOLOv3 [34]	2015	Usa tres escalas diferentes para la detección, usando tres tamaños de núcleos de detección: 13x13, 26x26 y 52x52.	*m*AP = 28.20 %	45 FPS
Faster R-CNN [35]	2016	Utiliza una Red de Propuesta de Regiones (RPN) para generar propuestas de regiones de alta calidad para la detección de objetos.	*m*AP = 18.9 %	5 FPS
SSD [25]	2016	Utiliza una arquitectura de "feed-forward" para predecir la clase y ubicación de los objetos directamente a partir de la imagen de entrada.	*m*AP = 23.2 %	46 FPS
Mask R-CNN [17]	2018	Es un modelo de red neuronal que genera una máscara que se ajusta a la forma precisa de cada instancia del objeto detectado.	*m*AP = 39.8 %	5 FPS
RetinaNet [23]	2018	Modelo de una etapa que utiliza una función de pérdida focal que asigna mayor importancia a ejemplos negativos para mejorar la precisión.	*m*AP = 31.9 %	12 FPS
EfficentDet [41]	2020	Utiliza un enfoque de escalado compuesto para crear versiones del modelo con diferentes niveles de capacidad, manteniendo el balance entre precisión y eficiencia.	*m*AP = 43.0 %	41.7 FPS
YOLO NAS [42]	2023	Se ejecuta en una sola etapa, presenta una velocidad de procesamiento mejorada respecto a modelos YOLO anteriores.	*m*AP = 49.3 %	150 FPS

Tabla 2.1: Comparación del rendimiento de las arquitecturas sobre el conjunto de datos MS COCO con un tamaño de entrada similar [47]. *m*AP (mean Average Precision) mide la precisión promedio de un modelo de detección de objetos, calculado promediando el AP (Average Precision) en 80 clases y 10 umbrales de IoU de 0.50 a 0.95. FPS (Frames Per Second) indica la cantidad de imágenes mostradas por segundo en un video.

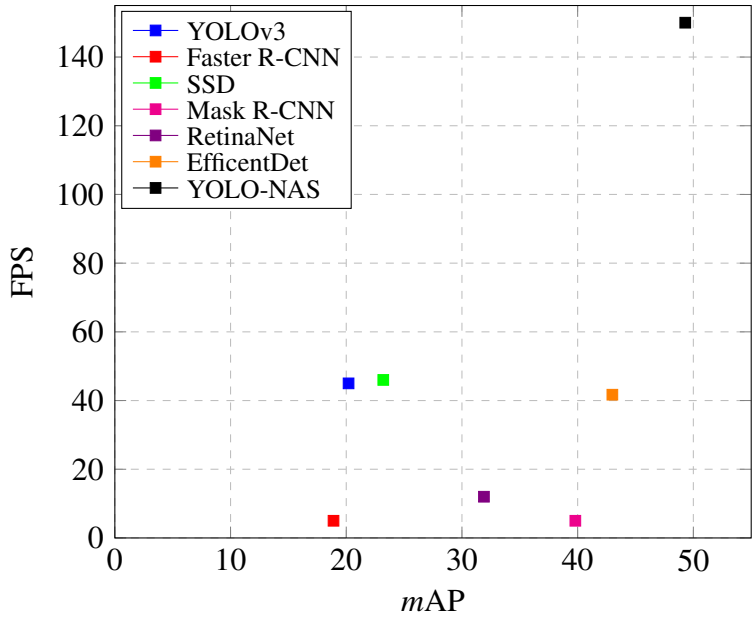


Figura 2.1: Relación de velocidad con la *m*AP de las arquitecturas revisadas.

2.2. Conjuntos de Imágenes

2.2.1. Conjuntos de entrenamiento

En la literatura se encontraron conjuntos de datos con grandes cantidades de imágenes que se usan comúnmente para el entrenamiento de CNN como es el caso del *Common Objects in Context* (COCO)[24] el cual contiene 330000 imágenes con 80 categorías de objetos, otro conjunto usado es *Objects365*[38] que contiene 365 categorías de objetos con mas de 2 millones de imágenes. Por otro lado *Roboflow 100*[3] es un conjunto de datos que abarca una amplia gama de dominios y contiene imágenes del mundo real que representan escenarios cotidianos. *ImageNet*[7] es un conjunto de datos masivo que contiene mas de 14 millones de imágenes cuidadosamente seleccionadas y anotadas, las imágenes abarcan una amplia gama de categorías.

2.2.2. Conjuntos especializados

Para el entrenamiento de las CNN se requiere una cantidad considerable de datos por lo que es necesario revisar en la literatura si ya hay bases de datos especializadas en instrumentos de laboratorio de síntesis de materiales.

En la búsqueda realizada se encontró el trabajo “LabPics V1” el cual contiene 2187 imágenes en 61 categorías de experimentos químicos con materiales dentro de recipientes en su mayoría transparentes en diversos entornos de laboratorio y en condiciones cotidianas. Cada imagen del conjunto de datos tiene una anotación de la región de cada fase del material y su tipo [11].

Para su segunda versión el conjunto de datos “LabPics V2” contiene 10528 imágenes anotadas. Las imágenes se dividen en dos conjuntos: un conjunto de entrenamiento con 8422 imágenes y un conjunto de prueba con 2.106 imágenes. Las imágenes del conjunto de datos “LabPics V2” se obtuvieron de una variedad de fuentes, incluyendo imágenes de laboratorios reales, imágenes de simulaciones de laboratorio entre otras [36], algunos ejemplos de las imágenes que contiene el conjunto se muestran en la figura 2.2.

Figura 2.2: Ejemplos de imágenes del conjunto de datos LabPics

Un trabajo más reciente es “Chemistry Laboratory Apparatus Dataset” el cual contiene 21 tipos diferentes de imágenes de instrumentos químicos de laboratorio, con no menos de 200 imágenes de cada tipo, principalmente instrumentos de vidrio están cuidadosamente etiquetados con información de estudiantes de química. Cada imagen puede contener una o más imágenes de instrumentos químicos [8] como las imagenes que se muestran en la figura 2.3.

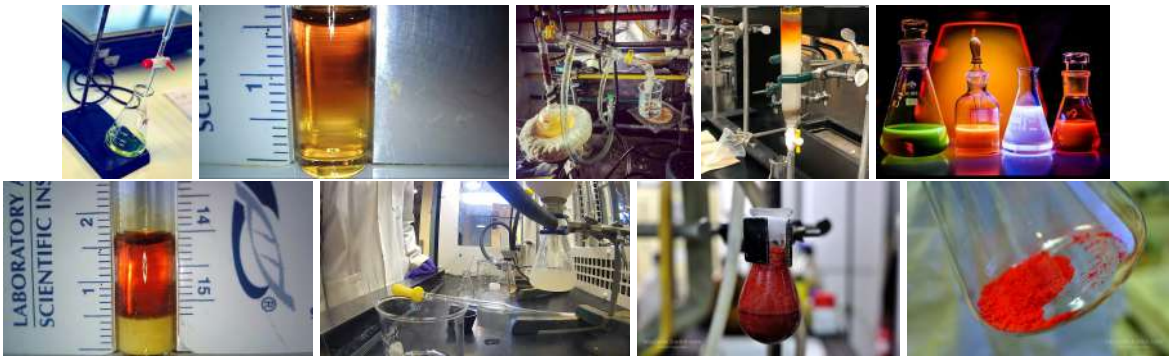


Figura 2.3: Ejemplos de imágenes del conjunto de datos CLAD

Conjuntos de datos de imágenes de equipo de laboratorio			
Nombre	**Tipo de imágenes**	**Cantidad de imágenes**	**Categorías**
COCO	Animales, personas, vehículos, muebles, utensilios de cocina, instrumentos musicales, deportes, ciudades, paisajes, campos, comida, animales salvajes y objetos domésticos	330000	80
Objects365	Objetos del mundo real en diversas condiciones de iluminación, pose, tamaño, forma y fondo	2 millones	365
Roboflow 100	Imágenes de aviones, drones, células, tejidos, criaturas marinas, documentos, campos de radar, animales, plantas, objetos, personas, lugares, eventos, transporte, arte, moda, comida y textos.	232000	828
Imagenet	Gran variedad de imágenes, incluyendo animales, plantas, objetos, personas, lugares, eventos, transporte, arte, moda, comida, textos y mucho más.	14 millones	22000
LabPics V1*	Materiales y recipientes en laboratorios de química.	2187	61
LabPics V2*	Materiales y recipientes en laboratorios de química y medicina.	10528	61
CLAD*	Instrumentos de laboratorio de química.	2246	21

Tabla 2.2: Tipo de imágenes que contiene cada conjunto así como el numero de categorías. *Son conjuntos especializados en objetos de laboratorio

2.3. Identificación de objetos

El campo de la agricultura es un área en la que se encuentran aplicaciones como en el artículo "Uso de redes neuronales convolucionales para la detección remota de frutos con cámaras RGB-D" [13] se presenta un método para la detección remota de frutos en imágenes tomadas con cámaras RGB-D. Los resultados mostraron que el método fue capaz de detectar y clasificar las manzanas con una precisión del 91,4 %. Los autores concluyen que su método es eficiente, preciso y se puede implementar en cualquier plataforma que admita CNN. Otro método de detección dentro del área de agricultura se muestra en [4] en donde se describe el desarrollo de un nuevo método para contar manzanas en árboles frutales. El método se basa en el uso de sensores 3D para escanear los árboles y luego utilizar técnicas de aprendizaje automático para identificar y contar las manzanas con una precisión del 95 %; el método es rápido, preciso y no requiere mano de obra humana. Otro trabajo dentro de la misma área es el propuesto por [48] en el que se propone un sistema para la detección y manipulación robótica para la recolección de cultivos mediante la combinación de la arquitectura de YOLO y la arquitectura VGG16 y entrenándolas con un conjunto de datos especifico para el sistema. La investigación demuestra la gran capacidad de las CNN para el reconocimiento de objetos ya la predicción precisa de la posición óptima de agarre del robot. El artículo [22] propone un modelo llamado YOLO-Submarine Cable (YOLO-SC), basado en la red neuronal YOLO-V3. Este modelo está específicamente modificado para detectar cables submarinos, teniendo en cuenta sus características únicas como forma alargada y presencia dominante en las imágenes. Entre los trabajos realizados con robots colaborativos, en particular con el robot ABB YuMi, se encuentra el presentado en [21]. El artículo presenta un caso de uso específico en el que YuMi, equipado con una cámara y algoritmos de visión, identifica y manipula las piezas del rompecabezas. El robot trabaja con el humano para completar el rompecabezas de manera eficiente, turnándose para recoger y colocar las piezas. Otro trabajo que interactúa con el mismo robot es [28] donde la investigación se basa en el uso de una cámara para reconocer los gestos de las manos y la implementación de inteligencia artificial para controlar un robot YuMi.

2.4. Estimación de distancia

En el ámbito de la visión por computadora, la estimación de la distancia a los objetos presentes en imágenes 2D representa un reto fundamental. La naturaleza bidimensional de las imágenes y la ausencia de información de profundidad dificultan la obtención de datos de profundidad a partir de ellas. Se han desarrollado diversos métodos y técnicas, cada uno con sus propias ventajas y limitaciones como el trabajo presentado en [30] en el cual se propone un Sistema Avanzado de Asistencia al Conductor en el que se realizan mediciones de distancias con una combinación de una cámara 2D y un sensor LiDAR(Light Detection and Ranging) 2D, reportando una taza de error de 0.197 metros. Otro articulo en el cual se estima la distancia con la combinación de un sensor con una cámara es el [9] el cual se presenta una solución para la del movimiento usando una cámara omnidireccional y un sensor LRF(Laser Range Finder), un articulo en el que solamente se usa una cámara 2D para estimar distancia es el presentado por Gouranga Mandal[27] que presenta un método para estimar la distancia entre vehículos en tiempo real utilizando una cámara 2D por la noche. La combinacion de vision por computadora y robots colaborativos se puede ver en el trabajo realizado en [32] en el que se estima la distancia entre un robot y una persona utilizando solamente una imagen capturada por una cámara 2D y la distancia euclidiana entre una oreja y un punto del torso.

La elección de la técnica adecuada para estimar la distancia en imágenes 2D depende de la aplicación específica y de las condiciones de captura. La investigación en esta área continúa activa, con el objetivo de desarrollar métodos más robustos y precisos para estimar la distancia en una amplia gama de escenarios.

Capítulo 3

Marco teórico

3.1. Redes neuronales convolucionales

Una red neuronal convolucional (CNN) es un tipo de red neuronal que puede aprender características muy abstractas de objetos para poder identificarlos con eficacia. Una CNN consiste en un conjunto finito de capas de procesamiento que pueden aprender varias características de los datos de entrada con múltiples niveles de abstracción. Las capas iniciales aprenden y extraen las características con menor abstracción, y las capas mas profundas aprenden y extraen las características con mayor abstracción [15]. Como se muestra en la figura 3.1 una CNN esta compuesta de múltiples capas que se describen a continuación.

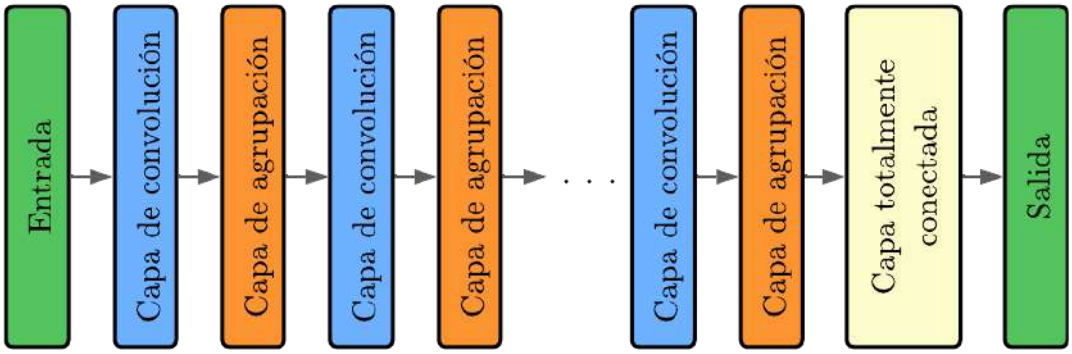


Figura 3.1: Modelo conceptual de una CNN

3.1.1. Capa de convolución

La capa de convolución es un componente fundamental que se encarga de extraer características de los datos de entrada. Contiene un conjunto de núcleos convolucionales, que al aplicarse a la imagen generan un mapa de características de salida.

Núcleo convolucional

Un núcleo convolucional también conocido como *kernel* o filtro es una matriz de valores discretos que recorre la imagen de entrada, realizando operaciones matemáticas con los píxeles que se encuentran dentro de su ventana. Después de cada recorrido el resultado de las operaciones se almacena en un mapa de características, cada elemento dentro del mapa de características representa la respuesta del *kernel* a un patrón local especifico de la imagen de entrada. En una capa se suelen utilizar diferentes *kernel* para capturar diferentes características, algunos ejemplos de filtros se muestran en la figura 3.2.

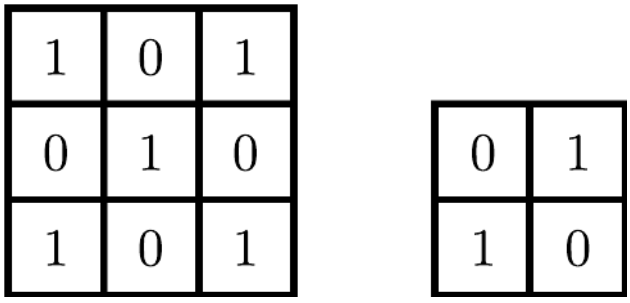


Figura 3.2: Ejemplos de *kernel* usados en la operación de convolución

Operación de convolución

La operación de convolución en una CNN es una un proceso matemático para extraer características importantes de las imágenes. La operación de convolución funciona recorriendo el *kernel* sobre toda la imagen y multiplicando elemento a elemento los valores de los píxeles con los valores del *kernel*. Este proceso genera un mapa de características que resalta la presencia de patrones específicos en la imagen. Al tener capas sucesivas de convolución permiten extraer características cada vez más abstractas, desde bordes y texturas hasta objetos completos.

Principales ventajas de las capas de convolución

- **Conectividad dispersa:** A diferencia de una red neuronal totalmente conectada donde cada neurona de una capa se conecta con cada neurona de la capa siguiente, en una CNN hay un número reducido de pesos entre dos capas lo que implica tener un numero reducido de conexiones por lo que la cantidad de memoria para almacenar dichas conexiones es pequeña volviendo mas eficiente el proceso.

- **Peso compartido:** En las CNNs todos los pesos funcionan con todos y cada uno de los pixeles de la matriz de entrada. En lugar de aprender nuevos pesos para todas las entradas, lo que reduce drásticamente el tiempo de entrenamiento [15].

3.1.2. Capa de agrupación

La capa de agrupación (pooling layer) es otro componente fundamental en las CNNs. Su principal función es reducir la dimensionalidad de los mapas de características obtenidos de las capas de convolución, conservando siempre las características mas dominantes en cada paso del pool. Existen distintos tipos de técnicas de agrupación que se utilizan en diferentes capas de agrupación, como *max pooling*, *min pooling*, *average pooling*, *gated pooling*, *L2 pooling*, etc.

Operaciones más comunes de agrupación en CNNs

- **Max Pooling:** Es la operación de agrupación más utilizada. Esta operación toma el valor máximo dentro del mapa de características. Al seleccionar el valor máximo, se resalta la característica mas importante en región del mapa del características.

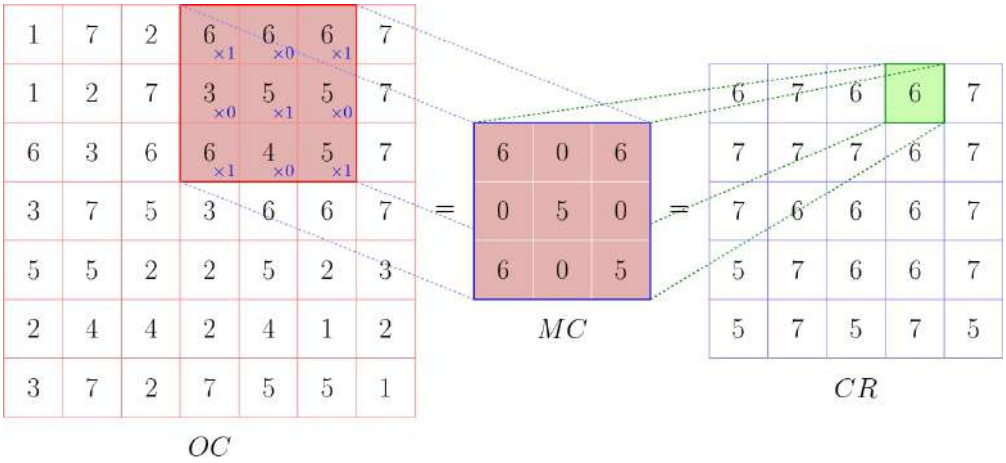


Figura 3.3: Ejemplo de Max Pooling.

- **Average Pooling:** Esta operación calcula el promedio de los valores dentro del mapa de características, no destaca las características mas destacadas como lo hace Max Pooling, lo que puede ser útil en tareas donde se busca una representación más uniforme.

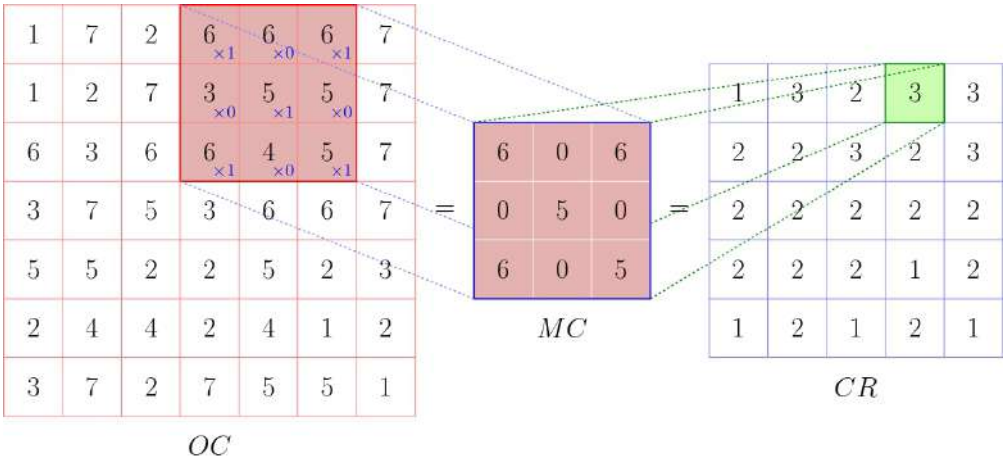


Figura 3.4: Ejemplo de Average Pooling.

- **L2 Pooling:** Esta operación calcula raiz cuadrada de la suma de los cuadrados de los valores dentro del mapa de características, este método ayuda a normalizar la distribución de valores en los mapas de características, lo que puede mejorar el rendimiento de la CNN.

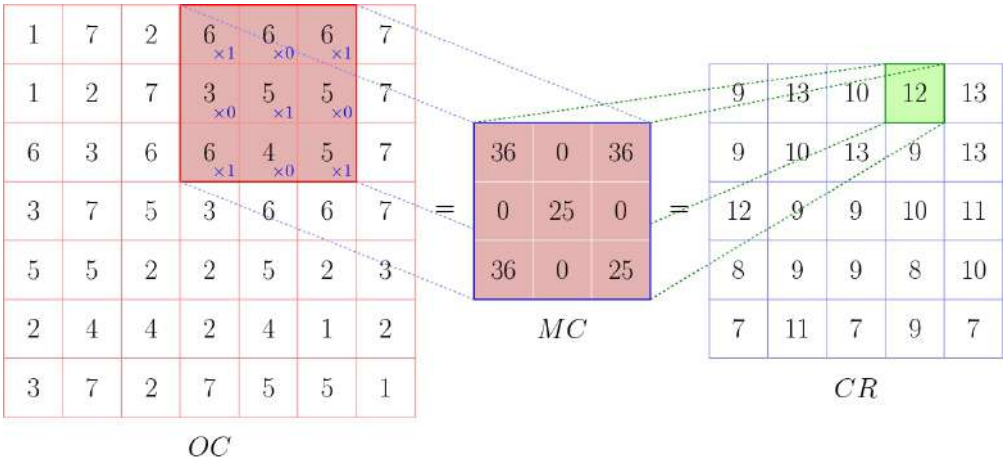


Figura 3.5: Ejemplo de L2 Pooling.

3.1.3. Capa de totalmente conectada

La última parte de cada arquitectura CNN (utilizada para la clasificación) está formada por capas totalmente conectadas, en las que cada neurona de una capa está conectada con cada neurona de su capa anterior. Las capas totalmente conectadas reciben información de la ultima capa convolucional o de agrupación (mapa de características), este mapa se aplana para crear un vector que se introduce en la capa totalmente conectada la ultima de estas capas se utiliza como capa de salida o clasificador de la arquitectura CNN[15] como se muestra en la figura 3.6.

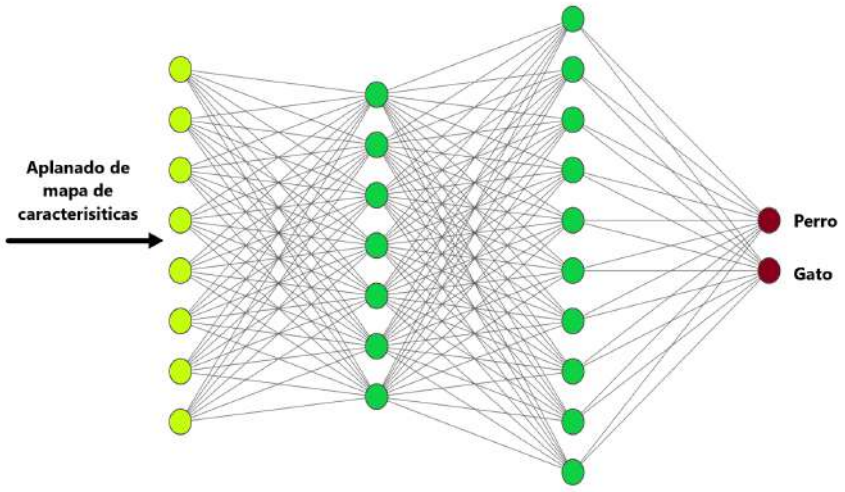


Figura 3.6: Capa totalmente conectada o clasificador.

3.1.4. Funciones de activación

La tarea principal la función de activación dentro de las redes neuronales es decidir si una neurona se activará o no para una entrada predeterminada produciendo la salida correspondiente. Dentro de las arquitecturas de CNNs se usan capas de activación no lineales después de las capas convolucionales y capas totalmente conectadas permitiendo a la CNN aprender cosas más complejas. Dentro de las funciones de activaciones más utilizadas se encuentran:

Sigmoide

La función de activación sigmoide toma números reales como entrada y enlaza la salida en el intervalo [0,1]. Esta función se usa principalmente en los modelos de clasificación binaria ya que reduce la salida a un valor de probabilidad entre 0 y 1 que puede interpretarse como la probabilidad de que la entrada pertenezca a una clase determinada.La representación matemática de sigmoide es:

$$f(x)_{sigm} = \frac{1}{1+e^{-x}}$$

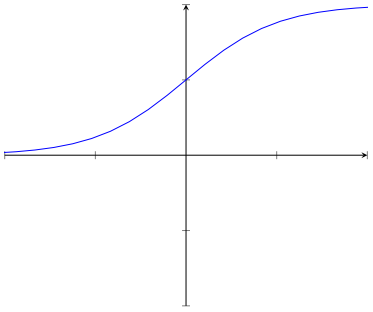


Figura 3.7: Sigmoide

Tanh

La función de activación *Tanh* se utiliza para ligar los valores de entrada dentro del rango [-1,1]. Esta función se utiliza con frecuencia en las capas ocultas de una red neuronal. Sirve principalmente para normalizar los datos ya que conn datos normalizados puede resultar un entrenamiento más eficiente. La representación matemática de *Tanh* es:

$$f(x)_{tanh} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

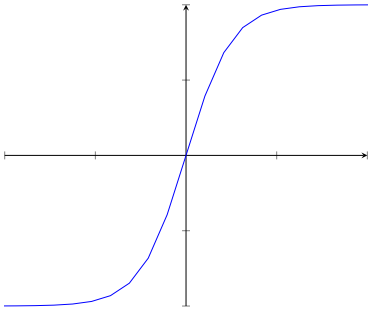


Figura 3.8: *Tanh*

ReLU

Es la función de activación más utilizada en las CNNs. Se utiliza para convertir todos los valores de entrada en números positivos. La ventaja de ReLU es que requiere una carga computacional mínima porque implica un simple umbral en cero. Esto permite a las redes escalar a muchas capas sin un aumento significativo de la carga computacional, en comparación con funciones más complejas como la función *Tanh* o la sigmoide. La representación matemática de ReLU es:

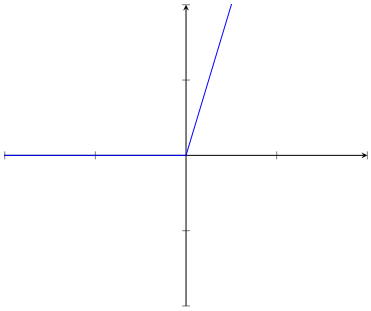


Figura 3.9: *ReLU*

$$f(x)_{ReLU} = \mathbf{max}(0, x)$$

3.1.5. Funciones de pérdida

Su función principal es cuantificar la discrepancia entre las predicciones realizadas por la red y los valores reales presentes en los datos de entrenamiento.
En la capa de salida, se calcula el error de predicción generado por el modelo CNN sobre las muestras de entrenamiento utilizando alguna función de pérdida. La función de pérdida calcula el error comparando la salida estimada del modelo CNN (predicción) con la salida real (etiqueta). Este error indica qué tan lejos está la predicción del modelo de la realidad y se utiliza para optimizar el modelo durante el proceso de aprendizaje. Hay diferentes tipos de funciones de pérdida que se utilizan en diferentes tipos de problemas, y la elección de la función adecuada depende de la tarea específica que se esté abordando.[15]

Algunos de los tipos comunes de funciones de pérdida en CNNs son:

- Entropía cruzada binaria: Utilizada para tareas de clasificación binaria, donde la salida de la red es una probabilidad entre 0 y 1.
- Entropía cruzada categórica: Similar a la entropía cruzada binaria, pero utilizada para tareas de clasificación de más de dos categorías.
- Error cuadrático medio (MSE): Utilizada para tareas de regresión, donde la salida de la red es un valor numérico.

3.2. YOLO (You Only Look Once)

YOLO es una arquitectura de red neuronal convolucional diseñada para la detección de objetos en imágenes y vídeos presentada por primera vez en el artículo [33]. YOLO se diferencia de los métodos de detección de objetos anteriores en que realiza la detección de objetos en un solo paso. Esto significa que la red neuronal no necesita recorrer la imagen varias veces para detectar todos los objetos. Esto hace que YOLO sea mucho más eficiente que los métodos anteriores, lo que lo hace ideal para aplicaciones que requieran respuestas cercanas al tiempo real.
YOLO cuenta con ventajas significativas sobre otros métodos de detección de objetos, entre sus principales ventajas destacan:

- **Velocidad:** YOLO es capaz de procesar imágenes a velocidades elevadas, tan solo la primera versión llega hasta 45FPS y ha aumentado con la salida de nuevas versiones llegando hasta 150FPS en ultimas versiones.
- **Precisión:** YOLO ofrece también una alta precisión en la detección de objetos. Alcanzando más del doble de precisión media (*m*AP) que otros métodos de detección de objetos.
- **Simplicidad:** Comparado con otros métodos la implementación y el entrenamiento de YOLO es sencillo, lo que convierte a este método en una opción preferida para los desarrolladores.
- **Detección en tiempo real:** Es capaz de realizar la detección de objetos en tiempo real, lo que hace ideal para aplicaciones donde la respuesta inmediata es esencial.

3.2.1. Arquitectura

La arquitectura de YOLO se divide en tres etapas:

- **Etapa de extracción de características:** Consta de 24 capas de convolución intercaladas por capas de *max pooling*.
- **Etapa de predicción:** En esta etapa hay dos capas totalmente conectadas, una predice las coordenadas de la caja delimitadora del objeto, la probabilidad de que la caja contenga un

objeto y la clase del objeto. La segunda predice la probabilidad de que la caja contenga un objeto de una clase especifica.

- **Etapa de postprocesamiento:** Esta etapa se encarga de fusionar las predicciones de las dos capas totalmente conectadas para generar una lista de objetos detectados en la imagen.

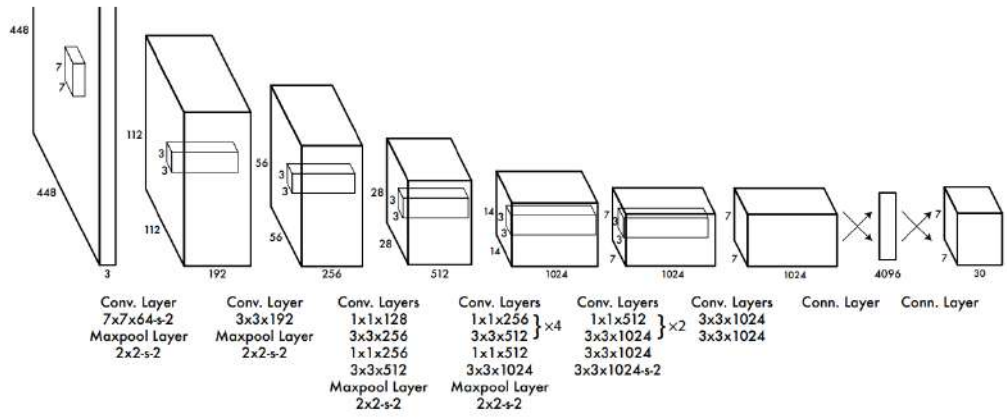


Figura 3.10: Arquitectura de YOLO

3.2.2. Funcionamiento de la detección de objetos con YOLO

Primero se divide la imagen en una cuadricula de ***SxS***. Cada celda de la cuadricula predice ***B*** recuadros delimitadores (*bounding boxes*) y calcula el nivel de confianza para cada recuadro. Estos niveles de confianza reflejan la seguridad que tiene el modelo de que la celda contiene un objeto y también lo precisa que cree que es la celda que predice. La confianza se define como $Pr(Object) * IOU_{pred}^{truth}$ donde si no hay un objeto en la celda, el nivel de confianza debe ser cero. En caso contrario el nivel de confianza es igual a la intersección sobre la unión (IOU) entre la celda predicha y la imagen real. Cada cuadro delimitador consta de 5 predicciones: *x,y,w,h,* y confianza. Las coordenadas (x,y) representan el centro del cuadro en relación con los límites de la celda. *w,h* es el ancho y el alto que se predicen en relación con toda la imagen. En cada celda de la cuadrícula también se predice C probabilidades de clase condicionales, $Pr(Class_i|Object)$, siempre y cuando la celda contenga un objeto. En el momento de la predicción, se multiplican

las probabilidades de clase condicionales y las predicciones de confianza de cada casilla, lo que nos da puntuaciones de confianza específicas de la clase para cada casilla. Estas puntuaciones codifican tanto la probabilidad de que esa clase aparezca en la casilla como la adecuación de la casilla predicha al objeto.[33]

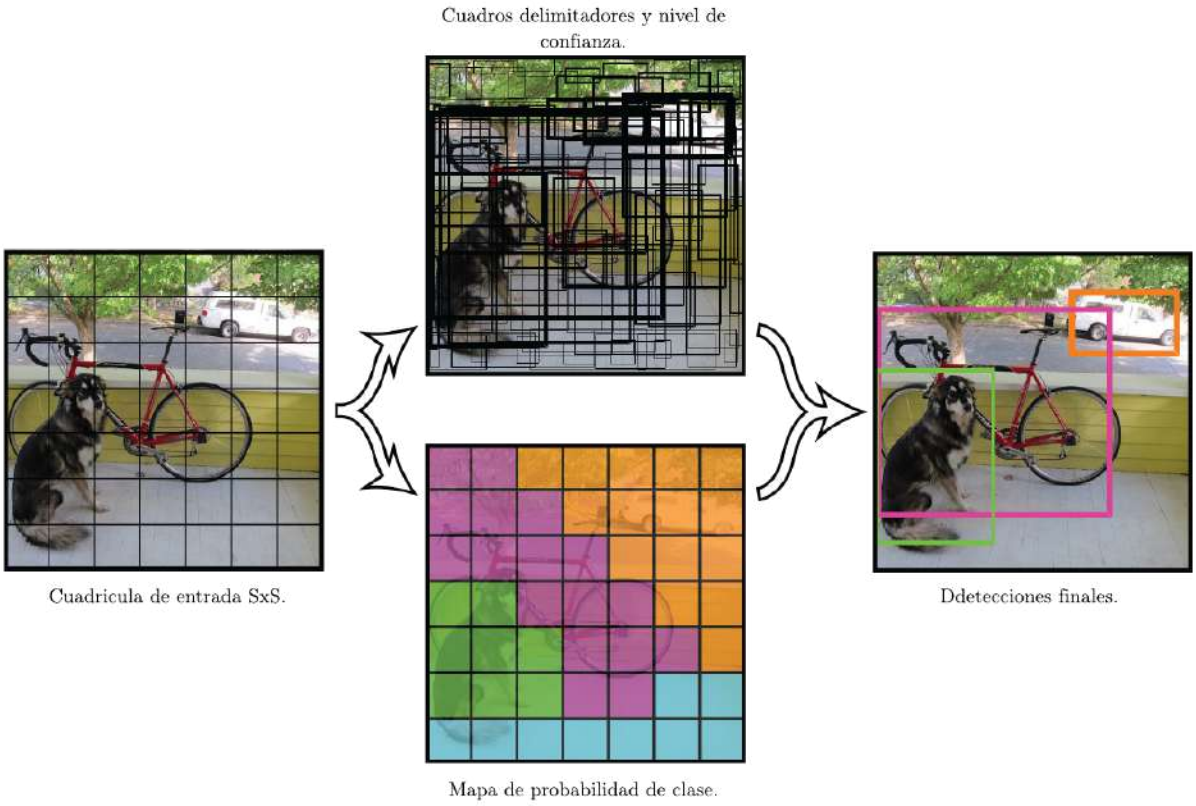


Figura 3.11: Funcionamiento de la detección de YOLO

3.2.3. YOLO-NAS

En 2023 se lanzo un nuevo modelo de YOLO, YOLO-NAS desarrollado por Deci el cual logra mejorar la velocidad y precisión de las versiones anteriores. El sistema AutoNAC, que fue decisivo en la creación de YOLO-NAS, es versátil y puede adaptarse a cualquier tarea, a las características específicas de los datos, al entorno para hacer inferencias y al establecimiento de objetivos de rendimiento. Esta tecnología tiene en cuenta los datos y el hardware y otros elementos que intervienen en el proceso de inferencia, como los compiladores y la cuantización. Se generaron tres arquitecturas variando la profundidad: YOLO-NAS S, YOLO-NAS M y YOLO-NAS L (S, M, L

para pequeño, mediano y grande, respectivamente)[42]. La figura 3.12 muestra la arquitectura del modelo de YOLO-NAS L.

El proceso de entrenamiento multiface de YOLO-NAS incluye el preentrenamiento usando el conjunto de datos llamado Object365, que contiene 2 millones de imágenes y 365 categorías. Posteriormente se entrena en un conjunto de datos adicional llamado COCO que contiene 330000 imágenes y 80 categorías. La combinación de estos dos pasos mejora el rendimiento del modelo YOLO-NAS[5]. YOLO-NAS es una excelente opción para aplicaciones de detección de objetos que requieren un alto nivel de precisión, eficiencia y flexibilidad. Su diseño modular, su facilidad de uso y su capacidad de escalabilidad lo convierten en una herramienta poderosa para la detección de objetos en imágenes y vídeos.

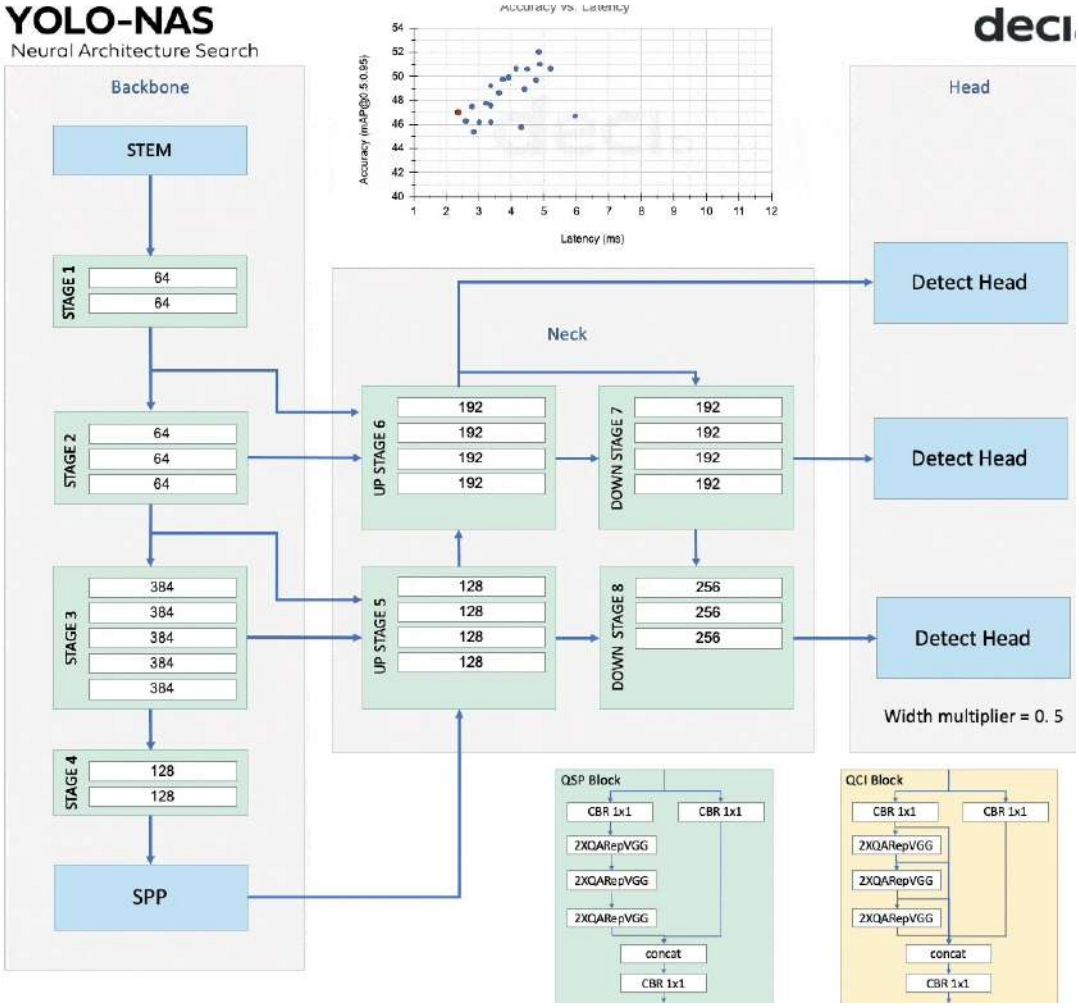


Figura 3.12: Arquitectura YOLO-NAS. La arquitectura optima se encuentra automáticamente mediante un sistema de búsqueda neural de arquitectura (NAS) llamado AutoNAC para equilibrar latencia y rendimiento.

Entrenamiento de YOLO-NAS con un conjunto de datos propio

El entrenamiento de YOLO-NAS con un conjunto de datos propio es el proceso de entrenar un modelo YOLO-NAS para que se adapte a las necesidades específicas de la aplicación. Entrenar YOLO-NAS con un conjunto de datos propio tiene varias ventajas:

- **Personalización:** Se puede adaptar el modelo para que reconozca objetos específicos de nuestro interés que no estén presentes en conjuntos de datos que se usan para el preentrenamiento.
- **Mejora del rendimiento:** Al entrenar con datos que son representativos de las condiciones y variaciones específicas de la aplicación, el modelo puede alcanzar una mayor precisión y robustez.
- **Control sobre los datos:** : El entrenamiento con un conjunto propio permite tener el control total sobre la calidad y cantidad de los datos, lo que permite mejorar el conjunto de datos de forma continua.

El proceso de entrenamiento de YOLO-NAS con un conjunto de datos propio se compone de los siguientes pasos[46]:

1. Recopilación y preparación de imágenes.

En este paso es donde se realiza el proceso de recolección de imágenes que sean representativas de los objetos que se desean detectar, es recomendable incluir diferentes ángulos, fondos e iluminaciones.

Posterior a la recopilación, es necesario etiquetar los objetos dentro de las imágenes, para realizar el etiquetado se definen las clases de los objetos y se realiza el etiquetado que es encerrar en un recuadro al objeto y asignarle una clase, esta tarea se puede hacer de forma manual o se puede hacer uso de herramientas ya sea en linea o aplicaciones de escritorio, por ejemplo:

- **LabelImg**[45]: Es una herramienta de anotación gráfica escrita en Python que es fácil de usar y compatible con los formatos de anotación YOLO y Pascal VOC.

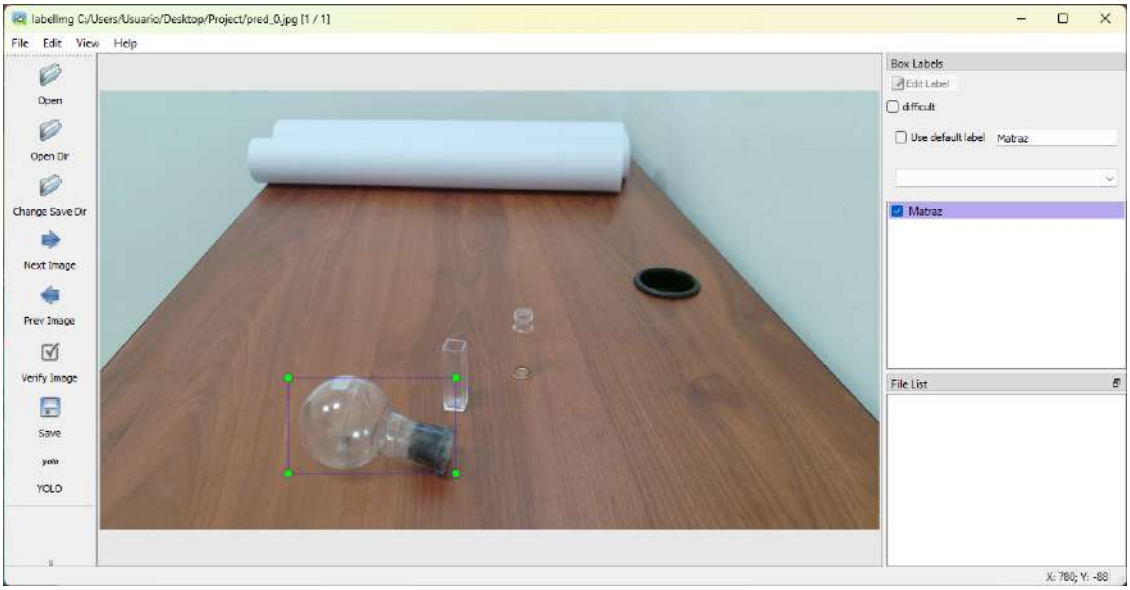


Figura 3.13: Interfaz de LabelImg

- **CVAT**[19]: Es una herramienta de anotacion de código abierto para tareas de visión por computadora, cuenta con una version online.

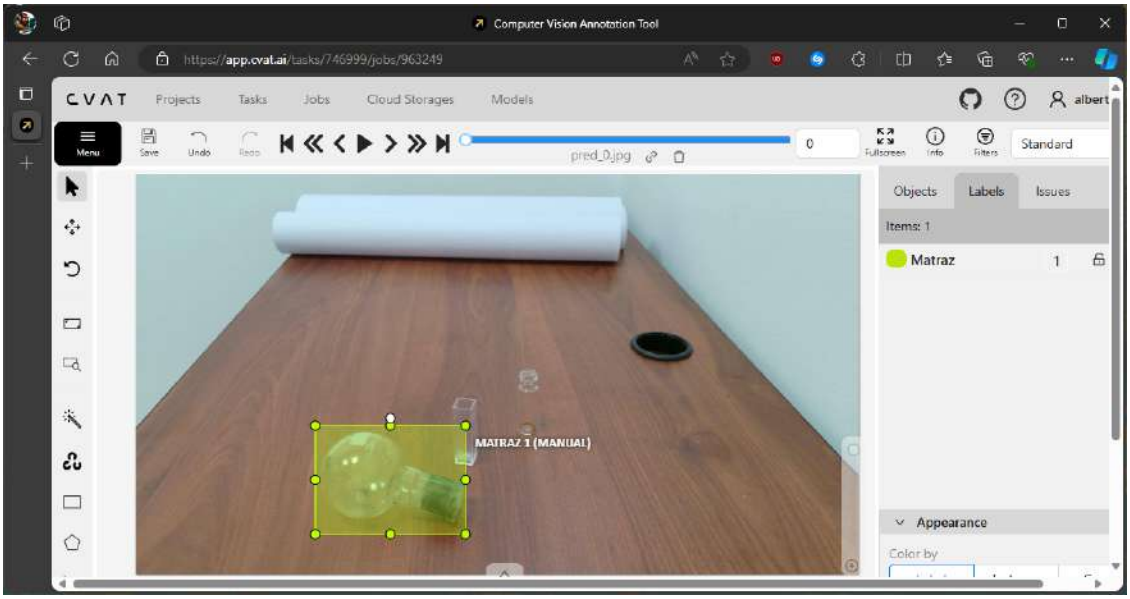


Figura 3.14: Interfaz de CVAT

- **MakeSense**[39]: Es una herramienta en linea, admite multiples tipos de etiquetas y admite formatos de archivo de salida como YOLO, VOC XML, VGG JSON, CSV.

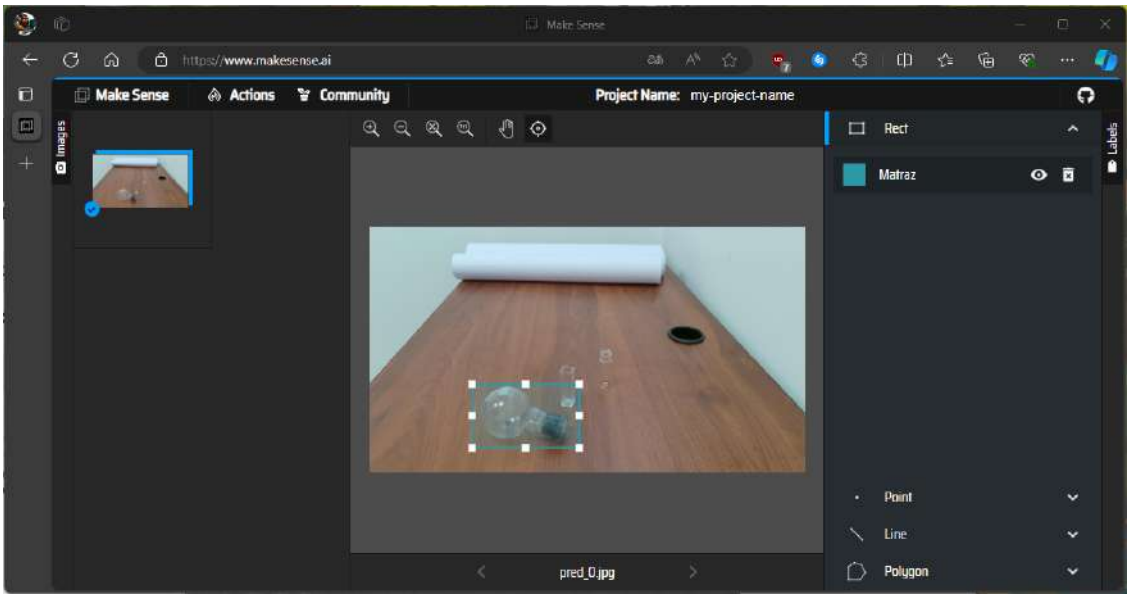


Figura 3.15: Interfaz de MakeSense

- **VGG Image Annotator(VIA)**[10]: Es una herramienta simple e independiente, creado en HTML, javascript y CSS, haciendo que pueda ejecutar en la mayoría de los navegadores como una aplicación sin conexión.

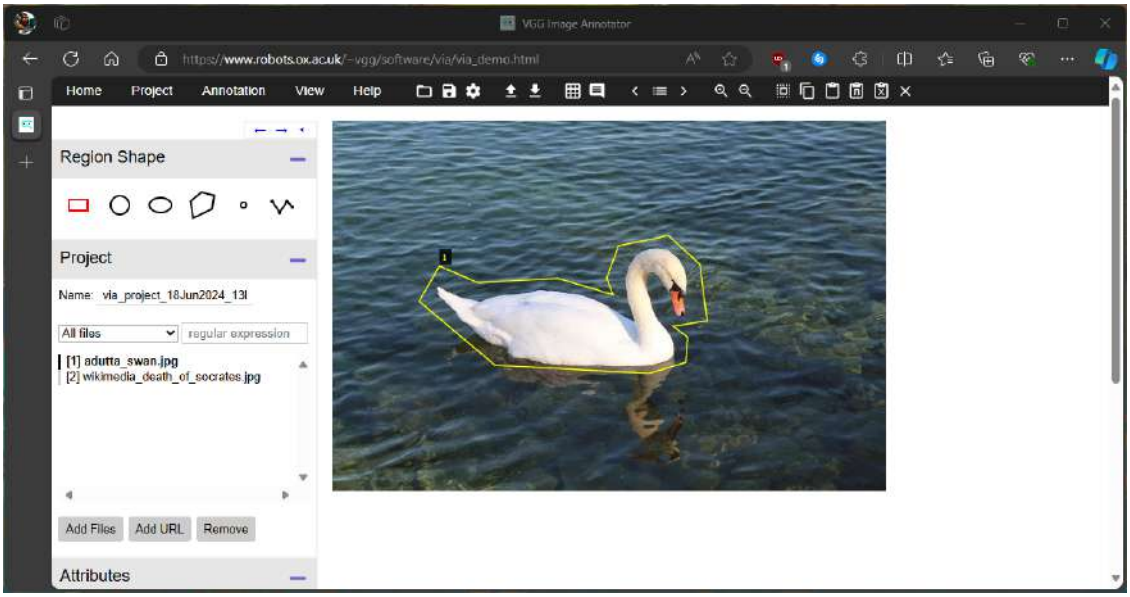


Figura 3.16: Interfaz de VIA

- **Label Studio**[43]: Es una herramienta de anotación de datos de código abierto en la que se pueden realizar diferentes tareas de *machine learning*. Con una interfaz intuitiva, permite a los usuarios realizar anotaciones para clasificación, etiquetado de texto, segmentación de imágenes y más. Además, se integra fácilmente con modelos de *machine learning* para facilitar la predicción de etiquetas y mejora la colaboración entre equipos mediante funciones para revisar y validar anotaciones. Es una solución ideal para equipos que requieren una herramienta robusta y adaptable para sus proyectos de *machine learning*.

Una vez que se etiquetaron los objetos se realiza el preprocesamiento, que es la etapa en la se normalizan el conjunto de imágenes ya sea ajustando el tamaño de las imágenes (640x640 para YOLO-NAS) o aplicando técnicas de aumento de datos para diversificar el conjunto, el estudio realizado por [20] muestran las diferentes técnicas de aumento de datos y que el uso de estas puede llegar a mejorar la precisión entre 2.83 % y un 95.85 %.

Por ultimo se divide el conjunto en tres subconjuntos **entrenamiento**, **prueba**, **validación**. Estos conjuntos permiten evaluar y mejorar la capacidad de generalización del modelo, lo cual es fundamental para su efectividad en la predicción de nuevos datos. Existen varios enfoques para dividir los conjuntos de datos en conjuntos de entrenamiento, prueba y validación. Algunos de los enfoques comunes incluyen:

- División Simple: Se puede utilizar una función para dividir los datos en porcentajes específicos, por ejemplo 70 % para entrenamiento, 15 % para prueba y 15 % para validación.
- División Aleatoria: Se pueden intercambiar aleatoriamente las filas de los datos para crear conjuntos de entrenamiento, prueba y validación.
- División Basada en Propósitos: Se pueden utilizar requisitos específicos del proyecto para dividir los datos de manera efectiva.

La selección del enfoque adecuado depende de la naturaleza del proyecto y los requisitos específicos [44].

2. Configuración del entorno de entrenamiento.

Para entrenar YOLO-NAS es necesario configurar un entorno adecuado que incluya lo siguiente:

- Instalar el lenguaje de programación Python ya que se ha convertido en el lenguaje de programación dominante para el Deep Learning debido a su simplicidad, versatilidad, amplia biblioteca de soporte, comunidad grande y activa, y capacidad para integrarse con otras herramientas.
- Instalar los paquetes necesarios para el entrenamiento de la CNN como son *PyTorch*, *openCV*, *supergradients*, *CUDA*, etc.
- Importar las librerías requeridas y establecer el directorio de trabajo y el nombre del experimento así como instanciar el modelo de YOLO-NAS que se entrenará con el conjunto propio.

3. Entrenamiento del modelo.

Antes de entrenar el modelo, es necesario configurar los parámetros. Si se usa la librería *supergradients*, la función de entrenamiento requiere ciertos parámetros obligatorios:

- **max_epochs**: Número máximo de épocas de entrenamiento.
- **loss**: La función de pérdida que desea utilizar.
- **optimizer**: Optimizador que va a utilizar.
- **train_metrics_list, valid_metrics_list**: Métricas a registrar durante el entrenamiento.
- **metric_to_watch**: Métrica según la cual se guardará el punto de control del modelo.

Los optimizadores son algoritmos utilizados en el entrenamiento de redes neuronales para ajustar los pesos de la red y mejorar su rendimiento. Su objetivo es minimizar la función de pérdida. En la tabla 3.1 se muestran optimizadores que se pueden usar en el entrenamiento.

Optimizador	Características
Adam	Convergencia rápida y estable, adecuado para una amplia gama de problemas.
AdamW	Regularización de peso decay para evitar el sobreajuste.
SGD	Simple y eficaz, pero puede ser lento y susceptible a los mínimos locales.
Lion	Efectivo para problemas con gradientes ruidosos o no uniformes.
RMSprop	Adapta la tasa de aprendizaje para cada parámetro de peso.

Tabla 3.1: Optimizadores que se pueden usar en el entrenamiento.

4. Evaluación del modelo.

Evaluar el entrenamiento de una red neuronal convolucional es esencial para garantizar su desempeño en datos nuevos. El objetivo es encontrar un equilibrio entre precisión y generalización, evitando tanto el sobreajuste (cuando el modelo memoriza los datos de entrenamiento) como el subajuste (cuando no aprende las características clave). Durante el entrenamiento, se utilizan diversas técnicas para medir la eficacia del modelo y ajustar sus parámetros. Estas incluyen:

Curvas de pérdida (*loss*) y precisión (*accuracy*)

Las curvas de pérdida y precisión son herramientas para evaluar el rendimiento de un modelo de una CNN ya que proporcionan información sobre cómo el modelo progresa en la tarea de aprendizaje y permiten identificar posibles problemas de entrenamiento, como el sobreajuste o el subajuste. Mientras la curva de pérdida representa la diferencia promedio entre las predicciones del modelo y los valores reales durante el entrenamiento la curva de precisión representa el porcentaje de predicciones correctas realizadas por el modelo en cada iteración del entrenamiento.

Un modelo sobreajustado muestra una disminución de la pérdida en el conjunto de entrenamiento, pero un aumento en el conjunto de validación. Este comportamiento sugiere que el modelo está memorizando los datos de entrenamiento en lugar de aprender las características generales subyacentes, lo que lleva a un buen rendimiento en el conjunto de entrenamiento pero a un mal rendimiento en nuevos datos no vistos. En contraste, un modelo subajustado generalmente presenta una pérdida relativamente alta tanto en el conjunto de entrenamiento como en el conjunto de validación. Esto indica que el modelo no está capturando suficientemente las tendencias en los

datos, lo que resulta en un mal rendimiento general.

Un modelo bien entrenado se caracteriza por una disminución constante de la pérdida junto con un aumento constante de la precisión a lo largo de las iteraciones. Esto indica que el modelo está aprendiendo efectivamente y generalizando correctamente a los datos.

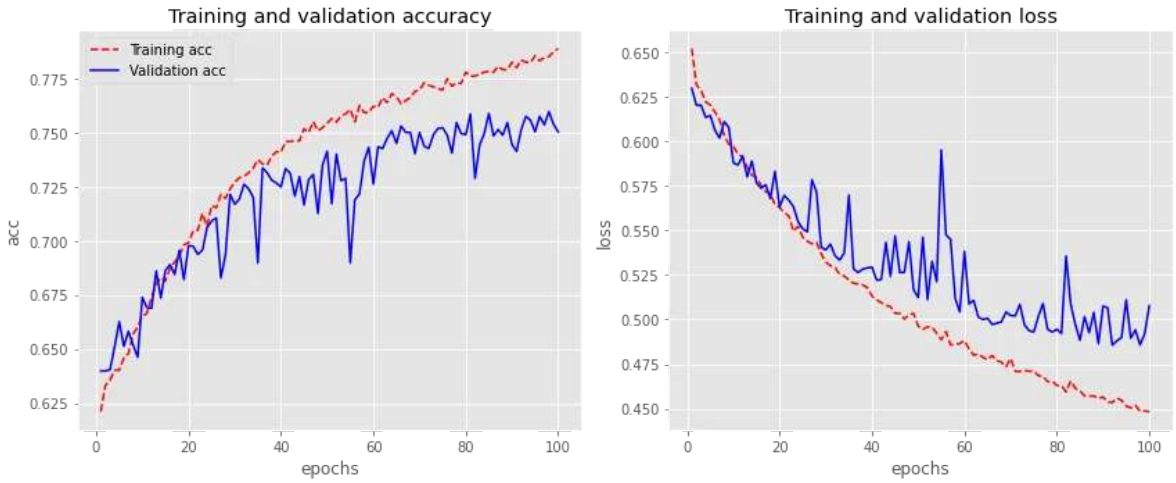


Figura 3.17: Ejemplo de curvas de precisión y perdida en un modelo bien entrenado

Matrices de confusión

Una forma mucho mejor de evaluar el rendimiento de un clasificador es observar la matriz de confusión. Esta matriz proporciona una representación visual y cuantitativa de cómo el modelo clasifica los diferentes tipos de ejemplos en un conjunto de imágenes. La matriz de confusión se organiza en filas y columnas, donde cada fila representa la clase real a la que pertenecen los ejemplos y cada columna representa la clase predicha por el modelo.

- Celdas diagonales: Representan el número de ejemplos correctamente clasificados en cada clase.
- Celdas fuera de la diagonal: Representan el número de ejemplos mal clasificados, indicando la clase real a la que pertenecen y la clase a la que fueron asignados erróneamente por el modelo.

El análisis de la matriz de confusión permite identificar patrones y errores de clasificación del modelo:

- Altos valores en las celdas diagonales: Indican un buen rendimiento general del modelo en la clasificación de ejemplos.
- Altos valores en las celdas fuera de la diagonal: Indican errores de clasificación específicos, revelando qué clases son más propensas a ser confundidas por el modelo.
- Análisis por clase: Permite evaluar el rendimiento del modelo para cada clase individualmente, identificando posibles sesgos o debilidades en la clasificación de ciertas categorías.

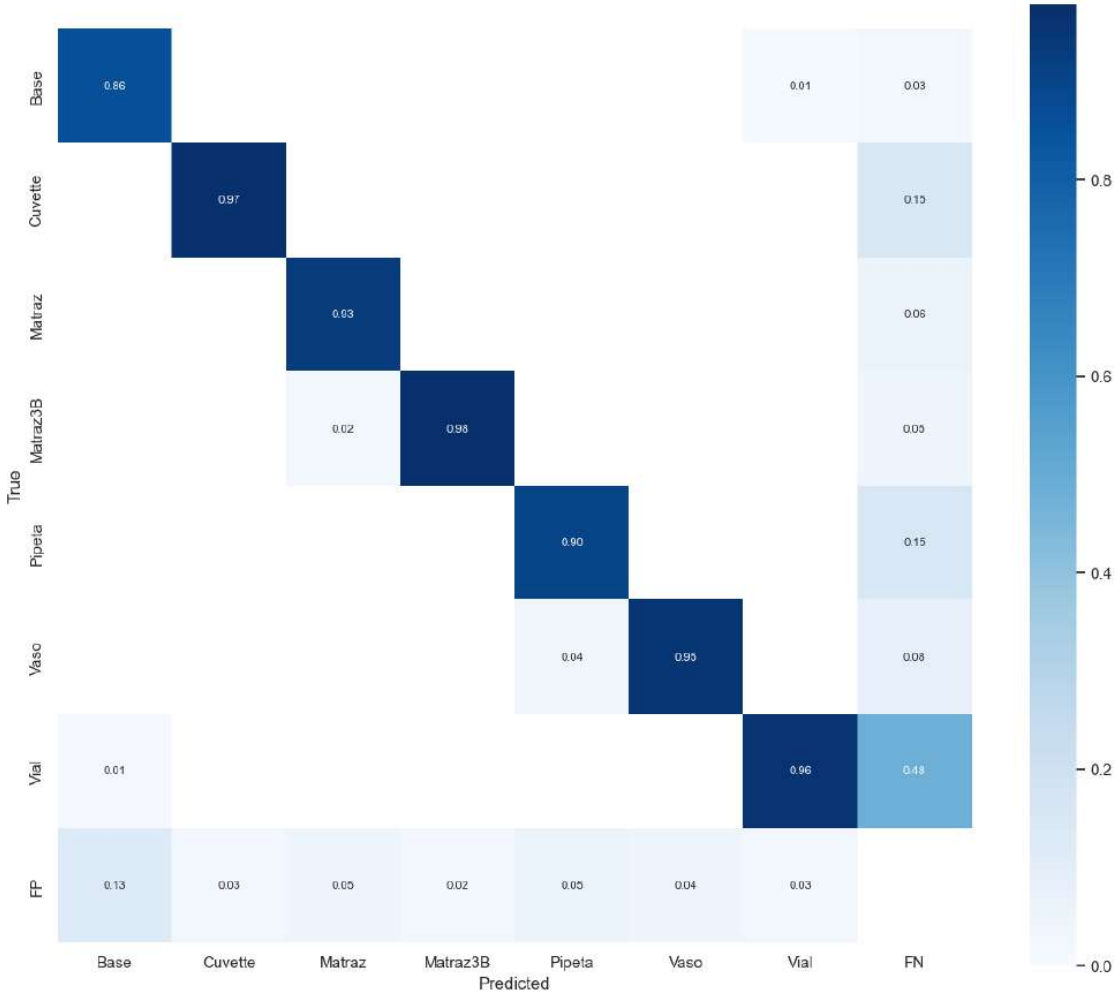


Figura 3.18: Matriz de confusión para evaluar el modelo de identificación de objetos de un laboratorio de síntesis de materiales

Métricas de evaluación

A partir de los valores de la matriz de confusión resultados verdaderos positivos (TP), falsos positivos(FP), falsos negativos(FN), verdaderos negativos (TN), se pueden calcular diversas métricas que proporcionan información detallada sobre el rendimiento del modelo:

- ***Precision***: es la métrica más utilizada por investigadores en el caso dicotómico y multiclase, debido a su facilidad de cálculo y comprensión para evaluar la efectividad general del algoritmo. Sin embargo una de sus principales desventajas es que produce menos valores discriminatorios y distintivos para el caso multiclase desbalanceado.[1]

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

Figura 3.19: Formula de *precision*

- **Recall**: es una medida que permite conocer la proporción de casos positivos que fueron correctamente clasificados. En un modelo perfecto el *recall* es igual a 1 para cada clase.[1]

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

Figura 3.20: Formula de *recall*

- ***F1-score***: fusiona las métricas *precision* con *recall*, presentando diferencias en el rendimiento de un clasificador que no son revelados únicamente con la *precision*. Es directamente proporcional al aumento de las dos medidas, por lo tanto valores altos de *F1-score* demuestra que el algoritmo de clasificación predice de mejor manera la clase positiva.[1]

$$Recall = \frac{TP}{TP+\frac{FN+FP}{2}}$$

Figura 3.21: Formula de *F1-score*

- ***Mean Average Precision (m*AP)**: Una métrica muy utilizada en las tareas de detección de objetos es la precisión media (*m*AP). Para entender esta métrica, debemos saber que cuanto

mayor sea el *recall*, es posible que la *precision* se menor y viceversa. Esto se puede visualizar en una curva de *Precision/Recall* (Figura 3.22). Se debe tener en cuenta que la curva de *Precision/Recall* puede contener algunas secciones en las que la *precision* aumenta cuando lo hace el *recall*, especialmente en valores bajos de *recall*. Este es uno de los motivos de la métrica *m*AP.

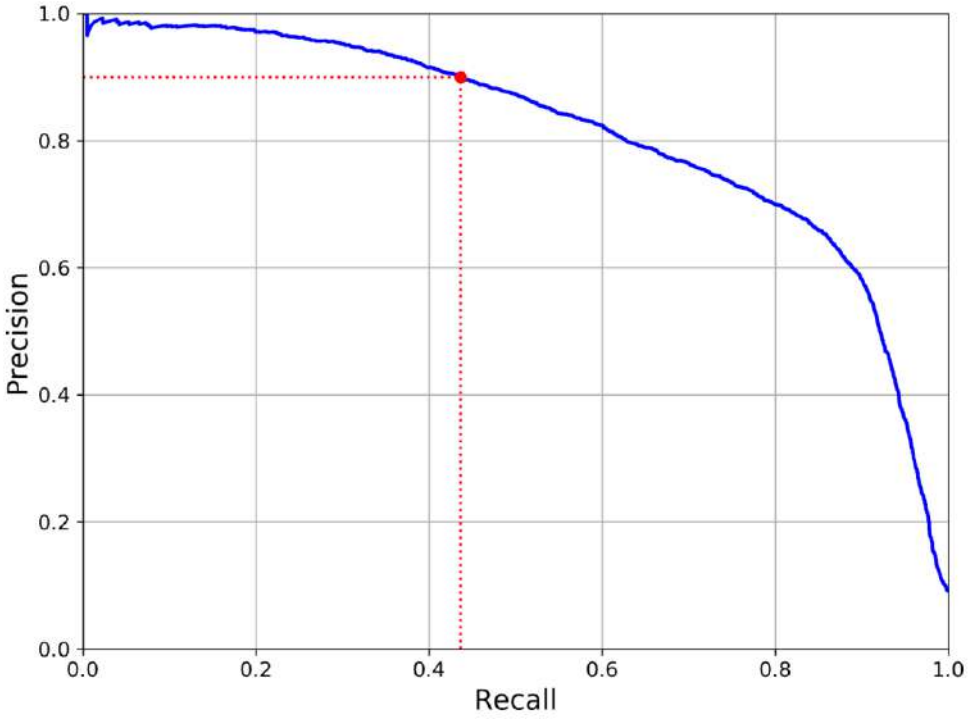


Figura 3.22: Curva de *Presicion/Recall*

El funcionamiento de la métrica es el siguiente:

- El modelo analiza una imagen produce una salida con una serie de cajas delimitadoras (*bounding boxes*) de los objetos detectados dentro de la imagen y la probabilidad de que cada objeto detectado pertenezca a una determinada clase.
- Se compara tanto las cajas delimitadoras como las clases de la salida con las verdaderas.
- Se calcula el **IoU** (*Intersection over Union*) que mide qué tan bien se superponen las cajas delimitadoras (predichas y verdaderas), para esto se establece un umbral (por ejemplo,, 0.5). Si el **IoU** es mayor a este umbral, se considera que la deteccion es correcta.

- Para cada clase, se construye una curva de *Presicion/Recall*. El área bajo la curva de *Presicion/Recall* para cada clase representa el AP (*Average Precision*). Un AP más alto indica un mejor rendimiento para esa clase en particular.
- Finalmente, se calcula el promedio de los valores de AP para todas las clases. Este valor, el *m*AP, representa el rendimiento general del modelo en todas las clases de objetos.

Ejemplo

Supongamos que se tiene un modelo de detección de objetos entrenado para clasificar perros, gatos y automóviles. La precisión media de cada clase se muestra en la tabla 3.2.

Clase	Average Precision (AP)
Perro	0.85
Gato	0.92
Automóvil	0.78

Tabla 3.2: Precisión media de las clases identificadas por el modelo de ejemplo

En este ejemplo, la *m*AP sería (0.85 + 0.92 + 0.78) / 3 = 0.85. Esto significa que, en promedio, el modelo es capaz de detectar correctamente los objetos con una precisión del 85 %.

*m*AP proporciona una medida integral del rendimiento de un modelo de detección de objetos, considerando tanto la *precision* como el *recall* para todas las clases de objetos presentes en el conjunto de datos. Un mAP más alto indica un mejor modelo.

Regularización

La restricción de un modelo para simplificarlo y reducir el riesgo de sobreajuste se denomina regularización. El sobreajuste ocurre cuando un modelo se ajusta demasiado bien a los datos de entrenamiento lo que resulta en un rendimiento pobre en datos no vistos (de prueba o validación).[14]

Existen diversas técnicas de regularización, cada una con sus propias ventajas y desventajas. Algunas de las técnicas más comunes incluyen:

- **Regularización L1 (LASSO)**:Agrega una penalización a la suma de los valores absolutos de los pesos del modelo. Esto anima al modelo a utilizar un menor número de pesos, lo que simplifica el modelo y reduce el riesgo de sobreajuste.

- **Regularización L2 (Ridge)**:Agrega una penalización a la suma de los cuadrados de los valores de los pesos del modelo. Esta técnica también promueve la simplicidad del modelo, pero de una manera más suave que la regularización L1.

- **Elastic Net**: Combina las regularizaciones L1 y L2, permitiendo controlar la influencia de cada una.

- **Dropout**: Desactiva aleatoriamente una parte de las neuronas en la red neuronal durante el entrenamiento. Esto obliga al modelo a aprender dependencias más robustas entre las neuronas restantes y reduce el sobreajuste.

- **Early stopping**: Detiene el entrenamiento del modelo antes de que comience a sobreajustarse a los datos de entrenamiento. Esto se logra monitoreando la pérdida del modelo en un conjunto de validación y deteniendo el entrenamiento cuando la pérdida comienza a aumentar.

Al aplicar técnicas de regularización adecuadas, se logra mejorar la generalización del modelo. Además, la regularización aumenta la robustez del modelo, haciéndolo menos susceptible al ruido y a las variaciones presentes en los datos de entrenamiento. Esto se traduce en modelos más confiables y menos propensos a errores en situaciones reales.

3.3. Ubicación espacial de objetos

La ubicación espacial de objetos es un área fundamental en la visión artificial, que se encarga de determinar la posición y la distancia de los objetos en un entorno tridimensional. Esta información es crucial para diversos campos, incluyendo la robótica, la realidad aumentada, la interacción humano-computadora y la conducción autónoma. Existen diversos métodos para la localización espacial de objetos, cada uno con sus propias ventajas y desventajas. Los métodos más comunes se pueden clasificar en dos categorías principales:

Basados en sensores de profundidad

- **LiDAR**: acrónimo de *Light Detection and Ranging*, es una tecnología de detección remota que utiliza rayos láser para medir distancias y crear mapas 3D detallados del entorno. Funciona según el principio de emitir pulsos láser y medir el tiempo que tardan los pulsos en regresar tras chocar con objetos del entorno. Analizando el tiempo de vuelo de los pulsos láser, los sensores LiDAR pueden determinar con precisión las distancias a los objetos y generar mapas de profundidad de alta resolución. La tecnología LiDAR ha encontrado aplicaciones en diversos campos, como los vehículos autónomos, la robótica, la vigilancia medioambiental, la arqueología y la planificación urbana. Proporciona valiosa información espacial y mejora la percepción y comprensión del mundo físico.[12]

  El LiDAR funciona mediante la emisión de pulsos láser y la medición del tiempo que tardan en regresar tras interactuar con objetos en el entorno. Este proceso permite a los sensores LiDAR determinar distancias con precisión y crear mapas 3D detallados.

  El sensor LiDAR emite pulsos cortos de luz láser, generalmente en la gama del infrarrojo cercano, hacia la escena de interés. Estos pulsos se reflejan al encontrarse con objetos y regresan al sensor. Los detectores del LiDAR captan los pulsos reflejados y miden el tiempo de vuelo con alta precisión, utilizando componentes electrónicos avanzados.

  Con la velocidad de la luz conocida, el tiempo de vuelo se traduce en distancia mediante una fórmula sencilla: velocidad de la luz multiplicada por el tiempo y dividida por dos, considerando el viaje de ida y vuelta del pulso láser. Además de la distancia, el LiDAR también registra la intensidad de los pulsos reflejados, proporcionando información sobre las características de las superficies.

  Al escanear en diferentes ángulos y direcciones, el LiDAR genera una nube de puntos, cada uno con coordenadas tridimensionales específicas, formando un mapa 3D del entorno. Este método es esencial para aplicaciones en cartografía, navegación y detección de objetos, ya que ofrece información espacial precisa y detallada.[12]

- **Cámaras 3D**: Las cámaras 3D, también conocidas como cámaras estereoscópicas, son dispositivos capaces de capturar imágenes en tres dimensiones. Imitan la visión binocular hu-

mana, tomando dos fotografías ligeramente diferentes de la misma escena desde perspectivas ligeramente diferentes, al igual que nuestros ojos. Estas imágenes se combinan luego para crear una sensación de profundidad y realismo al ser visualizadas.

Existen dos métodos principales para adquirir imágenes para sistemas de visión estereoscópica:

- Generación por ordenador: Las imágenes se crean completamente de forma digital utilizando software 3D. Este método es adecuado para imágenes fijas y animaciones, pero no funciona bien con imágenes del mundo real.
- Captura de imágenes del mundo real: Este método implica la captura de fotografías reales utilizando cámaras estereoscópicas. Es el método que nos ocupa y se realiza mediante técnicas de captura directa y mejora por ordenador de fotografías 2D individuales.

Una cámara estereoscópica consta de dos o más cámaras montadas juntas, separadas por una distancia similar a la distancia entre los ojos humanos. Cada cámara captura una imagen de la escena desde su propia perspectiva. La diferencia de perspectiva entre las imágenes crea un efecto de paralaje, que es la clave para la percepción de la profundidad.[29]

Basados en visión por computadora

- **Reconocimiento de objetos 3D**: El reconocimiento de objetos 3D se ha convertido en una herramienta poderosa para esta tarea, ya que proporciona información semántica sobre la escena que puede complementar la información geométrica tradicional. Existen dos enfoques principales para la estimación de profundidad usando reconocimiento de objetos 3D:
  - Basado en modelos: Este enfoque utiliza modelos 3D preentrenados de objetos para estimar la profundidad de los objetos presentes en la escena. Los métodos basados en modelos suelen ser computacionalmente eficientes y robustos a variaciones de iluminación y textura. Sin embargo, pueden ser sensibles a la oclusión y a la falta de modelos 3D precisos para todos los objetos de interés.

- Basado en el aprendizaje profundo: Este enfoque utiliza redes neuronales convolucionales (CNNs) para aprender a estimar la profundidad directamente a partir de imágenes. Los métodos basados en el aprendizaje profundo pueden ser más precisos que los métodos basados en modelos, especialmente en condiciones desafiantes. Sin embargo, suelen requerir grandes cantidades de datos de entrenamiento y pueden ser computacionalmente más costosos.

Basados en modelos geométricos

Los métodos geométricos aprovechan principios geométricos para estimar la distancia a objetos en una escena a partir de una sola imagen. Estos métodos se basan en la idea de que la geometría de la escena y la cámara utilizada para capturarla están intrínsecamente relacionadas, y que esta relación puede ser explotada para calcular la profundidad. Existen dos tipos principales de métodos geométricos para la estimación de profundidad:

- Métodos basados en la perspectiva: Estos métodos se basan en el principio de perspectiva, que establece que los objetos más cercanos a la cámara se proyectan en la imagen con un tamaño mayor que los objetos más lejanos. Al analizar la relación de tamaño entre objetos conocidos en la imagen, es posible estimar su distancia relativa a la cámara.

- Métodos basados en características: Estos métodos se basan en la identificación de características geométricas específicas en la imagen, como líneas rectas, esquinas o puntos de fuga. Al analizar la distribución y la orientación de estas características, es posible inferir la estructura tridimensional de la escena y, en consecuencia, la profundidad de los objetos.

3.3.1. Calibración de la cámara

La calibración de cámara es un proceso fundamental en visión por computadora que consiste en determinar los parámetros intrínsecos y extrínsecos de una cámara. Estos parámetros son esenciales para corregir las distorsiones presentes en las imágenes capturadas y obtener una representación precisa del mundo real. Las distorsiones geométricas más comunes en las cámaras son la distorsión radial y la distorsión tangencial.

Figura 3.23: Imagen afectada por distorsión de la cámara

La distorsión radial provoca que las líneas rectas en el mundo real se vean curvadas en la imagen, se modela utilizando polinomios de tercer orden y se caracteriza por coeficientes como k_1, k_2 y k_3. Mientras que la distorsión tangencial produce un desplazamiento de los puntos en la imagen, especialmente cerca de los bordes, se modela utilizando polinomios de segundo orden y se caracteriza por coeficientes como p_1 y p_2. Estas distorsiones pueden afectar significativamente la precisión de la detección de objetos en imágenes. Por ejemplo, un objeto rectangular puede parecer deformado en la imagen, lo que dificulta que los algoritmos de detección de bordes lo identifiquen correctamente.[31]

La distorsión radial se puede representar como, de la siguiente manera:

$$X_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$Y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Mientras que la distorsión tangencial se puede representar de la siguiente manera:

$$X_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$

$$Y_{distorted} = y + [2p_2xy + p_1(r^2 + 2x^2)]$$

Calibración de cámara usando *OpenCV*

Se necesitan encontrar cinco parámetros, conocidos como coeficientes de distorsión dados por:

$$Distortion\ coefficients = (k_1, k_2, p_1, p_2, k_3)$$

Además de esto, se requieren los parámetros intrínsecos y extrínsecos de la cámara. Los parámetros intrínsecos son específicos de una cámara, incluyen información como:

- Distancia focal (fx, fy): representa la distancia desde el centro óptico de la lente a la superficie del sensor de la cámara.
- Centros ópticos (cx, cy): representan las coordenadas del punto principal en la imagen, que es la intersección de los ejes óptico e imagen.

La distancia focal y los centros ópticos se pueden usar para crear una matriz de cámara, que se puede usar para eliminar la distorsión debida a las lentes de una cámara específica. La matriz de la cámara es única para una cámara específica, por lo que una vez calculada, se puede reutilizar en otras imágenes tomadas por la misma cámara. Se expresa como una matriz de 3x3:

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Para realizar el proceso de calibración se requieren algunas imágenes de prueba con un patrón bien definido, por ejemplo, un tablero de ajedrez (Fig. 3.24). En este patrón se tiene que encontrar puntos específicos cuyas posiciones relativas ya se conocen. Obteniendo estos punto y sabiendo las coordenadas de estos puntos en el espacio del mundo real se pueden obtener los coeficientes de distorsión. Para obtener mejores resultados, necesitamos al menos 10 patrones de prueba.

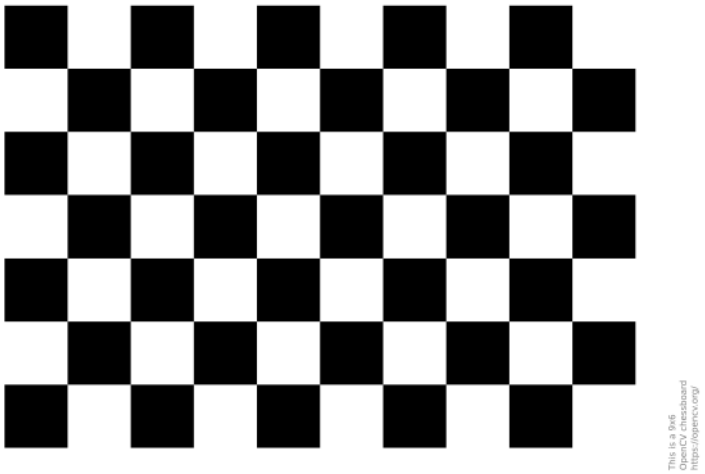


Figura 3.24: Tablero de ajedrez usado para la calibración de la cámara.

Pasos generales del proceso de calibración[31]

- Preparación del tablero de ajedrez: Se requiere imprimir un patrón de tablero de ajedrez de tamaño conocido y colocarlo de tal manera que el patrón sea visible y esté bien iluminado durante la captura de imágenes.

- Captura de imágenes: Capturar al menos 10 imágenes del tablero de ajedrez desde diferentes ángulos y distancias. Es importante que el tablero ocupe una parte significativa de la imagen y que haya suficiente variación en las orientaciones entre las capturas.

- Detección de esquinas: En *OpenCV* se utiliza la función *findChessboardCorners()* para identificar las esquinas del tablero de ajedrez en cada imagen capturada.

- Extracción de puntos: Las coordenadas de las esquinas detectadas en cada imagen representarán los puntos homólogos en el plano de la imagen.

- Calibración: Se usa la función *calibrateCamera()* en *OpenCV*, utilizando las coordenadas de los puntos homólogos y las dimensiones conocidas del tablero de ajedrez. Este algoritmo calculará los parámetros intrínsecos y extrínsecos de la cámara.

- Unidistorsión: Una vez calibrada la imagen se corrige la distorsión radial y tangencial causada por la lente de la cámara usando la función *undistortImage()* en *OpenCV*.

- Evaluación: Para evaluarla precisión de la calibración se calcula error de reproyección, que representa la distancia entre los puntos homólogos reproyectados en la imagen y las esquinas detectadas. Un error de reproyección bajo indica una calibración precisa.

Capítulo 4

Desarrollo

4.1. Descripción del sistema

Para el desarrollo del sistema de reconocimiento y ubicación espacial de objetos se planteo dividir el sistema en dos módulos, uno que identifica los objetos que se utilizan en el experimento seleccionado y otro que una vez identificado el objeto calcula la ubicación dentro del área de trabajo. Ambos módulos se apoyan de la red neuronal convolucional YOLO-NAS la cual ha sido reentrenada con un conjunto de datos que fue creado para este proyecto. En la figura 4.1 se muestra una visión general del sistema de detección y ubicación espacial.

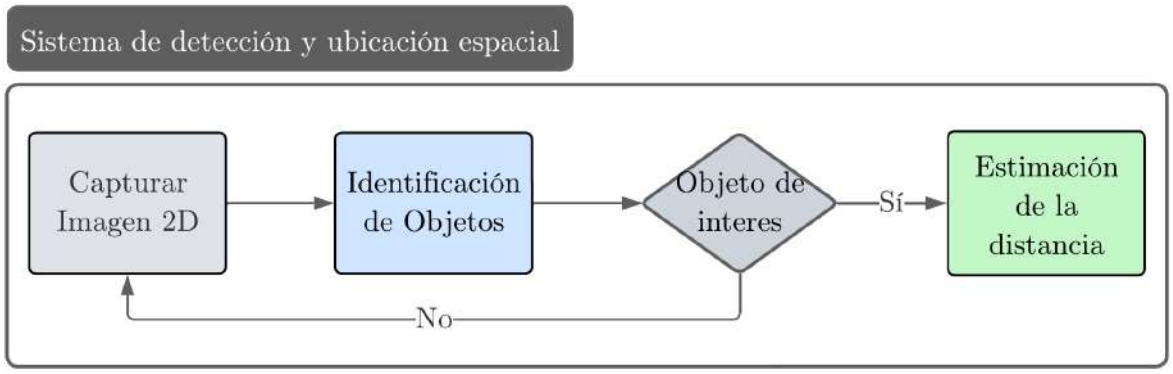


Figura 4.1: Diseño general del sistema de detección y ubicación espacial

4.2. Ubicación de la cámara

Para ubicación de la cámara se consideraron tres ubicaciones principales con la finalidad de no afectar el funcionamiento de los brazos del robot colaborativo (figura 4.2) y considerando un área de trabajo delimitada de 50 cm de ancho por 52 cm de profundidad.

- Base del área de trabajo: Esta posición ofrece una vista panorámica y estable del espacio de trabajo. Sin embargo, puede verse afectada por sombras proyectadas por el robot o elementos cercanos, y la perspectiva puede no ser la más adecuada para detectar objetos del fondo si están siendo bloqueados por otros.

- Toma desde arriba: Ubicar la cámara en una posición elevada proporciona una vista aérea que minimiza las oclusiones y permite una fácil identificación de los objetos. No obstante, puede limitar la detección de objetos muy cercanos al suelo-

- En el cuerpo de YuMi: Esta opción busca un equilibrio entre la flexibilidad y la no interferencia con los movimientos del robot. Sin embargo, es fundamental elegir un punto de fijación que no obstruya que no afecte el movimiento de los brazos.

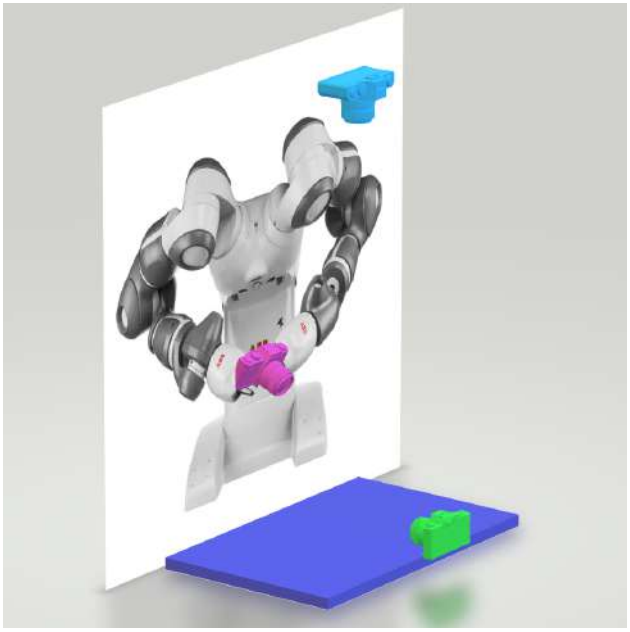
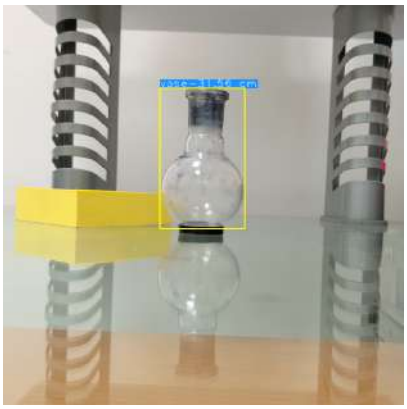


Figura 4.2: Diagrama de posiciones consideradas para la ubicación de la cámara en el robot Yumi. Base del área de trabajo (verde), toma desde arriba (azul), en el cuerpo de Yumi (rosa)



(a) Base del área de trabajo

(b) Toma desde arriba

(c) En el cuerpo de YuMi

Figura 4.3: Posiciones consideradas para la ubicación de la cámara. Base del área de trabajo (a), toma desde arriba (b), en el cuerpo de Yumi (c)

De las tres ubicaciones consideradas se optó por integrar la cámara directamente en el cuerpo de YuMi. Esta decisión se fundamenta en una serie de ventajas. Al estar montada en el cuerpo del robot la proximidad de la cámara a la zona de agarre proporciona una visión detallada de los objetos que el robot está manipulando dentro del espacio de 50x52 cm. Esta información visual es crucial para realizar tareas que requieren una alta precisión. Asimismo, la integración de la cámara en el cuerpo del robot permite una apariencia más limpia y estética, evitando la necesidad de cables o soportes adicionales que puedan interferir con el movimiento de YuMi dentro del espacio de trabajo asignado.

4.3. Detección de objetos

Las redes neuronales convolucionales (CNNs) son una herramienta poderosa para la detección de objetos en imágenes. Sin embargo, para que una CNN funcione correctamente, es necesario entrenarla con un conjunto de imágenes que representen adecuadamente las clases de objetos que se quieren detectar.

4.3.1. Creación de conjunto de datos

La elaboración de un conjunto de imágenes personalizados implica realizar una serie de pasos que van desde la recopilación y etiquetado de imágenes, hasta el aumento de datos y evaluación final del conjunto.

Recopilación de imágenes

En esta etapa se capturaron imágenes a color de los objetos seleccionados usando una cámara 2D con una resolución de captura de 1080x1920 píxeles obteniendo un total de 708 imágenes.

Etiquetado de imágenes

Una vez capturadas las imágenes se procedió al etiquetado manual de los objetos en las imágenes usando el software de código abierto *Label Studio*[43], una herramienta de gran ayuda para etiquetar imágenes de manera sencilla como se muestra en la figura 4.4.

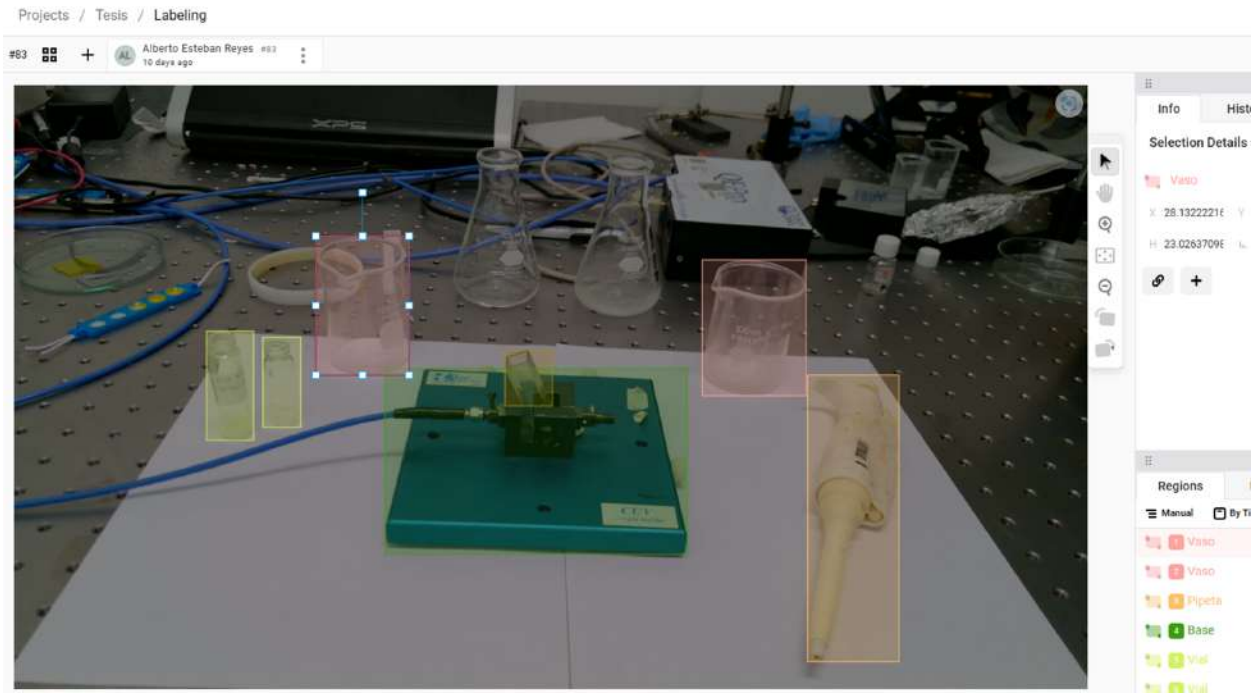
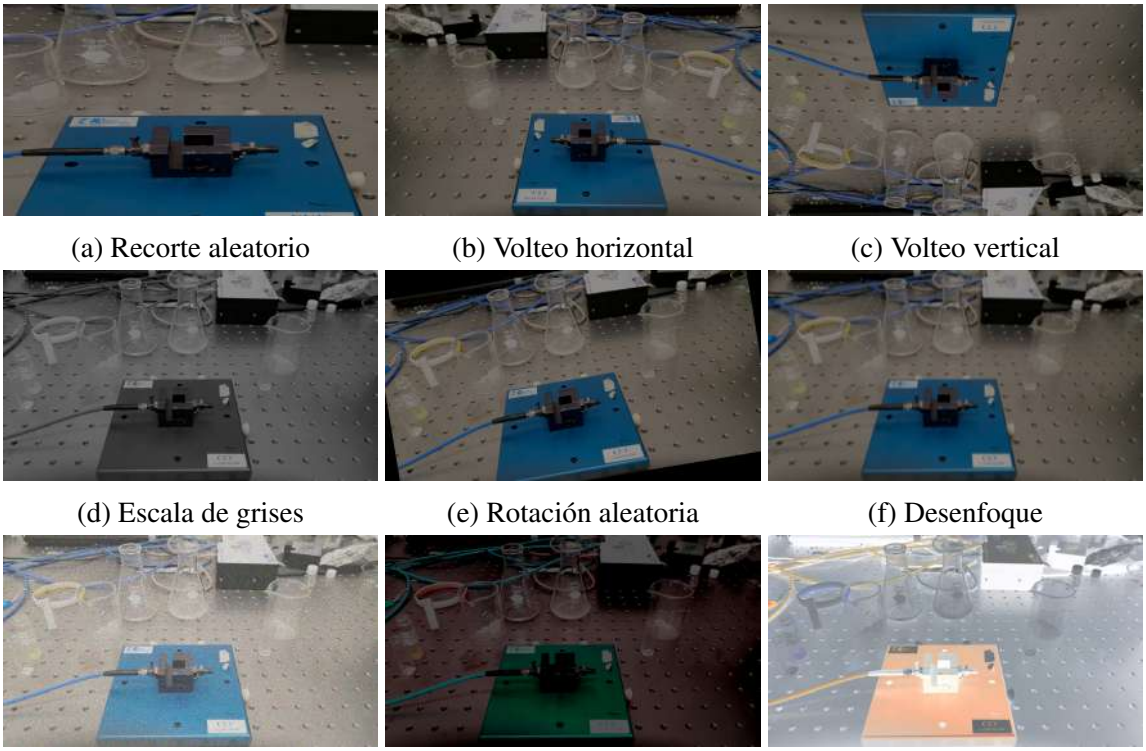


Figura 4.4: Etiquetado con el software *Label Studio*

Aumento de datos

El conjunto de imágenes generado fue muy reducido ya que con solo 809 imágenes para el entrenamiento de la CNN no se aseguran resultados óptimos. Por lo que se recurrió a utilizar técnicas de aumento de datos, se utilizaron las transformaciones geométricas y transformaciones de espacio de color. Estas técnicas fueron implementadas en *Python* con la ayuda de la librería *albumentations* [2], las operaciones que se realizaron son recorte aleatorio, volteo horizontal y vertical, rotación aleatoria, escala de grises, desenfoque aleatorio, inserción de ruido aleatorio, color aleatorio y por último inversión de color. Con esta implementación se obtuvieron 6777 imágenes, en la figura 4.5 se muestran las transformaciones usadas.



(a) Recorte aleatorio (b) Volteo horizontal (c) Volteo vertical

(d) Escala de grises (e) Rotación aleatoria (f) Desenfoque

(g) Ruido aleatorio (h) Color aleatorio (i) Invertir color

Figura 4.5: Imágenes generadas usando transformaciones geométricas y de espacio de color a partir de la imagen original.

Luego de aplicar técnicas de aumento de datos, el conjunto de imágenes resultante era lo suficientemente grande para volver a entrenar la red neuronal YOLO-NAS. Sin embargo, se detectó un desequilibrio en la distribución de las clases, siendo la clase *vaso* la más afectada (ver figura 4.5). Este desbalance puede llevar a que el modelo se 'enfoque' demasiado en la clase mayoritaria, lo que se conoce como sobreajuste. Para solucionar este problema, se empleó una técnica de submuestreo que consiste en reducir el número de muestras de la clase *vaso*, logrando así una distribución de clases más equilibrada (ver figura 4.7).

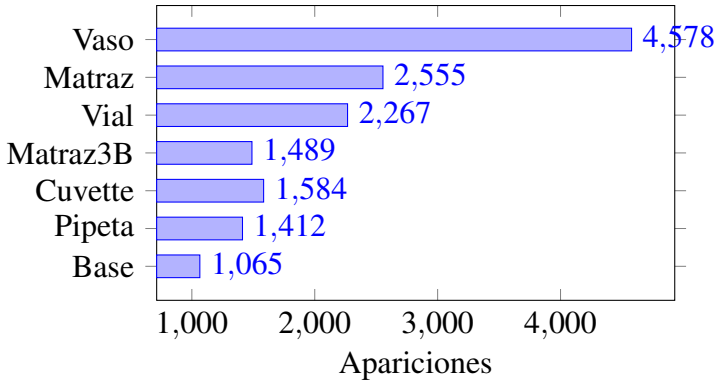


Figura 4.6: Distribución de clases en el conjunto

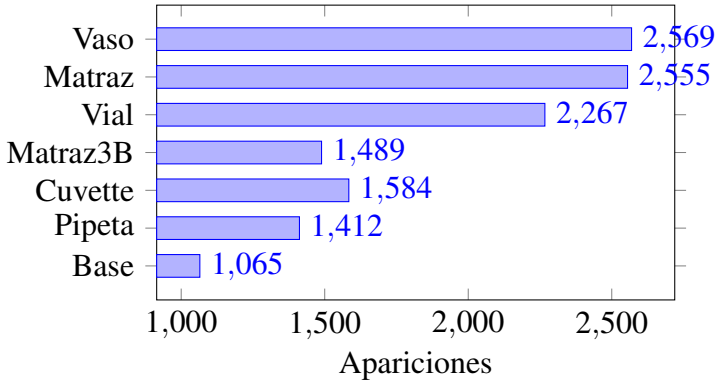


Figura 4.7: Distribución de clases en el conjunto después de aplicar la técnica de submuestreo

Reentrenamiento de CNN

Una vez generado el conjunto se preprocesan las imágenes reduciendo el tamaño de cada imagen a 512x512 píxeles, esto para reducir el tiempo de reentrenamiento de YOLO-NAS. También se dividió el conjunto con la distribución que normalmente se usa, 70 % para entrenamiento, 20 % para validación y 10 % para prueba. La distribución de las imágenes se muestra en la gráfica 4.8.

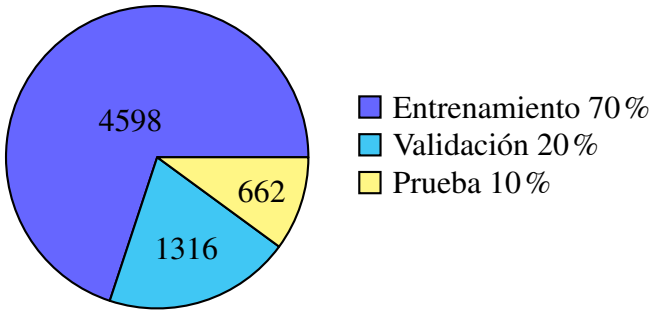


Figura 4.8: Distribución de las imágenes para el entrenamiento.

El reentrenamiento se realizó usando el lenguaje *Python* con la ayuda de la librería *supergradients* usando la configuración que se muestra en el código 1 en un equipo que cuenta con GPU para reducir el tiempo de reentrenamiento.

```
train_params = {
        'silent_mode': True,
        "average_best_models":True,
        "warmup_mode": "linear_epoch_step",
        "warmup_initial_lr": 1e-6,
        "lr_warmup_epochs": 3,
        "initial_lr": 5e-4,
        "lr_mode": "cosine",
        "cosine_final_lr_ratio": 0.1,
        "optimizer": "Adam",
        "optimizer_params": {"weight_decay": 0.0001},
        "zero_weight_decay_on_bias_and_bn": True,
        "ema": True,
        "ema_params": {"decay": 0.9, "decay_type": "threshold"},
        "max_epochs": 20,
        "mixed_precision": True,
        "loss": PPYoloELoss(
            use_static_assigner=False,
            num_classes=len(dataset_params['classes']),
            reg_max=16
        ),
        "valid_metrics_list": [
            DetectionMetrics_050(
                score_thres=0.1,
                top_k_predictions=300,
                num_cls=len(dataset_params['classes']),
                normalize_targets=True,
                post_prediction_callback=PPYoloEPostPredictionCallback(
                    score_threshold=0.01,
                    nms_top_k=1000,
                    max_predictions=300,
                    nms_threshold=0.7
                )
            )
        ],
        "metric_to_watch": 'mAP@0.50'
    }
```

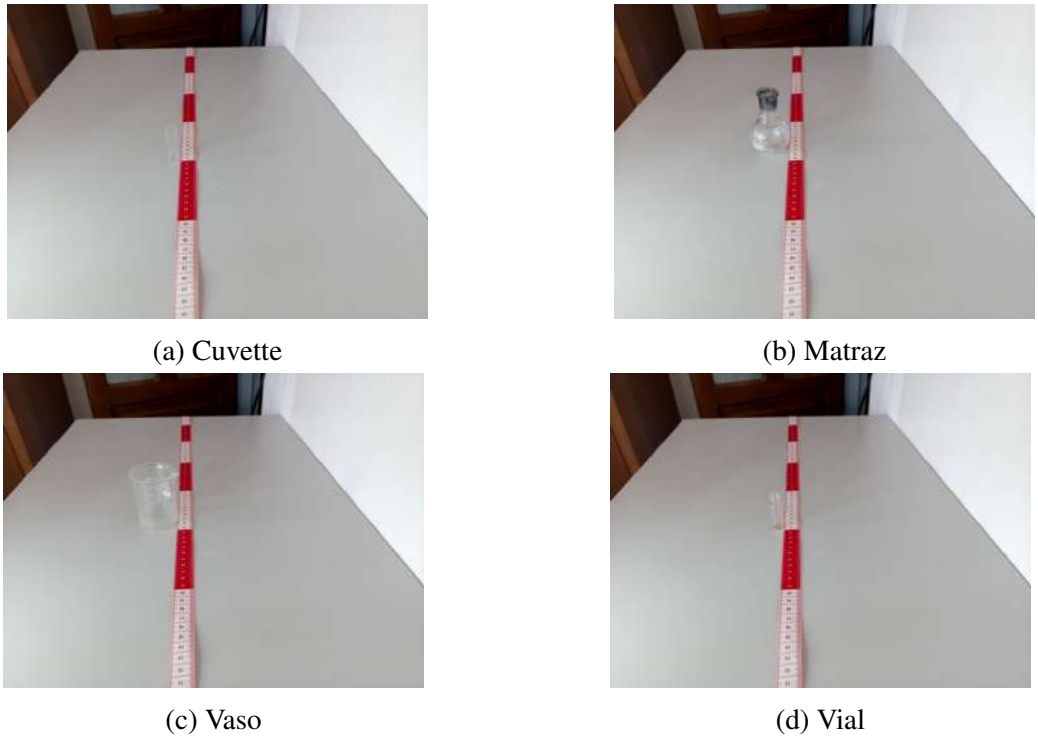
Código 1: Parámetros con los que se entreno la CNN, hay parámetros obligatorios que se describen en el capitulo anterior y los restantes se pueden consultar en [6]

4.4. Estimación de la distancia

Una vez identificado el objeto, la siguiente tarea es conocer la profundidad a la que se encuentra dicho objeto. Para llevar a cabo esta tarea, utilizamos la *similitud del triangulo* para lo cual se necesita tener la medida del ancho del objeto (W) y la distancia de referencia al objeto (D). Al tomar la captura del objeto para guardarlo en la distancia de referencia, se obtiene también el ancho en píxeles del objeto (P), una vez obtenidos los valores se calcula la distancia focal usando la siguiente formula:

$$F = \frac{(P \times D)}{W}$$

Se desarrollo un módulo en *Python* el cual con la ayuda del modelo de detección de objetos se encargo de capturar las imágenes a la distancia de referencia la cual fue de 30cm de 4 objetos del laboratorio de síntesis de materiales (figura 4.9) . Se obtiene también el ancho en píxeles, y se realiza el calculo de la distancia focal, guardándolo en un archivo de configuración para su posterior uso. Los resultados obtenidos se muestran en la tabla 4.1.



(a) Cuvette (b) Matraz

(c) Vaso (d) Vial

Figura 4.9: Imágenes de distancia de referencia.

Una vez registradas las distancias focales se puede obtener una estimación de distancia usando

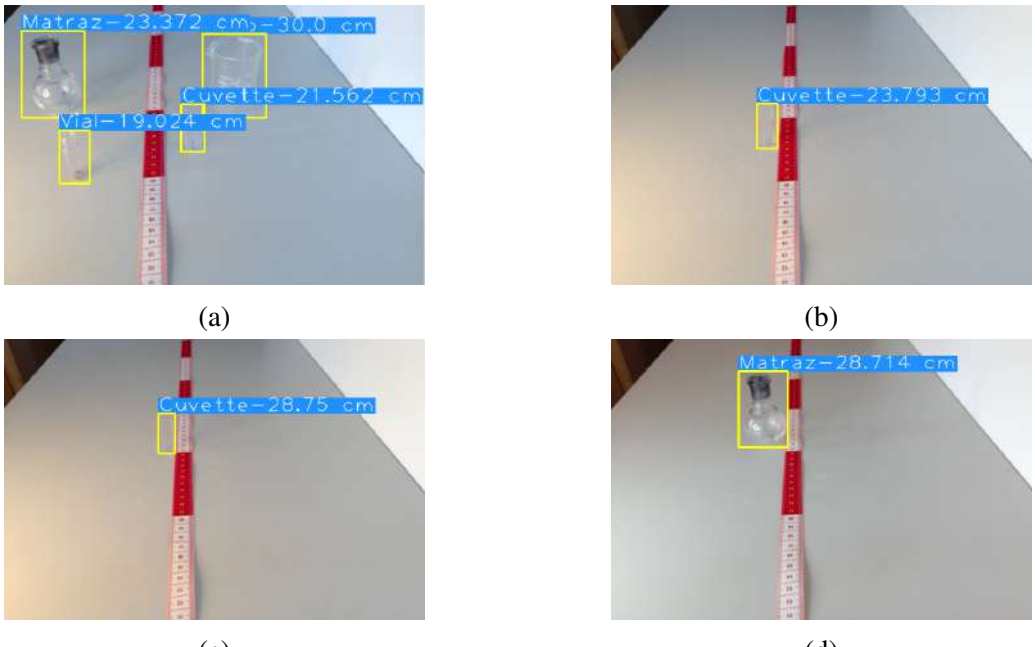
la formula:

$$D' = \frac{(W \times F)}{P}$$

Objeto	Distancia de referencia	Ancho	Ancho píxeles	Distancia Focal
Cuvette	30cm	1.5cm	23px	460.0
Matraz	30cm	5cm	64px	402.0
Vaso	30cm	5.8cm	87px	450.0
Vial	30cm	1.5cm	26 px	52.0

Tabla 4.1: Cálculo de distancia focal.

Sabiendo esto, se añadió al modulo de detección de objetos el calculo de la distancia de los objetos siempre y cuando se tenga registrada la distancia focal del objeto detectado. Algunos ejemplos de estimación se muestran en la figura 4.10.



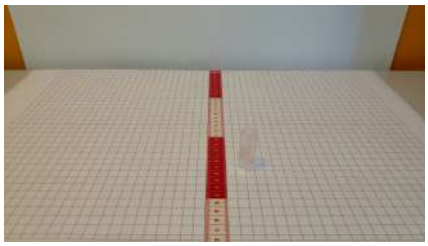
(a) (b)

(c) (d)

Figura 4.10: Ejemplos de estimación de distancia.

Una vez obtenidos los datos de las estimaciones iniciales de distancia, se procede a realizar una nueva simulación. Este nuevo cálculo se lleva a cabo en un entorno que tenga las mismas dimensiones del área de trabajo del robot colaborativo. De esta manera, se puede evaluar si las

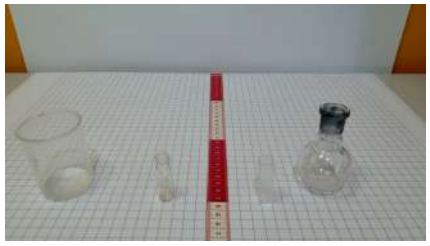
estimaciones de distancia son las más adecuadas para garantizar un funcionamiento óptimo del robot en su entorno de trabajo.



(a) Cuvette

(b) Matraz

(c) Vaso



(d) Vial

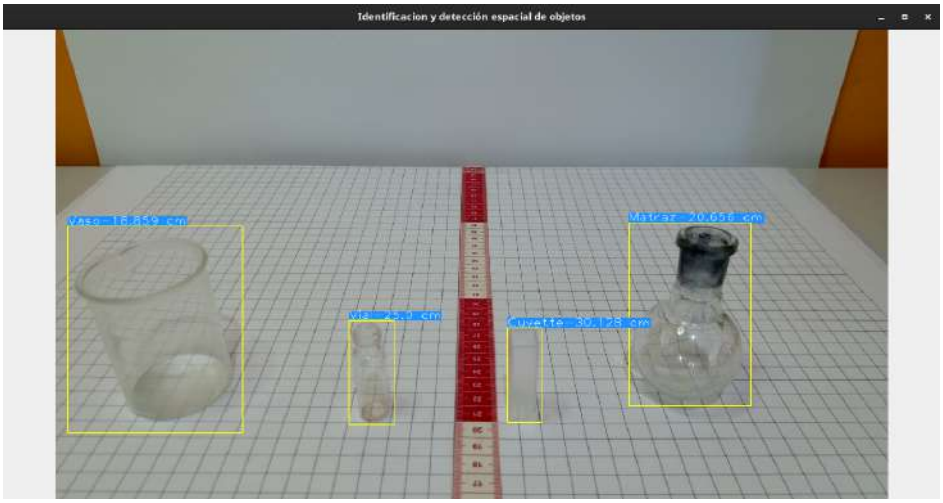
Figura 4.11: Imágenes de distancia de referencia en un área de trabajo igual a la del robot colaborativo.

Se realiza un nuevo calculo de distancia focal obteniendo los resultados que se muestran en la tabla

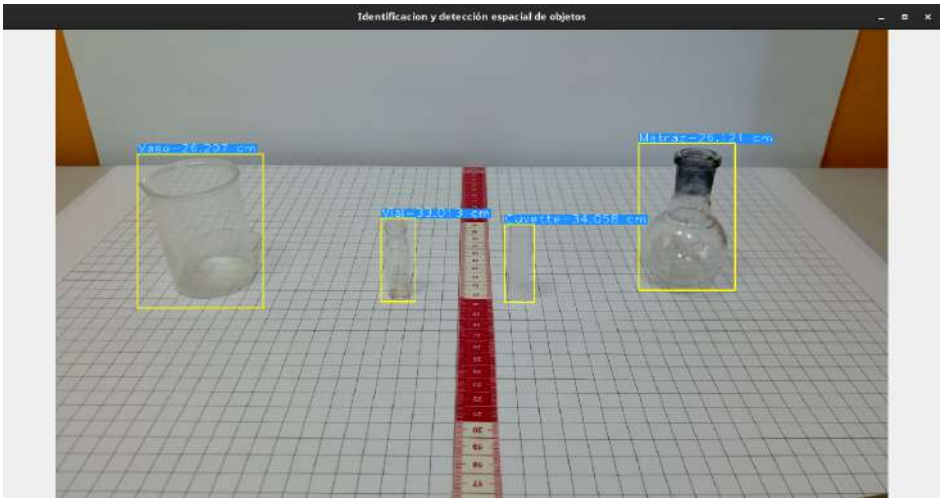
Objeto	**Distancia de referencia**	**Ancho**	**Ancho píxeles**	**Distancia Focal**
Cuvette	25cm	1.5cm	23px	1566.66
Matraz	25cm	5cm	64px	1165.0
Vaso	25cm	5.8cm	87px	1310.34
Vial	25cm	1.5cm	26 px	1716.66

Tabla 4.2: Nuevo cálculo de distancia focal.

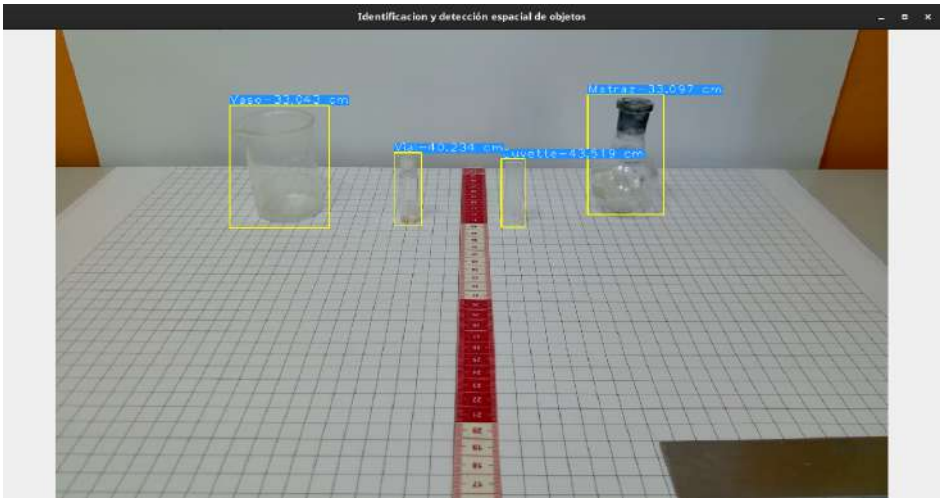
Posteriormente se realizan estimaciones de distancia a a intervalos de 10cm (20, 30 y 40cm), cuyos resultados, mostrados en la figura 4.12, indican que hay oportunidades para refinar el modelo y obtener resultados más precisos.



(a)



(b)



(c)

Figura 4.12: Ejemplos de estimación de distancia en intervalos de distancia. 20cm (a), 30cm (b) y 40cm (c).

4.5. Calibración de la cámara.

Para llevar a cabo la calibración, se utilizó un patrón de calibración estándar como es el tablero de ajedrez de 9x6 (figura 4.13) y se capturaron múltiples imágenes desde diferentes perspectivas usando le método que se muestra en el código 2.

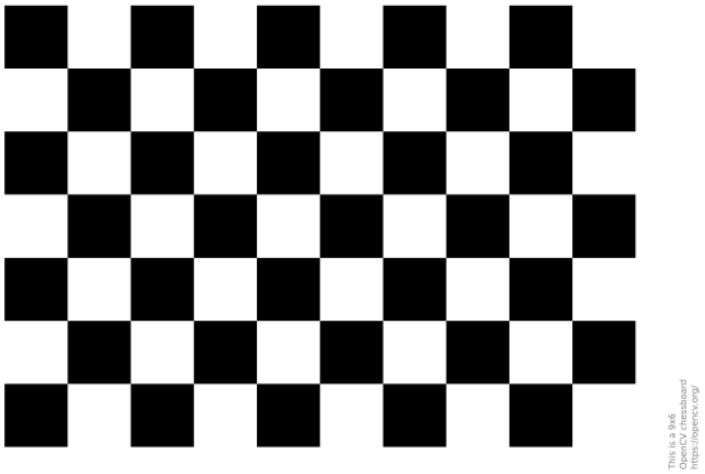


Figura 4.13: Tablero de ajedrez usado para la calibración de la cámara.

```
def getImages(self):
    cap = cv.VideoCapture(self.camIndex)
    num = 0
    while cap.isOpened():
        sucess, img = cap.read()
        k = cv.waitKey(5)
        if k == 27:
            break
        elif k == ord('c') or k == ord('C'):
            name =
             ↪  os.path.join(self.constants['calibrationDir'],'img'+str(num)+'.png')
            cv.imwrite(name, img)
            print(f'Imagen {num} guradada.')
            num += 1
        cv.imshow('Imagen capturada', img)
    cap.release()
    cv.destroyAllWindows()
```

Código 2: Método para la captura de imágenes que contenga el patrón de calibración.

Posteriormente, se implementó un algoritmo en Python (código 3) utilizando la biblioteca OpenCV para detectar las esquinas del patrón en cada imagen. Los puntos 3D conocidos del patrón y los puntos 2D detectados en las imágenes se utilizaron como entrada para la función

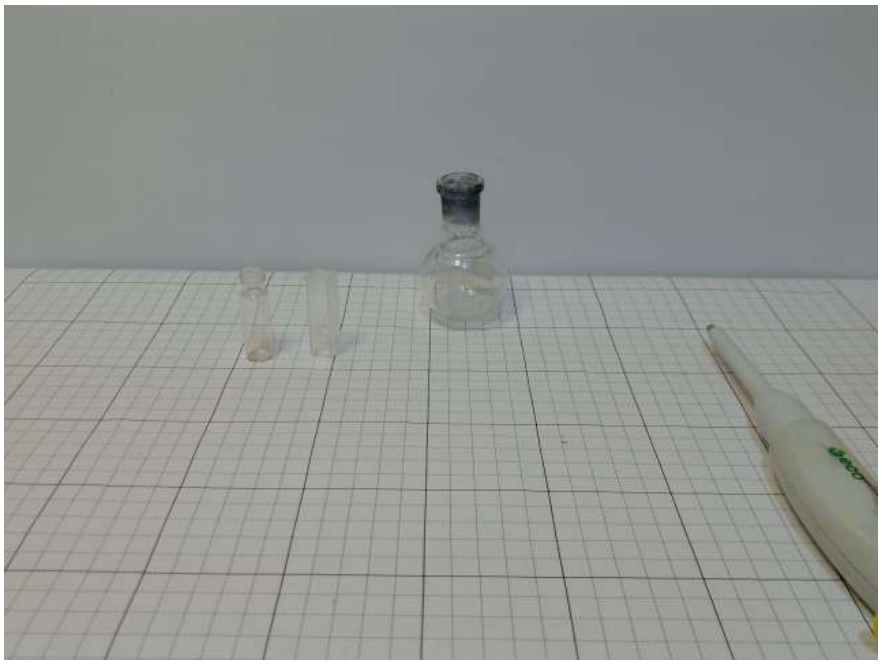
cv.calibrateCamera, la cual calculó los parámetros intrínsecos y extrínsecos de la cámara. Los parámetros intrínsecos describen las características internas de la cámara, como la longitud focal y el centro óptico, mientras que los parámetros extrínsecos relacionan el sistema de coordenadas de la cámara con el sistema de coordenadas del mundo real. Una vez obtenidos los parámetros de calibración, se guardan en un archivo para usarlos en la estimación de distancia. Los resultados muestran una mejora significativa en la calidad de la imagen, eliminando las distorsiones típicas de las lentes.

```
def calibrate(self):
    chessboardSize = (9,6)
    frameSize = (1920,1080)

    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER,30,0.001)
    objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
    objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)

    size_of_chessboard_squares_mm = 25
    objp = objp * size_of_chessboard_squares_mm

    objpoints = [] # 3d point
    imgpoints = [] # 2d points

    images = glob.glob(os.path.join(self.constants['calibrationDir'], '*.png'))
    for image in images:
        img = cv.imread(image)
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)
        if ret == True:
            objpoints.append(objp)
            corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
            imgpoints.append(corners)
            cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
            cv.imshow('img', img)
            cv.waitKey(100)
    cv.destroyAllWindows()

    #CALIBRATION
    ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,
    ↪  frameSize, None, None)
    pickle.dump((cameraMatrix, dist),
    ↪  open(os.path.join(self.constants['calibrationDirSaves'], 'calibration.pkl'),
    ↪  "wb" ), protocol = 2)
    pickle.dump(cameraMatrix, open(
    ↪  os.path.join(self.constants['calibrationDirSaves'], 'cameraMatrix.pkl'), "wb"
    ↪  ), protocol = 2)
    pickle.dump(dist, open(os.path.join(self.constants['calibrationDirSaves'],
    ↪  'dist.pkl'), "wb" ), protocol = 2)
```

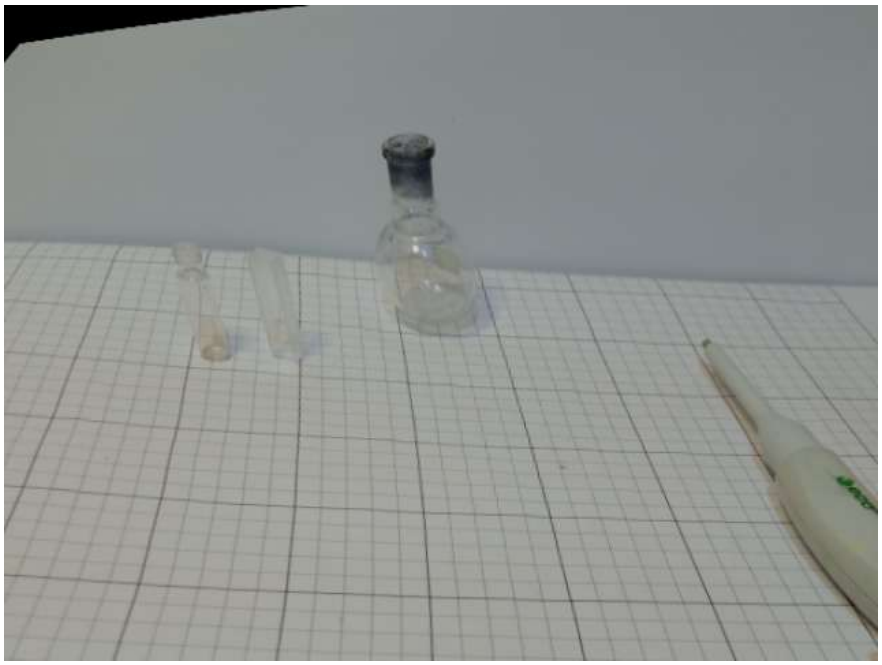
Código 3: Método para la calibración de la cámara usando la librería *openCV*

Una vez obtenidos los parámetros de calibración, se procedió a evaluar su efectividad en imágenes del conjunto de entrenamiento. se aplicó la función *cv.undistort* para corregir la distorsión

utilizando los parámetros calculados. En la figura 4.14 de muestra tanto la imagen original como la imagen corregida.



(a) Imagen original



(b) Imagen corregida con la calibración de la cámara

Figura 4.14

Capítulo 5

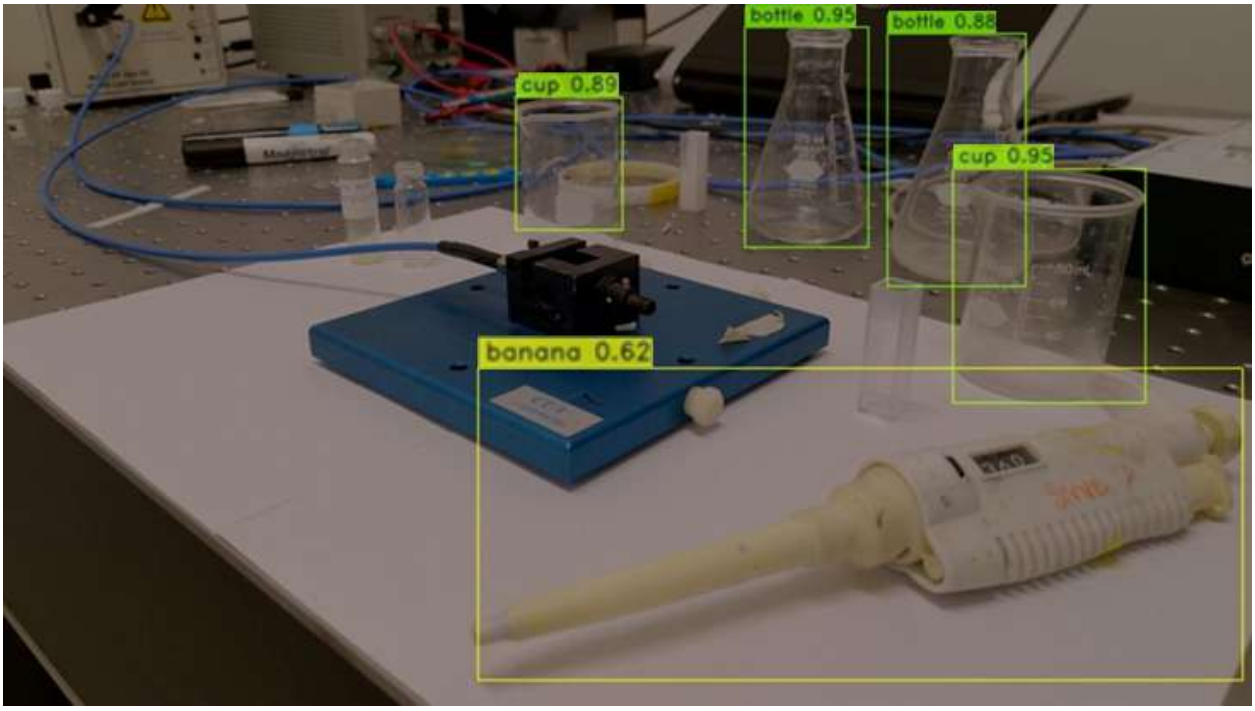
Resultados

5.1. Identificación de objetos

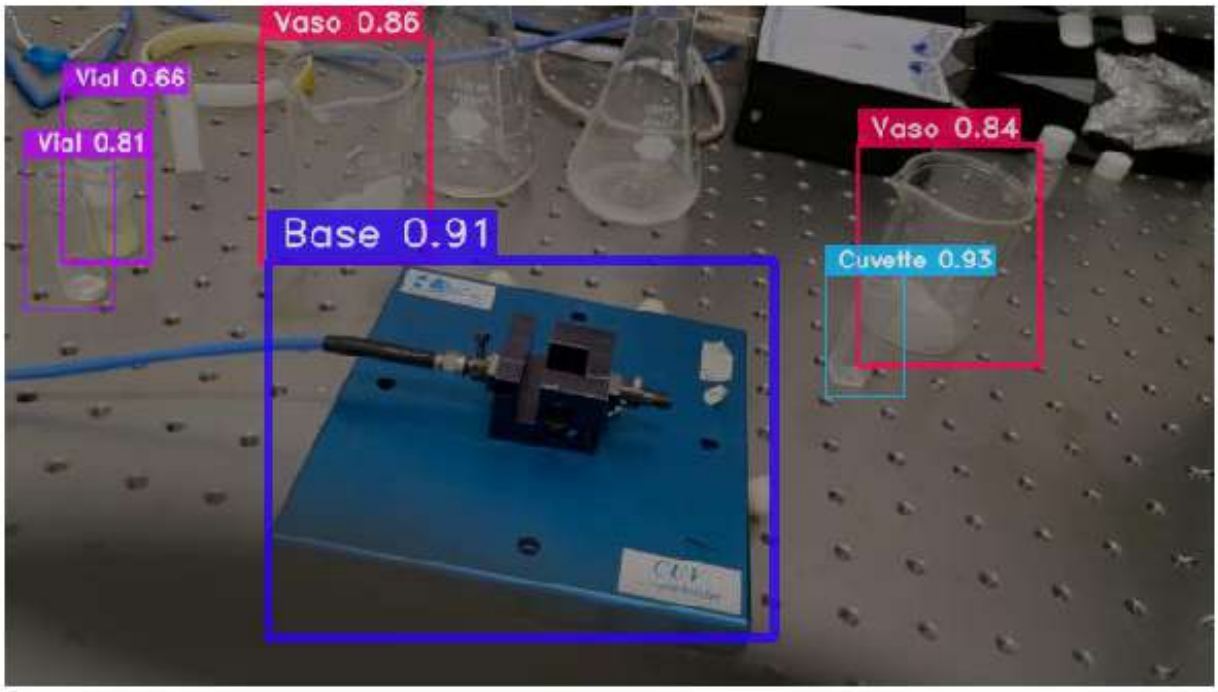
Al comparar el modelo sin reentrenar con nuestro modelo personalizado, se observó una diferencia significativa en los resultados. Como se muestra en la figura 5.1, el modelo original presenta predicciones incorrectas, incluso con altos niveles de confianza, mientras que nuestro modelo personalizado identifica correctamente los objetos seleccionados. El modelo fue evaluado con la métrica de *m*AP@0.50 y *m*AP@0.75 obteniendo los resultados mostrados en la tabla 5.1.

Instrumento	AP@0.50	AP@0.75
Base	0.96	0.94
Cuvette	0.95	0.86
Matraz	0.96	0.91
Matraz3B	0.98	0.93
Pipeta	0.94	0.67
Vaso	0.98	0.92
Vial	0.88	0.79
	***m*AP** 0.95	***m*AP** 0.86

Tabla 5.1: En la tabla se muestra la precisión media de cada objeto para los dos umbrales (0.50 y 0.75) al final de la columna se muestra la *m*AP para cada umbral.



(a)



(b)

Figura 5.1: Predicciónes usando el modelo original (a) y usando el modelo reentrenado (b).

La matriz de confusión (Figura 5.2) confirma esta superioridad, mostrando valores superiores a 0.75 en la diagonal principal y bajos índices de falsos positivos y falsos negativos.

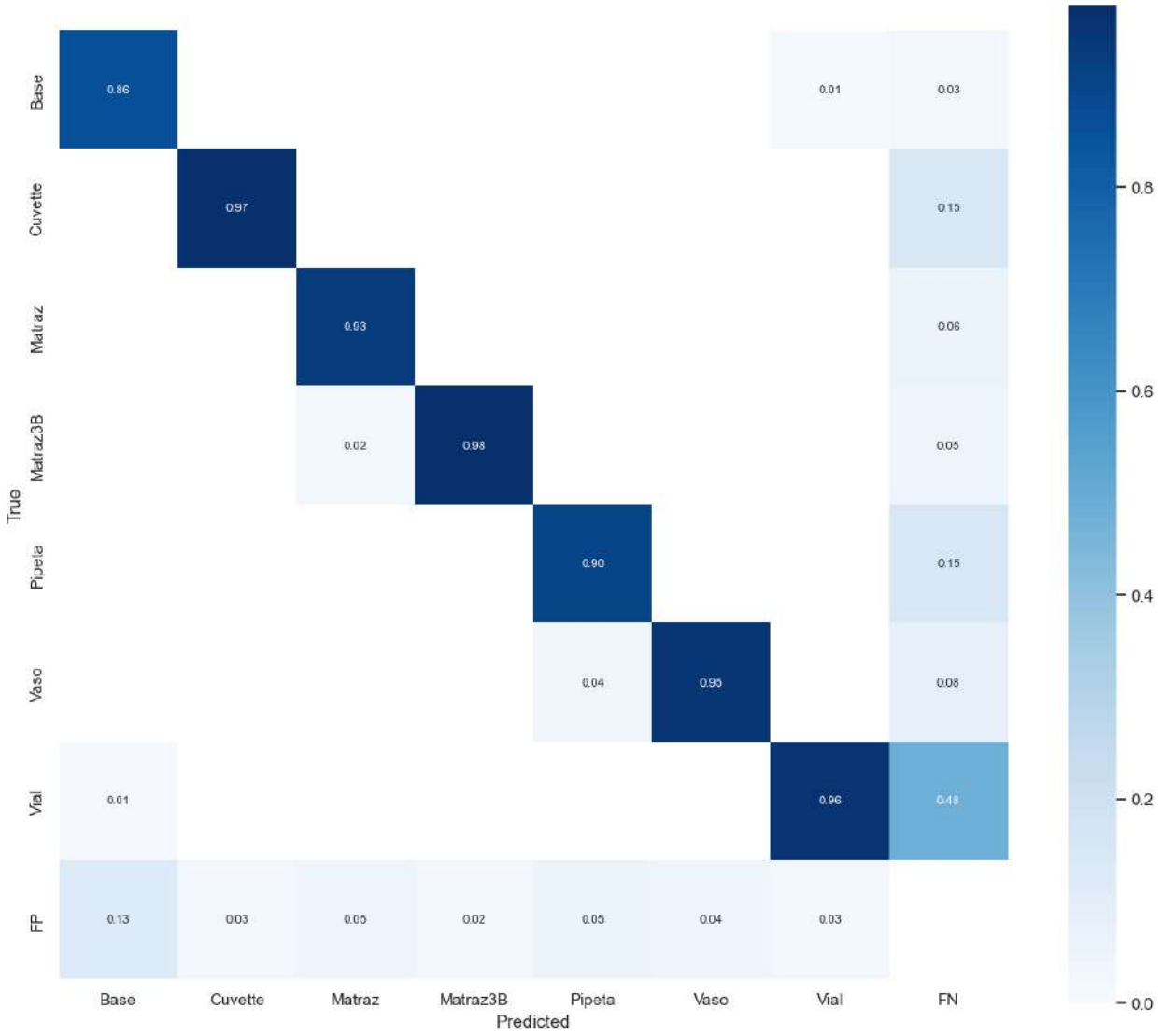


Figura 5.2: Matriz de confusión obtenida de la evaluación del modelo generado con el conjunto de datos personalizado.

5.2. Estimación de la distancia

Se realizaron pruebas de estimación de distancia de un vaso de precipitado, cuvette, matraz, vial, se colocaron en distancias de 20,25,30,35,40 cm y se realizo una comparación con la distancia real para calcular el porcentaje de error con la formula.

$$e_r(\%) = \frac{|D_{est} - D_{real}|}{D_{real}} \times 100$$

Estimación de la distancia de Cuvette			
Distancia real	Distancia estimada	Diferencia	Porcentaje de error
20 cm	20.294 cm	0.294 cm	1.47 %
25 cm	23.793 cm	-1.207 cm	4.82 %
30 cm	28.750 cm	-1.250 cm	4.16 %
35 cm	28.750 cm	-6.500 cm	18.57 %
40 cm	32.857 cm	-7.143 cm	17.85 %

Tabla 5.2: Cálculo de error en la estimación de distancia de cuvette.

Estimación de la distancia de Matraz			
Distancia real	Distancia estimada	Diferencia	Porcentaje de error
20cm	21.613 cm	1.613 cm	8.06 %
25cm	25.443 cm	0.443 cm	1.77 %
30cm	28.714 cm	-1.286 cm	4.28 %
35cm	29.130 cm	-5.870 cm	16.77 %
40cm	34.068 cm	-5.932 cm	14.83 %

Tabla 5.3: Cálculo de error en la estimación de distancia de matraz.

Estimación de la distancia de Vaso de Precipitado			
Distancia real	Distancia estimada	Diferencia	Porcentaje de error
20cm	21.750 cm	1.750 cm	8.75 %
25cm	24.623 cm	-0.377 cm	1.50 %
30cm	32.222 cm	2.222 cm	7.40 %
35cm	32.625 cm	-2.375 cm	6.78 %
40cm	35.753 cm	-4.247 cm	10.61 %

Tabla 5.4: Cálculo de error en la estimación de distancia de vaso de precipitado.

Estimación de la distancia de Vial			
Distancia real	Distancia estimada	Diferencia	Porcentaje de error
20cm	22.941 cm	2.941 cm	14.70 %
25cm	24.375 cm	-0.625 cm	2.50 %
30cm	28.899 cm	-1.101 cm	3.67 %
35cm	33.913 cm	-1.087 cm	3.10 %
40cm	35.455 cm	-4.545 cm	11.36 %

Tabla 5.5: Cálculo de error en la estimación de distancia de vial.

Los resultados presentados en las tablas muestran una clara tendencia: la precisión de las estimaciones disminuye a medida que aumenta la distancia entre la cámara y el objeto. Esto se debe a la pérdida de detalle del objeto a mayores distancias. Se observó un porcentaje de error máximo del 17.85 % y mínimo del 1.47 % para el cuvette. Sorprendentemente, la segunda estimación no mostró una mejora, sino que, en algunos casos, el porcentaje de error incluso aumentó.

Estimación de la distancia de Cuvette			
Distancia real	Distancia estimada	Diferencia	Porcentaje de error
20 cm	30.128 cm	10.128 cm	50.64 %
30 cm	34.058 cm	4.058 cm	13.52 %
40 cm	43.519 cm	3.519 cm	8.79 %

Tabla 5.6: Cálculo de error en la estimación de distancia de cuvette.

Estimación de la distancia de Matraz			
Distancia real	Distancia estimada	Diferencia	Porcentaje de error
20cm	20.656 cm	0.656 cm	3.28 %
30cm	26.121 cm	-3.879 cm	12.93 %
40cm	33.097 cm	-6.903 cm	17.25 %

Tabla 5.7: Cálculo de error en la estimación de distancia de matraz.

Estimación de la distancia de Vaso de Precipitado			
Distancia real	Distancia estimada	Diferencia	Porcentaje de error
20cm	18.895 cm	-1.105 cm	5.525 %
30cm	26.207 cm	-3.793 cm	12.64 %
40cm	33.043 cm	-6.957 cm	17.39 %

Tabla 5.8: Cálculo de error en la estimación de distancia de vaso de precipitado.

Estimación de la distancia de Vial			
Distancia real	Distancia estimada	Diferencia	Porcentaje de error
20cm	25.0 cm	5 cm	25 %
30cm	32.188 cm	2.188 cm	7.29 %
40cm	40.234 cm	0.234 cm	0.58 %

Tabla 5.9: Cálculo de error en la estimación de distancia de vial.

Capítulo 6

Conclusiones

En este trabajo se desarrolló un sistema de visión por computadora basado en una Red Neuronal Convolucional (CNN) para identificar instrumentos químicos en un laboratorio de síntesis de materiales. Inicialmente, se generó un conjunto de datos de 809 imágenes pero con la implementación de técnicas de aumento de datos se lograron obtener 6777 imágenes. Esta expansión permitió reentrenar de manera adecuada a una CNN y obteniendo una *mean Average Precision* *m*AP@0.50 de 0.95 y *m*AP@0.75 de 0.86. Estos valores indican una alta precisión en la identificación de los instrumentos.La importancia de contar con conjuntos de datos personalizados radica en la capacidad de las CNN de aprender características específicas de los objetos de interés, evitando confusiones comunes como por ejemplo identificar como "frasco" a un vaso de precipitado y a una matraz usando conjuntos de imágenes genéricos.

La estimación de la distancia mediante triangulación, aunque es un método ampliamente utilizado, presenta ciertas limitaciones que pueden afectar su precisión. La correspondencia de puntos entre las imágenes, la calibración de las cámaras y las distorsiones ópticas son factores que pueden introducir errores en la medición. Además, la precisión de la estimación disminuye a medida que aumenta la distancia al objeto o cuando las características del objeto son poco distintivas.

Como siguiente paso, se propone evaluar el desempeño del sistema en entornos reales, donde las condiciones son más variables y complejas. Esto permitirá validar su robustez y precisión en escenarios del mundo real. Además, se explorarán técnicas de aprendizaje profundo para refinar las estimaciones de profundidad y se considerará la integración de sensores LiDAR para obtener medidas de distancia más precisas y robustas, ampliando así las capacidades del sistema.

Bibliografía

[1] R. Borja-Robalino, A. Monleon-Getino, and J. Rodellar. Estandarización de métricas de rendimiento para clasificadores machine y deep learning. *Revista Ibérica de Sistemas e Tecnologias de Informação*, (E30):184–196, 2020.

[2] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2):125, Feb. 2020.

[3] F. Ciaglia, F. S. Zuppichini, P. Guerrie, M. McQuade, and J. Solawetz. Roboflow 100: A rich, multi-domain object detection benchmark, 2022.

[4] A. Crespo Rodríguez. Sistema de detección y estimación de pose de objetos basado en visión por computador para planta piloto industria 4.0. Master's thesis, Universidad de Oviedo, July 2019.

[5] Deci. YOLO-NAS by DECI achieves SOTA performance on object detection using neural architecture search, 1 2024.

[6] Deci-Ai. super-gradients/src/super_gradients/recipes/training_hyperparams/default_train_params.yaml at master · Deci-AI/super-gradients.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li. Imagenet: a large-scale hierarchical image database. pages 248–255, 06 2009.

[8] Z. Ding. Chemistry laboratory apparatus dataset, 2023.

[9] H. V. Dung and K.-H. Jo. A simplified solution to motion estimation using an omnidirectional camera and a 2-d lrf sensor. *IEEE Transactions on Industrial Informatics*, 12:1–1, 06 2016.

[10] A. Dutta and A. Zisserman. Vgg image annotator (via). http://www.robots.ox.ac.uk/~vgg/software/via/, 2019.

[11] S. Eppel, H. Xu, M. Bismuth, and A. Aspuru-Guzik. Vector-LabPics dataset for images of materials in vessels in the chemistry lab, 2020.

[12] E. Frank. Lidar sensors: Technology and applications author. 05 2024.

[13] J. Gené-Mola, V. Vilaplana, J. R. Rosell-Polo, J.-R. Morros, J. Ruiz-Hidalgo, and E. Gregorio. Uso de redes neuronales convolucionales para la detección remota de frutos con cámaras RGB-D. 2019.

[14] A. Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Incorporated, 2019.

[15] A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti, and D. De. *Fundamental Concepts of Convolutional Neural Network*, pages 519–567. 01 2020.

[16] M. Guertler, L. Tomidei, N. Sick, M. Carmichael, G. Paul, A. Wambsganss, V. Hernandez Moreno, and S. Hussain. When is a robot a cobot? moving beyond manufacturing and arm-based cobot manipulators. *Proc. Des. Soc.*, 3:3889–3898, July 2023.

[17] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn, 2018.

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[19] Intel. Cvat: Computer vision annotation tool. https://github.com/openvinotoolkit/cvat, 2020.

[20] C. Khosla and B. S. Saini. Enhancing performance of deep learning models with different data augmentation techniques: A survey. In *2020 International Conference on Intelligent Engineering and Management (ICIEM)*, pages 79–85, 2020.

[21] D. Kirschner, R. Velik, S. Yahyanejad, M. Brandstötter, and M. Hofbaur. YuMi, come and play with me! a collaborative robot for piecing together a tangram puzzle. In *Lecture Notes in Computer Science*, Lecture notes in computer science, pages 243–251. Springer International Publishing, Cham, 2016.

[22] Y. Li, X. Zhang, and Z. Shen. Yolo-submarine cable: An improved yolo-v3 network for object detection on submarine cable images. *Journal of Marine Science and Engineering*, 10(8), 2022.

[23] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection, 2018.

[24] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2015.

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. *SSD: Single Shot MultiBox Detector*, page 21–37. Springer International Publishing, 2016.

[26] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[27] G. Mandal, D. Bhattacharya, and De. *Real-Time Vision-Based Vehicle-to-Vehicle Distance Estimation on Two-Lane Single Carriageway Using a Low-Cost 2D Camera at Night*, pages 54–65. 12 2020.

[28] R. Michalík, A. Janota, M. Gregor, and M. Hruboš. Human-robot motion control application with artificial intelligence for a cooperating yumi robot. *Electronics*, 10(16), 2021.

[29] D. J. Montgomery, C. K. Jones, J. N. Stewart, and A. Smith. Stereoscopic camera design. In M. T. Bolas, A. J. Woods, J. O. Merritt, S. A. Benton, and M. T. Bolas, editors, *Stereoscopic Displays and Virtual Reality Systems IX*, volume 4660, pages 26 – 37. International Society for Optics and Photonics, SPIE, 2002.

[30] A. Mulyanto, R. Indra, P. Prasetyawan, and A. Sumarudin. Implementation 2d lidar and camera for detection object and distance based on ros. *JOIV : International Journal on Informatics Visualization*, 4, 12 2020.

[31] OpenCV. Opencv: Camera calibration.

[32] S. K. Pathi, A. Kiselev, A. Kristoffersson, D. Repsilber, and A. Loutfi. A novel method for estimating distances from a robot to humans using egocentric rgb camera. *Sensors*, 19(14), 2019.

[33] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016.

[34] J. Redmon and A. Farhadi. Yolov3: An incremental improvement, 2018.

[35] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[36] Sagi Eppel, Haoping Xu, Alan Aspuru-Guzik, Mor Bismuth. LabPics dataset for visual understanding of medical and chemistry labs, 2021.

[37] M. Samiei and R. Li. Object detection with deep reinforcement learning, 2022.

[38] S. Shao, Z. Li, T. Zhang, C. Peng, G. Yu, X. Zhang, J. Li, and J. Sun. Objects365: A large-scale, high-quality dataset for object detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8429–8438, 2019.

[39] P. Skalski. Makesense: Free online image annotation tool. https://www.makesense.ai/, 2019.

[40] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, Cham, 2022.

[41] M. Tan, R. Pang, and Q. V. Le. Efficientdet: Scalable and efficient object detection, 2020.

[42] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716, Nov. 2023.

[43] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov. Label Studio: Data labeling software, 2020-2022. Open source software available from https://github.com/heartexlabs/label-studio.

[44] TOOLIFY. Importancia de la división en conjuntos de entrenamiento, prueba y validación en aprendizaje automático, 2024. Recuperado de TOOLIFY, la mejor fuente de herramientas de IA.

[45] Tzutalin. Labelimg. https://github.com/tzutalin/labelImg, 2015.

[46] Ultralytics. Train custom data, 6 2024.

[47] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee. A survey of modern deep learning based object detection models. *Digital Signal Processing*, 126:103514, 2022.

[48] H. Zhao, Z. Tang, Z. Li, Y. Dong, Y. Si, M. Lu, and G. Panoutsos. Real-time object detection and robotic manipulation for agriculture using a yolo-based learning approach, 2024.