UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DETECTOR INTEGRADO DE INSTANCIAS DE OBJETOS EN TIEMPO REAL

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

JUAN PABLO CÁCERES BRIONES

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTIN

PROFESOR CO-GUÍA:
PATRICIO LONCOMILLA ZAMBRANA

MIEMBROS DE LA COMISIÓN:
CLAUDIO PÉREZ FLORES
IVÁN SIPIRÁN MENDOZA

Esta tesis ha sido parcialmente financiado por el proyecto FONDEQUIP EQM170041 y por el proyecto BASAL AFB230001

SANTIAGO DE CHILE
2024

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: JUAN PABLO CÁCERES BRIONES
FECHA: 2024
PROFESOR GUÍA: JAVIER RUIZ DEL SOLAR SAN MARTÍN

DETECTOR INTEGRADO DE INSTANCIAS DE OBJETOS EN TIEMPO REAL

Esta investigación aborda la detección y reconocimiento de instancias de objetos en tiempo real. Se centra en integrar un detector de objetos y un módulo de extracción de descriptores en una única red, utilizando YOLOv7 modificado y una CNN especializada, JPNet. El objetivo es desarrollar un sistema capaz de detectar objetos genéricos y extraer descriptores globales de manera eficiente. Para esto, se modificó YOLOv7, adaptándolo a la detección de objetos genéricos y se entrenó con COCO etiquetado con la red SAM. Este enfoque demostró superar la precisión de YOLOv7. Para el desarrollo del módulo extractor de descriptores, se diseñó JPNet, una arquitectura con operaciones RoI, GeM y Neck. Se investigó su integración con diferentes stages de YOLOv7 y se descubrió que conectado al stage 1 se obtienen los mejores descriptores. El sistema completo, YOLOv7 modificado con JPNet, se validó en términos de rendimiento y velocidad, demostrando ser eficaz en tiempo real con un F1-score promedio de 0.742 a 32 FPS. Se realizó una comparativa con YOLOSPoC, reconocedor de instancias de objetos en base a YOLO y descriptores SPoC, encontrando que aunque el sistema propuesto lo supera en ciertos escenarios, en el promedio de escenarios YOLOSPoC muestra un mejor rendimiento.

A la persona que estuvo para mí cuando más lo necesite.

Agradecimientos

En primer lugar, quiero expresar mi más sincero agradecimiento a mi profesor guía, Javier Ruiz del Solar, y a mi co-guía, Patricio Loncomilla, por darme la oportunidad de trabajar en este interesante tema de tesis, por su constante motivación y orientación durante su desarrollo. Agradezco especialmente al profesor Javier por aceptarme como su alumno, por la confianza depositada en mí, así como por los valiosos consejos y enseñanzas que he recibido a lo largo de los cursos y proyectos realizados en el laboratorio de robótica. Todo ello ha sido fundamental para mi crecimiento como estudiante y profesional.

También quiero agradecer profundamente a mi familia, en especial a mis abuelos, padres, hermanos y mi tía, por su incondicional apoyo, por brindarme todas las facilidades y recursos necesarios para ingresar y mantenerme en la universidad. No puedo dejar de mencionar a mi querido gato Pepi, quien ha sido un pilar emocional clave en la recta final de mi vida universitaria.

Mis agradecimientos van igualmente a todos los compañeros que tuve en el laboratorio de robótica, por las experiencias compartidas y todo lo que he aprendido de cada uno de ellos. Agradezco asimismo a mis colegas de trabajo en Dual Vision, por los conocimientos y los buenos momentos vividos juntos.

Un agradecimiento especial a mi amigo y socio Giovanni Pais, por el apoyo mutuo que nos brindamos en el cierre de nuestras carreras, por los consejos compartidos, y por esas largas noches y fines de semana dedicados a trabajar en nuestras respectivas tesis.

Finalmente, agradezco a todos mis amigos con quienes he compartido a lo largo de mi vida universitaria, por hacer de esta etapa una experiencia más enriquecedora y amena. En particular, quiero expresar mi gratitud a mi buen amigo Ignacio Araya, quien ha sido un apoyo fundamental en este camino.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Lista de acrónimos

- **ADAM**: Adaptive Moment Estimation. Estimación Adaptativa de Momentos.
- **ANN**: Artificial Neural Network. Red Neuronal Artificial.
- **CE**: Cross Entropy. Entropía Cruzada.
- **CNN**: Convolutional Neural Network. Red Neuronal Convolucional.
- **DL**: Deep Learning. Aprendizaje Profundo.
- **DRL**: Deep Reinforcement Learning. Aprendizaje Reforzado Profundo.
- **FN**: False Negative. Falso Negativo.
- **FP**: False Positive. Falso Positivo.
- **FPS**: Frames per Second. Fotogramas por Segundo.
- **GAN**: Generative Adversarial Networks. Redes Generativas Adversarias.
- **GeM**: Generalized Mean Pooling. Agrupamiento de Media Generalizada.
- **GPU**: Graphics Processing Unit. Unidad de Procesamiento Gráfico.
- **IA**: Artificial Intelligence. Inteligencia Artificial.
- **IoU**: Intersection over Union. Intersección sobre Unión.
- **mAP**: Mean Average Precision. Precisión Media Promedio.
- **ML**: Machine Learning. Aprendizaje de Máquinas.
- **RPN**: Region Proposal Network. Red de Propuesta de Regiones.
- **RoI**: Region of Interest. Región de Interés.
- **RL**: Reinforcement Learning. Aprendizaje Reforzado.
- **RPC**: Retail Product Checkout. Pago de Productos en Retail.
- **SAM**: Segment Anything Model. Modelo para Segmentar Cualquier Cosa.
- **SGD**: Stochastic Gradient Descent. Descenso del Gradiente Estocástico.
- **TN**: True Negative. Verdadero Negativo.
- **TP**: True Positive. Verdadero Positivo.

- **YOLO**: You Only Look Once. Sólo Miras Una Vez.
- **YOLOSPoC**: You Only Look Once & Sum-Pooled Convolutional Features. Sólo Miras Una Vez y Convoluciones Agrupadas por Suma.

Capítulo 1

Introducción

1.1. Motivación

En las últimas décadas, la inteligencia artificial y la visión computacional han experimentado un crecimiento exponencial, impulsando avances significativos en múltiples campos de investigación. Esta evolución ha sido posibilitada en gran medida por el aprendizaje profundo y el acceso a enormes conjuntos de datos, permitiendo el entrenamiento de modelos con millones de parámetros, superando así a sistemas basados en características diseñadas a mano.

En este contexto, la detección de objetos se ha destacado como una área fundamental. Esta disciplina busca capacitar a las máquinas para identificar y localizar objetos en imágenes o secuencias de vídeos. Sin embargo, gran parte de la investigación y desarrollo en este campo se ha enfocado en detectores que reconocen un conjunto limitado de clases de objetos, condicionado principalmente por su conjunto de datos de entrenamiento. Por ejemplo, detectores entrenados con el conjunto de datos Microsoft COCO [29] se limitan a reconocer tan solo 80 clases de objetos, mientras que otros entrenados con Pascal VOC 2012 [27] trabajan con un conjunto aún más reducido de solo 20 clases de objetos.

Esta limitación, aunque sirva para muchas aplicaciones generales, se vuelve problemática cuando se trata de aplicaciones más específicas y centradas en la vida cotidiana. En escenarios habituales, como la interacción de robots en entornos domésticos o la revisión de góndolas de tiendas y supermercados, se requiere la identificación y localización de objetos que no necesariamente pertenecen a las clases estándar predefinidas en estos conjuntos de datos.

La solución convencional para este problema implica el etiquetado manual de miles de imágenes de los objetos de interés y un proceso de entrenamiento desde cero. Sin embargo, esta tarea es prácticamente inviable para la mayoría de las personas debido a lo complejo y costoso en tiempo y recursos. Además, para obtener resultados satisfactorios, es necesario implementar técnicas especializadas, como el uso de estrategias de *bag of freebies* y *bag of special* que se introducen en YOLOv4 [1], que requieren un conocimiento experto en el campo de la visión computacional.

Ante este desafío, surge la necesidad de desarrollar detectores capaces de localizar e identificar instancias de objetos. Estos detectores permitirían encontrar objetos específicos, como una taza de café, una billetera o un teléfono celular, sin la necesidad de capturar y etiquetar miles de imágenes de estos. En su lugar, se podrían utilizar una o varias imágenes de referencia de estos objetos, cuya cantidad dependería de la simetría y variabilidad de los mismos. Este enfoque innovador tiene el potencial de simplificar enormemente la detección de objetos en contextos de uso cotidiano, eliminando la barrera del etiquetado manual intensivo y haciéndola más accesible para una amplia gama de aplicaciones y usuarios.

Un ejemplo destacado de los resultados obtenidos por este tipo de detectores se encuentra en el método denominado *YOLOSPoC* [33]. Este método emplea dos redes convolucionales profundas de manera independiente, una destinada a generar propuestas de detección de objetos y la otra para calcular descriptores globales de cada objeto detectado. Posteriormente, se lleva a cabo un proceso de reconocimiento de instancias mediante la comparación de los descriptores obtenidos, seguido por un proceso de eliminación de detecciones redundantes.

YOLOSPoC, a pesar de ser un sistema completo para la detección de instancias de objetos, enfoca su estudio primordialmente en la extracción de descriptores y el reconocimiento de instancias. Esto se refleja en sus resultados en el conjunto de datos *DSLL* [34], donde al utilizar como generador de propuestas la red de detección de clases de objetos *YOLOv3* [11] entrenada con *COCO*, se alcanza un *F1-score* de 0,843. En contraste, cuando el extractor de descriptores utiliza las ubicaciones verdaderas de los objetos, el sistema logra un impresionante *F1-score* de 0,941. Esta diferencia puede explicarse debido a las dificultades del generador de propuestas cuando se encuentra con objetos que no pertenecen o son muy diferentes a las 80 clases de *COCO*. Otra limitación de este modelo es la utilización de dos redes convolucionales independientes para tareas complementarias, lo cual podría conllevar un gasto computacional innecesario y la pérdida de información valiosa que podría compartir la red que detecta los objetos con la red que extrae descriptores.

El trabajo anterior proporciona la motivación principal que impulsó el desarrollo de esta tesis. Esta investigación se enfoca en la creación de un detector integrado de instancias de objetos particulares, donde la red generadora de propuestas está integrada con la red extractora de descriptores. De esta forma, ambas redes pueden compartir características esenciales, resultando en un sistema más eficiente en cuanto a costos. Además, es fundamental que el generador de propuestas se base en un detector de objetos genérico, capaz de identificar cualquier tipo de objeto, independientemente de su clase.

1.2. Hipótesis

La hipótesis de esta tesis es que es posible desarrollar un detector integrado de instancias de objetos basado en técnicas de visión computacional, aprendizaje profundo y aprendizaje métrico, que permita la localización e identificación precisa de instancias de objetos en tiempo real, sin la necesidad de un extenso etiquetado manual de datos de entrenamiento de éstos.

Se desea que este enfoque, sea efectivo y práctico en una variedad de aplicaciones cotidianas y especializadas. Además, se espera que este detector contribuya significativamente a

resolver problemas en dominios como la robótica y la interacción hombre-máquina, mejorando la comodidad y la eficiencia del uso del detector en entornos reales.

1.3. Objetivos generales

El objetivo de esta tesis consiste en diseñar e implementar un detector integrado de instancias de objetos basado en técnicas de visión computacional, aprendizaje profundo y aprendizaje métrico, con un enfoque en la localización e identificación precisa de instancias de objetos en tiempo real. Esta investigación busca superar las limitaciones de los detectores tradicionales, al permitir la detección de objetos no pertenecientes a clases predefinidas, utilizando únicamente un número limitado de imágenes de referencia.

1.4. Objetivos específicos

Con el propósito de cumplir el objetivo general y verificar la hipótesis de este trabajo, se definen los siguientes objetivos específicos:

- El diseño, implementación y entrenamiento de un detector de objetos genéricos que detecte cualquier tipo de objeto en tiempo real.
- Evaluación y elección del mejor detector de objetos genéricos diseñado y entrenado.
- El diseño, implementación y entrenamiento de una red extractora de descriptores unida al detector genérico seleccionado, que permita el reconocimiento mediante la generación de descriptores representativos.
- Evaluación y elección de la red convolucional y configuración que mejor se adapte al módulo extractor de descriptores.
- Evaluación y análisis del sistema completo, incorporando el detector de objetos genéricos seleccionado con el mejor módulo de extracción de descriptores.

1.5. Alcances de este trabajo

El sistema se desarrollará y validará a través del entrenamiento con conjuntos de datos tradicionales. Como se estableció en los objetivos específicos, se espera que el detector localice una amplia variedad de objetos genéricos sin la necesidad de futuros entrenamientos. No obstante, es posible que, cuando la red extractora de características trate de generar descriptores de objetos que varíen considerablemente en forma y contexto, con respecto a los conjuntos de datos utilizados para el entrenamiento del extractor, el rendimiento del sistema de reconocimiento podría verse afectado.

Dado esto, puede ser esencial un entrenamiento adicional del módulo de extracción de descriptores usando un conjunto de clasificación que contenga formas y contextos similares de los objetos deseados. Por ejemplo, si se entrena el módulo con datos de objetos del hogar, su eficiencia para identificar vehículos o personas podría no ser óptima. En tales casos, sería recomendable ajustar su entrenamiento con un conjunto de datos más acorde al contexto. Cabe señalar que este entrenamiento adicional se limitaría únicamente al módulo extractor de descriptores y no al sistema completo, lo cual disminuye considerablemente el costo y tiempo de entrenamiento.

1.6. Estructura del documento

Este documento está organizado de la siguiente manera:

- En el Capítulo 2, se revisará los antecedentes generales de la detección de objetos particulares, entregando en primera instancia un marco general de que es la visión computacional y el aprendizaje profundo supervisado. Posteriormente se dará una revisión sobre la detección de objetos y aprendizaje métrico supervisado.
- En el Capítulo 3, se detallará el enfoque adoptado para el detector integrado de instancias de objetos, proporcionando una explicación completa de cada uno de los componentes que conforman el sistema.
- En el Capítulo 4, se expondrán los entrenamientos y experimentos para la evaluación del sistema propuesto para cumplir con el objetivo de esta tesis.
- En el Capítulo 5, se realizará un análisis detallado de los resultados obtenidos, profundizando en su relevancia y su impacto en el contexto de la investigación.
- En el Capítulo 6, se expondrán las conclusiones extraídas de esta tesis, resumiendo los descubrimientos más destacados y su relevancia en el ámbito de estudio. Además, se identificarán posibles direcciones y oportunidades para investigaciones futuras relacionadas con este trabajo.

Capítulo 2

Marco teórico

En este capítulo se presentan los conceptos y antecedentes más importantes que sustentan el trabajo realizado. En primera instancia, se dan definiciones generales de qué es la Inteligencia Artificial, la Visión Computacional y el Aprendizaje de Máquinas, dando un breve repaso en los tipos de aprendizajes más comunes. Posteriormente, se profundizará sobre el aprendizaje profundo, explicando qué son las redes neuronales artificiales y las convoluciones y cómo se entrenan. Luego se explicará en qué consisten los detectores de clases de objetos y detectores de instancias de objetos, entregando ejemplos de los principales modelos en los que se basó esta tesis. A continuación, se describirán métodos de aprendizaje métrico supervisado, centrando la explicación principal en los enfoques utilizados en esta tesis. Seguidamente, se presenta el algoritmo innovador empleado para el nuevo etiquetado, que se relaciona con una avanzada red de segmentación de objetos.

2.1. Inteligencia Artificial

La base de todo este trabajo es la Inteligencia Artificial (en inglés *Artificial Intelligence*) o también conocida como IA. Esta es una disciplina de la computación que se enfoca en el desarrollo de programas y sistemas que sean capaces de realizar tareas que los seres humanos hacen y que requieran inteligencia por parte de estos. Estas tareas incluyen el razonamiento lógico, el aprendizaje, la percepción visual, la percepción auditiva, la toma de decisiones y la comprensión del lenguaje natural, entre muchas otras tareas. El objetivo principal es crear algoritmos que puedan asemejar la inteligencia humana, permitiendo a las máquinas la automatización y adaptación de acciones al entorno que las rodea.

2.2. Visión computacional

Una de las disciplinas más destacada de la IA es la Visión Computacional (en inglés *Computer Vision*), también conocida como visión artificial o visión por computadora. Esta se enfoca en el desarrollo de algoritmos y sistemas capaces de interpretar y entender información

visual de imágenes o secuencias de vídeos. Su objetivo principal es permitir a las máquinas ver y entender el mundo al igual que lo hacen los seres humanos, lo que implica que estas puedan tener la capacidad de localizar, identificar y comprender objetos y patrones.

2.3. Aprendizaje de máquinas

La sub-área más relevante de la IA ha sido el Aprendizaje de Máquinas (en inglés *Machine Learning*) [49], esta se enfoca en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender a partir de conjuntos de datos. Este proceso de enseñar a los algoritmos con una gran cantidad de datos se le suele denominar entrenamiento. En el proceso de entrenamiento, los algoritmos ajustan y optimizan sus parámetros para aprender a realizar tareas específicas, tales como, modelado, clasificación y regresión.

Existen tres tipos principales de aprendizaje [49]

- **Aprendizaje Supervisado**: en inglés *Supervised Learning*, es el tipo de aprendizaje más común. En este método, el modelo es entrenado mediante la observación de un conjunto de ejemplos, donde cada entrada se asocia con una salida esperada, conocida como etiqueta (en inglés *label*). A partir de estos datos de entrenamiento, el algoritmo busca encontrar una función que pueda mapear de manera adecuada la entrada con la salida. El objetivo principal es que el algoritmo entrenado pueda hacer predicciones con datos no vistos en el conjunto de entrenamiento.
- **Aprendizaje no Supervisado**: en inglés *Unsupervised Learning*, este tipo de algoritmos aprende patrones en los datos de entrada sin necesidad de retroalimentación explícita o etiquetas supervisadas. Las tareas más comunes en el aprendizaje no supervisado incluyen el agrupamiento (en inglés *clustering*), donde se buscan similitudes en el conjunto de datos para formar grupos o *clusters*; y el modelamiento generativo, como es el caso de las Redes Generativas Adversarias (GANs, por sus siglas en inglés *Generative Adversarial Networks*) [15], que son capaces de generar nuevos datos que son estadísticamente similares a sus propios datos de entrada.
- **Aprendizaje Reforzado**: en inglés *Reinforcement Learning*, es un tipo de algoritmo donde el modelo actúa como un agente (en inglés *agent*) que aprende a tomar decisiones optimizando sus acciones basadas en la retroalimentación recibida. Este agente interactúa con su entorno y recibe recompensas o castigos (en inglés *rewards*) en respuesta a sus acciones. El objetivo del agente es aprender a identificar cuáles acciones maximizan la suma de las recompensas futuras, lo cual se logra mediante la estimación de funciones de valor. Estas funciones evalúan el valor de los estados o acciones tomando en cuenta las recompensas acumuladas a largo plazo.

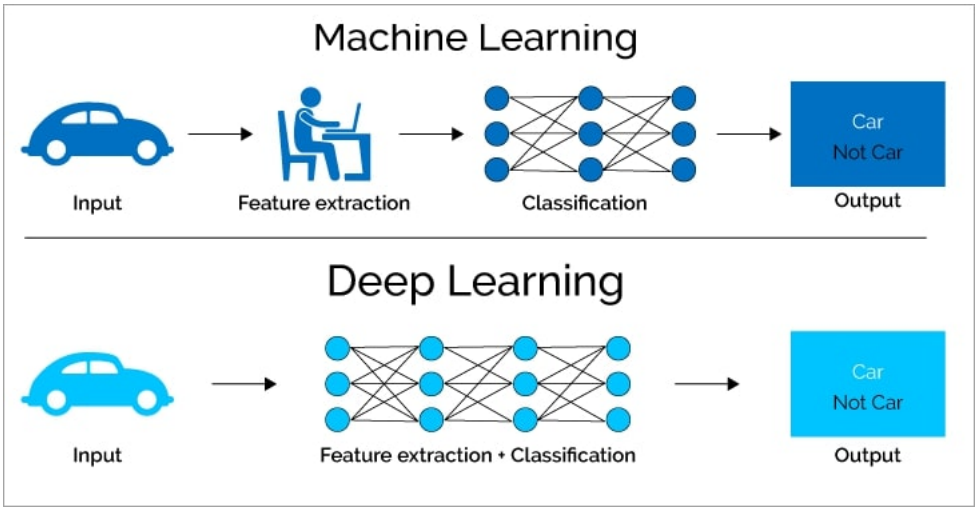


Figura 2.1: Comparación entre el Aprendizaje de Máquinas tradicional (imagen superior) y el Aprendizaje Profundo (imagen inferior). En el enfoque tradicional, se necesita realizar ingeniería de datos para que un clasificador pueda realizar predicciones, mientras que en el caso del Aprendizaje Profundo, las múltiples capas de la red neuronal artificial aprenden a extraer características y realizar la clasificación de forma automática. Imagen obtenida de [54].

2.4. Aprendizaje profundo

Como se mencionó en la sección anterior, el sistema presentado en este trabajo utiliza modelos del estado del arte del aprendizaje de máquinas, en particular, modelos basados en aprendizaje profundo. El aprendizaje profundo (conocido como *Deep Learning* en inglés) [20] es un tipo de aprendizaje de máquinas que, a diferencia de los métodos clásicos, no requiere la ingeniería manual de características para el funcionamiento de sus algoritmos. En lugar de eso, introduce el uso de redes neuronales artificiales (conocidas como *Artificial Neural Networks* en inglés) con múltiples capas. Estas capas permiten que la red aprenda y extraiga características directamente de los datos, comenzando desde las estructuras más básicas presentes en las imágenes, como líneas o texturas, y avanzando hacia las más complejas y abstractas [10]. La Figura 2.1 ilustra la diferencia entre ambos enfoques.

2.4.1. Redes neuronales artificiales

Las redes neuronales artificiales (ANNs por sus siglas en inglés) son modelos matemáticos y computacionales inspirados en las redes neuronales del cerebro humano. Similar a cómo las redes biológicas procesan información, las ANNs también están diseñadas para resolver problemas específicos, como clasificación o regresión. Aunque las ANNs se originaron en los años 40 y 50, ganaron prominencia principalmente en la década de 1980 con el desarrollo del algoritmo de retro-propagación (en inglés *Backpropagation*) [48]. No obstante, las limitaciones tecnológicas de esa época restringieron su evolución y aplicación. Gracias a los avances modernos, como la creación de GPUs (por sus siglas en inglés *Graphics Processing Unit*) más rápidas y la disponibilidad de grandes conjuntos de datos, las ANNs han experimentado un

resurgimiento significativo y una expansión en diversas áreas de la investigación.

Las ANN están compuestas por unidades básicas llamada neuronas, estas son una función matemática que recibe como entrada un vector x, a la que le se aplica una transformación lineal y posterior una función no lineal llamada función de activación [20]. A la agrupación de neuronas se le denomina como capa totalmente conectada (en inglés *fully connected layer*). Una capa producida por un conjunto de m neuronas queda representada en la ecuación (2.1), donde $x \in \mathbb{R}^n$ es el vector de entrada n, $W \in \mathbb{R}^{mxn}$ es la matriz de los parámetros conocida como pesos (en inglés *weights*), $b \in \mathbb{R}^m$ es un vector denominado sesgo (en inglés *bias*) y σ es la función de activación no lineal.

$$y = \sigma(Wx + b) \tag{2.1}$$

A la concatenación de múltiples capas se le llama ANN. Formalmente, para una red compuesta por k capas f_i con parámetros $\theta_i = [W_i, b_i]$ con $i \in \{1, ..., k\}$, la relación entre la entrada x y su salida y se expresa en la ecuación (2.2), donde $\theta = [\theta_1, ..., \theta_k]$.

$$y = f_\theta(x) = f_k(f_{k-1}(...(f_i(x))) \tag{2.2}$$

Exceptuando la última, todas las capas de una red neuronal se conocen como capas ocultas (o *hidden layers* en inglés). Estas capas ocultas pueden emplear una diversidad de funciones de activación. Entre las más tradicionales se encuentran la función sigmoide (o *sigmoid* en inglés), la tangente hiperbólica ($tanh$) y la unidad lineal rectificada ($ReLU$) [36]. Con el avance y desarrollo de redes más modernas, se han introducido numerosas funciones de activación, muchas de las cuales son variaciones o adaptaciones de las tres funciones previamente mencionadas.

La función de activación de la última capa suele definirse según el tipo de problema que se desea resolver. Para problemas de clasificación, se suelen utilizar funciones de activación que puedan representar la salida en probabilidades de clasificación, es decir, salidas acotadas entre 0 y 1. En clasificación binaria, se suele usar como función la sigmoide, donde la función da la probabilidad de la neurona de que el ejemplo de entrada sea o no de la clase en cuestión. En la clasificación multiclase, se requiere entregarle una probabilidad a K neuronas, de tal forma que la suma de todas estas probabilidades sea 1, donde K es el número de clases. La función que comprende este tipo de problemas es la función *softmax*, definida en la ecuación (2.3), donde z_i corresponde a la neurona i-ésima de la última capa con $i \in \{1, ..., K\}$.

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.3}$$

2.4.2. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNNs, por sus siglas en inglés *Convolutional Neural Network*), son redes neuronales artificiales diseñadas específicamente para el análisis y

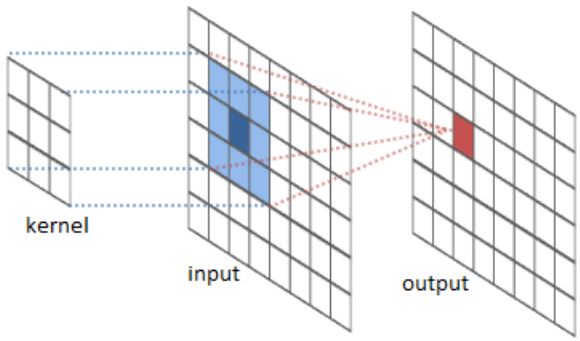


Figura 2.2: Diagrama ilustrativo de una convolución 2D. En esta representación, una imagen de entrada (input) es procesada mediante la aplicación de una convolución utilizando un filtro o núcleo (kernel) de dimensiones $3x3$. A medida que el filtro se desplaza a través de la imagen de entrada, va generando de manera progresiva una matriz de salida (output), que encapsula las características convolucionadas de la imagen de entrada. Imagen obtenida de [54].

procesamiento visual de datos, como imágenes o vídeos. Estas redes han revolucionado el campo de la visión computacional al inspirarse en la estructura y función del córtex visual del cerebro [20].

El componente distintivo de las CNNs son las capas convolucionales (en inglés *convolutional layers*), las cuales basan su función en la operación matemática convolución. Las capas convolucionales utilizan filtros o kernels que se convolucionan con la entrada para generar un mapa de características (en inglés *feature maps*) como salida. En la figura 2.2 se representa visualmente esta operación. Matemáticamente, la operación de convolución discreta utilizada en estas capas queda definida en la ecuación (2.4), donde x es la entrada y k es el filtro a aplicar.

$$x[m,n] * k[m,n] = \sum_{j=0}^{M} \sum_{i=0}^{N} x[i,j]k[m-i,n-j] \tag{2.4}$$

La operación convolucional implica aplicar un filtro k sobre toda la entrada x. Para esto, es necesario deslizar el filtro a través de la entrada, realizando este desplazamiento cada s píxeles, donde s se denomina *stride*. Como resultado de aplicar esta operación, la dimensión de la salida se reduce en comparación con la entrada; esta reducción está determinada por el tamaño del filtro aplicado y el *stride* utilizado. Una técnica común para compensar esta reducción espacial es el uso de *zero-padding*, que involucra el relleno de la salida con ceros en sus bordes, permitiendo de esta manera mantener la consistencia en la dimensión espacial entre la entrada y la salida.

Una de las principales ventajas de utilizar capas convolucionales está en la capacidad de aprovechar la información espacial contenida en las imágenes para la extracción de características mediante el uso de filtros o kernels. Otra ventaja significativa es la reducción del número de parámetros entrenables comparado con las capas *fully connected*. Esto se debe a que el único parámetro entrenable es el kernel, que usualmente es mucho menor en tamaño

que la entrada de la capa. Esto último ha permitido disminuir considerablemente los tiempos de entrenamiento, aumentando la eficiencia de las redes neuronales.

Similar a las capas *fully connected*, las capas convolucionales suelen estar acompañadas de funciones de activación para introducir no linealidades, las que permiten al modelo aprender funciones no lineales. Adicionalmente, si es necesario, se puede incluir otra operación no-lineal conocida como *pooling*, que permite reducir la dimensionalidad y a destacar las características más significativas, realizando operaciones sobre vecindarios específicos de la salida de la función de activación. Los tipos de *pooling* más comunes son *max pooling*, *average pooling* y *global average pooling*. En *max pooling*, se selecciona el valor máximo de un vecindario definido por una ventana deslizante. En *average pooling*, se calcula el promedio de los valores dentro de la ventana. En el *global average pooling*, se toma el promedio de todos los valores de la entrada, considerando toda la entrada como un único vecindario.

2.4.3. Entrenamiento de redes neuronales

Las redes neuronales artificiales son diseñadas para abordar y solucionar problemas específicos mediante un proceso conocido como entrenamiento (en inglés *training*). Este proceso es fundamental, ya que en situaciones prácticas, rara vez se tiene acceso a la distribución completa de datos a la que una ANN podría estar expuesta. Por lo tanto, esta se tiene que basar únicamente en un conjunto representativo de datos, comúnmente denominado conjunto de entrenamiento (en inglés *train set*). A través de este proceso, se ajustan los parámetros internos de la red con el objetivo de minimizar la discrepancia entre las predicciones de la red y los valores reales en el conjunto de entrenamiento. El objetivo final de esta, es que, una vez entrenada la red, esta sea capaz de hacer predicciones precisas sobre datos no vistos durante el entrenamiento.

Formalmente, el problema se define como, dada una red f_θ de parámetros entrenables $\theta \in \Theta$, un conjunto de entrenamiento X de tamaño N, representado como $\{(x_i, y_i)\}_{i=1}^{N}$. Se desea mapear una función $y = f(x)$, con $x \in X$. El objetivo del entrenamiento es encontrar el valor óptimo de θ que minimice una función de pérdida L, definida como la discrepancia entre las predicciones de la red y los verdaderos valores de salida. Matemáticamente, esto se puede expresar como en la ecuación (2.5), donde L es la función de pérdida (o en inglés *loss function*) y θ^* es el conjunto de parámetros que produce la menor pérdida promedio en el conjunto de entrenamiento.

$$\theta^* = \text{argmin}_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x_i), y_i) \tag{2.5}$$

La función de pérdida L entrega un valor que indica qué tan bien o mal la red está realizando predicciones en comparación con los datos reales (en inglés se suele conocer como *ground truth*). Entre las funciones de pérdida más comunes se encuentran:

- **Error Cuadrático Medio**: también conocido como MSE (por sus siglas en inglés *Mean Square Error*), es una función utilizada frecuentemente en problemas de regresión. Se

calcula como el promedio de las diferencias al cuadrado entre el valor predicho $\hat{y}$ por la red y por el valor real y de N ejemplos. Esta función viene dada por la ecuación (2.6).

$$L_{MSE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.6}$$

- **Entropía Cruzada**: también conocida CE (por sus siglas en inglés *Cross Entropy*) es una función utilizada frecuentemente en problemas de clasificación multi-clase. Esta cuantifica la diferencia entre dos distribuciones de probabilidad. Dada una distribución real y y una distribución estimada por la red $\hat{y}$ sobre K clases, la entropía cruzada CE se calcula como la ecuación (2.7).

$$L_{CE}(y, \hat{y}) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \tag{2.7}$$

- **Pérdida de Intersección sobre Unión**: también conocida como *IoU Loss*, es una función utilizada frecuentemente en problemas de segmentación y detección de objetos [68]. El *IoU Loss* mide la superposición entre dos regiones, normalizada por el área total cubierta por ambas. Dadas una región predicha $\hat{y}$ y una región verdadera y, la métrica IoU se define según la ecuación (2.8), donde $|y \cap \hat{y}|$ representa el área de intersección de las regiones y $|y \cup \hat{y}|$ indica el área de su unión. Para adaptar esta métrica como una función de pérdida, es decir, una función que tenga un valor menor cuando hay una mayor similitud entre y e $\hat{y}$, se formula según la ecuación (2.9).

$$IoU(y, \hat{y}) = \frac{|y \cap \hat{y}|}{|y \cup \hat{y}|} \tag{2.8}$$

$$L_{IoU} = 1 - IoU(y, \hat{y}) \tag{2.9}$$

Definido lo que son las funciones de pérdida, es necesario definir un proceso que permita la minimización de la función de perdida $L(\theta)$ mediante el ajuste de los parámetros del modelo θ, a este proceso se le denomina optimización. Entre los métodos de optimización más populares están:

- **Descenso de Gradiente Estocástico**: conocido como SGD (por sus siglas en inglés *Stocastic Gradient Descent*)[6], es una variante del clásico método descenso de gradiente. Similar al descenso de gradiente tradicional, SGD actualiza los parámetros en la dirección opuesta del gradiente de la función de pérdida $L(\theta)$ con respecto a dichos parámetros. Sin embargo, a diferencia del método tradicional donde la actualización se realiza utilizando todo el conjunto de datos, en SGD se efectúa utilizando pequeños lotes de muestras, conocidos como *batches*. Esta actualización está definida por la ecuación (2.10), donde θ_t representa los parámetros en la iteración t, η es la tasa de aprendizaje, y $\nabla L(\theta_t)$ es el gradiente de la función de pérdida con respecto a θ_t.

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t) \tag{2.10}$$

Existen diversas variantes del SGD que buscan optimizar su rendimiento. Una de ellas es *SGD con momentum* [40], que incorpora un término conocido como *momentum* para acelerar el SGD en la dirección más relevante y suavizar las oscilaciones durante la actualización de los parámetros. Otra variante es *Gradiente Acelerado de Nesterov* (NAG por sus siglas en inglés) [37], una versión mejorada del *momentum*, que en vez de calcular el gradiente con respecto a los parámetros actuales, lo hace con respecto a una aproximación futura de estos.

- **Estimación Adaptativa de Momentos**: conocido como ADAM (por sus siglas en inglés *Adaptive Moment Estimation*) [23], es un método que combina las ideas de dos optimizadores *Adagrad* [8] y *RMSprop*. Este optimizador actualiza los parámetros usando estimaciones de primer y segundo momento de los gradientes. Estos momentos son aproximadamente el promedio y la varianza no central de los gradientes, respectivamente. La actualización está dada por la ecuación (2.11), donde $\hat{m}_t$ y $\hat{v}_t$ son estimaciones del primer y segundo momento de los gradientes y ε es un pequeño valor para evitar divisiones por cero. ADAM a diferencia de SGD, ajusta la tasa de aprendizaje para cada parámetro individualmente.

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}} \tag{2.11}$$

Un cálculo esencial para la optimización en el entrenamiento de redes neuronales, es la determinación de los gradientes de la función de pérdida con respecto a cada uno de los parámetros o pesos de la red. Dado que una red puede contener una gran cantidad de parámetros, el cómputo de estos gradientes en cada iteración puede resultar ineficiente. Esto, a su vez, hace que el proceso de optimización se vuelva excesivamente costoso computacionalmente, tanto en términos de memoria como de procesamiento.

Para abordar este problema, en 1986 se desarrolló el algoritmo de retropropagación (en inglés *backpropagation*) [48]. Este algoritmo se fundamenta en la regla de la cadena, también conocida como el teorema de las funciones compuestas, y permite calcular de forma eficiente el gradiente de la función de pérdida con respecto a cada uno de los pesos de la red. El proceso comienza con un paso hacia adelante (*forward*), donde se introduce una entrada a la red y se obtiene una salida correspondiente. Seguidamente, se realiza un paso hacia atrás (*backward*), en el cual se determina el error comparando la salida obtenida con el valor real deseado y se propaga este error de vuelta a través de la red, desde la última capa hasta la primera. Durante este paso, se actualizan los parámetros en función de su contribución al error total.

2.5. Detección de objetos

La detección de objetos es una de las tareas fundamentales en el campo de la visión por computadora y el aprendizaje profundo. Su objetivo principal es localizar e identificar objetos específicos dentro de una imagen. Para esto, el enfoque principal es entrenar una red convolucional para que sea capaz de entregar puntos espaciales de la posición de los objetos

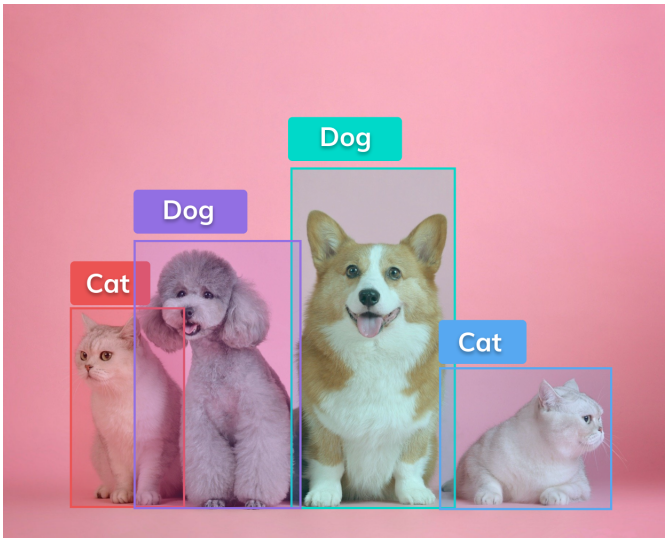


Figura 2.3: Representación de la detección de clases de objetos, donde los animales en la imagen son identificados y clasificados en las categorías *Dog* y *Cat*. Imagen obtenida de [45].

con respecto a la imagen de entrada. Sin embargo, los detectores van un paso más allá, ya que, además de la localización de los objetos, se busca identificar cada uno de ellos.

Para una mejor comprensión de este trabajo, se subdividen los detectores de objetos en dos categorías según el tipo de identificación que estos sean capaces de entregar a sus objetos, diferenciando así, detectores de clases de objetos y detectores de instancias de objetos.

2.5.1. Detectores de clases de objetos

Es el tipo de detector más usual. Un detector de clases de objetos, es un detector que aparte de localizar objetos, es capaz de identificar a estos en clases o categorías predefinidas en el entrenamiento. Por ejemplo, si se tuviera una imagen de una mesa con diferentes tipos de tazas, tazas con distintas formas y color, el detector etiquetaría cada una de como objetos de la clase taza. Otro ejemplo gráfico se aprecia en la figura 2.3, donde a pesar de que cada mascota es diferente una de otra, estas se etiquetan únicamente dentro de las clases *Dog* y *Cat*.

Dentro de los detectores de clases de objetos, se diferencian dos enfoques principales: los detectores de dos etapas, conocidos en inglés como *Two-stage Detectors*, y los detectores de una sola etapa, conocidos como *One-stage Detectors*.

Detectores de dos etapas

Este tipo de detectores, dividen la tarea de detección en dos etapas. Una primera etapa de propuestas de regiones de interés (en inglés *region proposals*) y una segunda etapa de clasificación y ajuste de estas. A continuación se detallará ambas etapas:

- **Propuesta de regiones de interés**: esta etapa consiste en la generación de múltiples propuestas de áreas o regiones de interés, abreviadas como RoI (por sus siglas en inglés *Region of Interest*). El objetivo en este punto es identificar posibles ubicaciones dentro de la imagen que puedan contener un objeto, sin realizar aún una clasificación específica de los mismos.
- **Clasificación y Ajuste de Detecciones**: esta etapa implica procesar cada una de las regiones de interés identificadas previamente. Aquí, se clasifica cada región en una de las clases de objetos predefinidas o, si no contiene un objeto relevante, se marca como fondo (*background*). Además, se realiza un ajuste fino de los bordes de cada región mediante técnicas de regresión para lograr una localización más exacta del objeto.

Entre las redes de detección más populares basadas en este enfoque están R-CNN [14], Fast R-CNN [13] y Faster R-CNN [44].

Detectores de una etapa

Unas de las desventajas de los detectores anteriores, es que el proceso de entrenamiento tiende a ser extenso, debido a que es necesario considerar entrenamientos por cada etapa de forma independiente y de forma conjunta para que el modelo completo alcance los resultados deseados. Es por esto que surge la necesidad de la creación de detectores de una sola etapa, que puedan fusionar las propuestas de regiones y la clasificación en una sola fase. Para lograr esto, la red debe ser capaz de localizar y clasificar objetos en toda la imagen de entrada en una sola pasada. Estos detectores al ser de una sola pasada, son más eficientes que los detectores de 2 pasadas.

Entre las redes de detección más populares basadas en este enfoque están RetinaNet [28], FCOS [58], SSD [31], y todas las versiones y variaciones de YOLO [42][43][11][1][61].

2.5.2. Ejemplos de detectores de clases de objetos

A continuación se presentan ejemplos de los detectores de clases de objetos que más influyeron en el desarrollo del modelo presentado en esta tesis.

Faster R-CNN

Faster R-CNN [44] es la continuación de las redes R-CNN [13] y Fast R-CNN [44], todos modelos de detección de 2 etapas. Esta red viene a solucionar el problema de eficiencia en la generación de propuestas de su antecesor Fast R-CNN. Para esto proponen una Red de Propuestas de Regiones (RPN por sus siglas en inglés *Region Proposal Network*) que comparta capas convolucionales con la red de detección final. RPN en vez de usar una arquitectura piramidal para trabajar el rango de escalas de los objetos, introduce el uso de anclajes (en inglés *anchors*). La arquitectura general de esta red se puede observar en la figura 2.5.

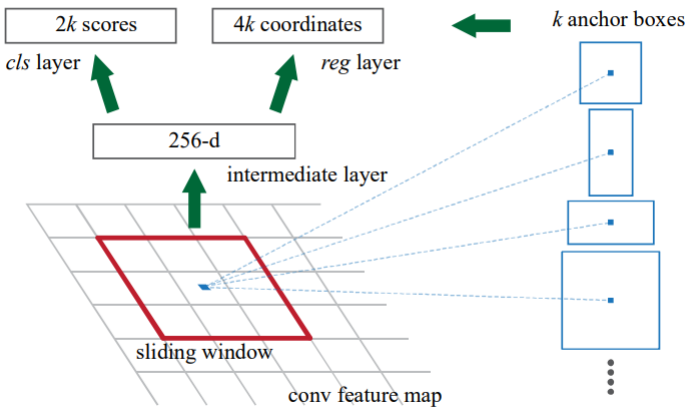


Figura 2.4: Red de Propuesta de Regiones (RPN). Imagen obtenida de [44].

Entre las innovaciones claves de esta red están:

- **Red de Propuesta de Regiones (RPN)**: Para la generación de propuestas de regiones, se utiliza una pequeña red neuronal que recorre el *feature map* resultante de la primera CNN compartida en la arquitectura. Esta red procesa ventanas de entrada de dimensión $n \times n$, donde comúnmente $n = 3$. Cada ventana se transforma en una representación de baja dimensión, ya sea 256-d o 512-d. Estas representaciones se dirigen posteriormente a dos capas *fully connected* paralelas: una dedicada a la regresión de la caja delimitadora y otra enfocada en la clasificación objeto/fondo. En la figura 2.4 se representa esta red.
- **Cajas de Anclaje**: En inglés conocidas como *anchors boxes*, son cuadros delimitadores de referencia utilizados para predecir la localización de objetos. Estos cubren múltiples escalas y relaciones de aspecto, lo que permite a la RPN detectar objetos de diversas formas y tamaños. Por cada ventana deslizante, se tiene k propuestas, es decir, k *anchors*. Se usan 3 escalas y 3 relaciones de aspecto, lo que quiere decir que $k = 9$. En la figura 2.4 se observan la forma de los *anchors* en una ventana.
- **Compartición de características para RPN y Fast R-CNN**: Faster R-CNN propone una estructura que comparte la extracción de características entre la RPN y la red de detección principal Fast R-CNN. En [44], se discuten 3 formas de entrenar redes con características compartidas, sin embargo, adoptan un algoritmo de entrenamiento de 4 pasos, denominado *Step Alternating Training* para aprender características compartidas mediante optimizaciones alternadas, el cual se explica a continuación:
  1. **Primer paso**: Se entrena la Red de Propuestas de Regiones. Esta red se inicializa con un modelo previamente entrenado con ImageNet [2] y posteriormente se ajusta (*fine-tuning*) para la tarea específica de generación de propuestas de regiones.
  2. **Segundo paso**: De forma independiente, se entrena la red de detección Fast R-CNN, utilizando como entrada las propuestas generadas en el primer paso. Al igual que la RPN, esta red de detección también se inicia con un modelo preentrenado en ImageNet. Hasta este momento, ambas redes no comparten los pesos de sus capas convolucionales.

3. **Tercer paso**: La red de detección ya entrenada sirve para inicializar la RPN, pero se congelan las capas convolucionales que ambas redes comparten, realizando *fine-tuning* únicamente en las capas propias de la RPN. Esto permite que ambas redes comiencen a compartir las mismas capas convolucionales.
4. **Cuarto paso**: Manteniendo fijas las capas convolucionales compartidas, se realiza el ajuste fino (*fine-tuning*) de las capas de detección de la red Fast R-CNN.

- **Función de pérdida de detección**: Para definir la función de pérdida de esta red, se deben considerar las siguientes definiciones: un *anchor* puede ser asignado como positivo de dos formas (i) si es el *anchor* con el mayor IoU en comparación con un *ground truth box*, o (ii) si tiene un IoU superior a 0,7 con cualquier *ground truth box*. Por otro lado, se asigna como negativo a aquel *anchor* cuyo IoU es menor a 0.3 con cualquier *ground truth box*. Cualquier *anchor* que no cumpla con estas condiciones será ignorado. A partir de estas definiciones, se formula la función de pérdida multitarea como se muestra en la ecuación (2.12). Aquí, i es el índice de un *anchor* en un mini-lote y p_i es la probabilidad predicha de que el *anchor* i contenga un objeto. El valor de *ground truth* es 1 si el *anchor* es positivo y 0 si es negativo. t_i es un vector que representa las 4 coordenadas parametrizadas del *bounding box* predicho, y t_i^* corresponde al *ground truth box* asociado con un *anchor* positivo.

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} p_i^* L_{reg}(t_i, t_i^*) \tag{2.12}$$

Faster R-CNN demostró una mejora significativa en velocidad en comparación con Fast R-CNN, sin sacrificar la precisión. De hecho, Faster R-CNN superó a sus predecesores en términos de precisión. Gracias a la RPN, la necesidad de métodos externos, como *Selective Search* [59] para la generación de propuestas, se eliminó completamente, resultando en un *pipeline* más limpio y rápido.

YOLO (You Only Look Once)

YOLO (You Only Look Once) [42] representa un nuevo enfoque para la detección de objetos. A diferencia de la Faster R-CNN y otros métodos tradicionales hasta la fecha, YOLO enmarca la detección de objetos como un solo problema de regresión de *bounding boxes* espacialmente separados y probabilidades de clase asociadas, es decir, corresponde a un detector de una sola etapa que predice las detecciones directamente a partir de una imagen completa en una sola evaluación. Gracias a esto, YOLO se puede optimizar de extremo a extremo en el rendimiento de detección. En la figura 2.6 se puede visualizar el sistema general de YOLO.

Entre las características más destacadas de esta red están:

- **Detección unificada**: para realizar la detección en una sola pasada, YOLO divide la imagen en una cuadrícula de SxS. Para cada celda de la cuadrícula se predicen B *bounding boxes* junto sus confianzas asociadas (que tan probable que el *bbox* contenga un objeto), por ende, cada *bbox* consiste en 5 predicciones: x, y, w, h y la confianza.

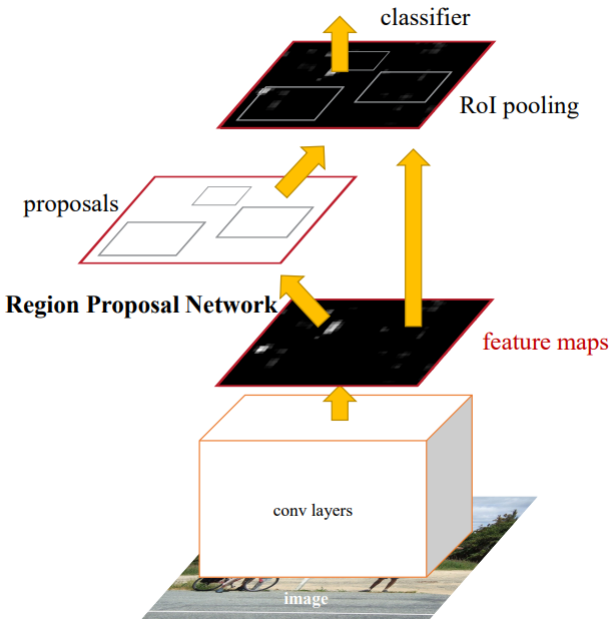


Figura 2.5: Arquitectura de Faster R-CNN. Imagen obtenida de [44].

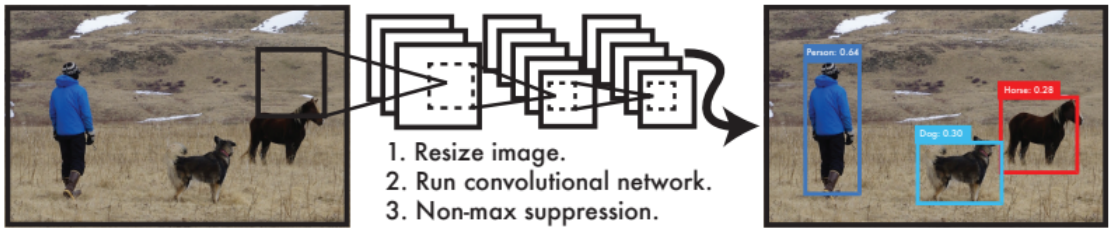


Figura 2.6: Proceso de detección utilizando YOLO. El sistema redimensiona primero la imagen de entrada a un tamaño de 448×448, luego aplica una red neuronal convolucional única y, por último, realiza una supresión no máxima (*Non-max suppression*) para filtrar las detecciones por la confianza del modelo. Imagen obtenida de [42].

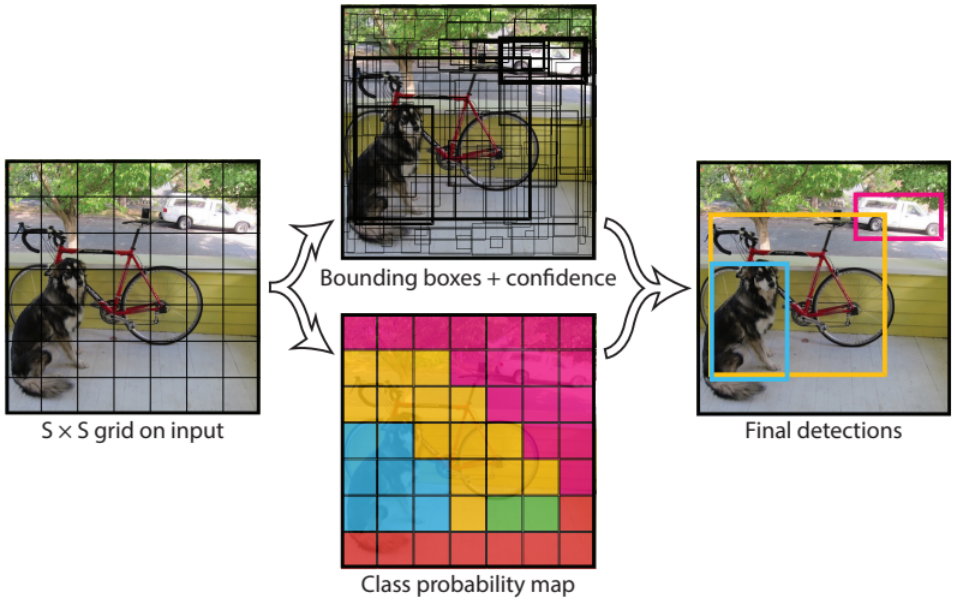


Figura 2.7: La imagen es dividida en una cuadrícula de tamaño SxS, para cada celda se predicen cajas delimitadoras y sus respectivas confianzas. Paralelamente, se crea un mapa de probabilidad de clases. Finalmente, las detecciones son refinadas para mostrar las cajas delimitadoras finales sobre los objetos identificados. Imagen obtenida de [42].

Adicionalmente, para cada celda se predice una distribución de probabilidad de clase para C clases. Así las predicciones de detección de cada evaluación pueden ser codificadas como un tensor de SxSx$(B * 5 + C)$. En la figura 2.7 se representa visualmente como YOLO modela el problema de detección.

- **Función de pérdida unificada**: se propone una función de pérdida que considera simultáneamente tanto la localización del objeto como la clasificación. Durante el entrenamiento, se busca que un predictor de cuadro delimitador sea responsable de cada objeto, esto se hace con el predictor que tenga un IOU más alto con respecto al *ground truth*. La función de pérdida se presenta en la ecuación (2.13), donde la primera línea corresponde a la regresión del centro de los *bboxes*, la segunda del ancho y alto, la tercera a la certeza de la existencia del objeto, la cuarta si la celda i no contiene objeto y la última la probabilidad de la clase en la celda i.

$$
\begin{aligned}
L = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y})^2] \\
+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
+\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
+\sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{2.13}
$$

- **Velocidad y eficiencia**: Gracias a la arquitectura unificada, YOLO es una red extremadamente rápida con respecto a los detectores de esa época. El modelo YOLO base procesa imágenes en tiempo real a 45 fotogramas por segundo (también conocido como *FPS*), mientras que Fast R-CNN procesa a 0,7 y Faster R-CNN a 7.

Aunque YOLO supera a Faster R-CNN en términos de velocidad de detección, no lo hace en precisión, alcanzando un mAP (*Mean Average Precision*) de 63,4 en comparación con el 73,2 de Faster R-CNN utilizando VGG-16 [53] como *backbone*. YOLO tiende a cometer más errores en la localización de objetos pero tiene menor tendencia a generar falsos positivos en el fondo.

YOLO es el punto de partida de una serie de versiones en base a este modelo, cada versión nueva viene a mejorar problemas, la velocidad y precisión del modelo anterior. YOLO9000 [43], también conocido como YOLOv2, se diseñó para solucionar problemas de localización identificados en la versión original, introduciendo mejoras como la Normalización por Lotes (*Batch Normalization*), el incremento de resolución para la clasificación, el uso de *anchors* optimizados por *clustering*, predicciones directas de la localización y entrenamiento multi-escala. YOLOv3 [11] mejoró aún más el modelo al predecir la confianza mediante regresión logística, utilizar entropía cruzada binaria para la clasificación, y la implementación de una Red Piramidal de Características (FPN) en tres escalas diferentes, además de integrar Darknet-53, un híbrido entre Darknet-19 y redes residuales, como extractor de características. YOLOv4 [1] continuó con la innovación al integrar Conexiones Parciales en Etapas Cruzadas (CSP) en el *backbone*, nuevas técnicas de aumento de datos como *Cutmix* y *Mosaic Data*, suavizado de clases, y reemplazo de la regresión para la localización por CIoU, además de la incorporación de bloques SAM y PAN modificados en el detector. Finalmente, YOLOR [61] propone una red unificada capaz de realizar múltiples tareas a través de la integración de conocimiento implícito y explícito

Aunque se han desarrollado más versiones de YOLO que las mencionadas, estas han sido producto del trabajo de otras organizaciones, como *Ultralytics* con el desarrollo de YOLOv5

y la reciente YOLOv8. Estas variantes no se han tomado en cuenta para este trabajo debido a la falta de documentación académica que explique las contribuciones específicas de cada una de estas redes.

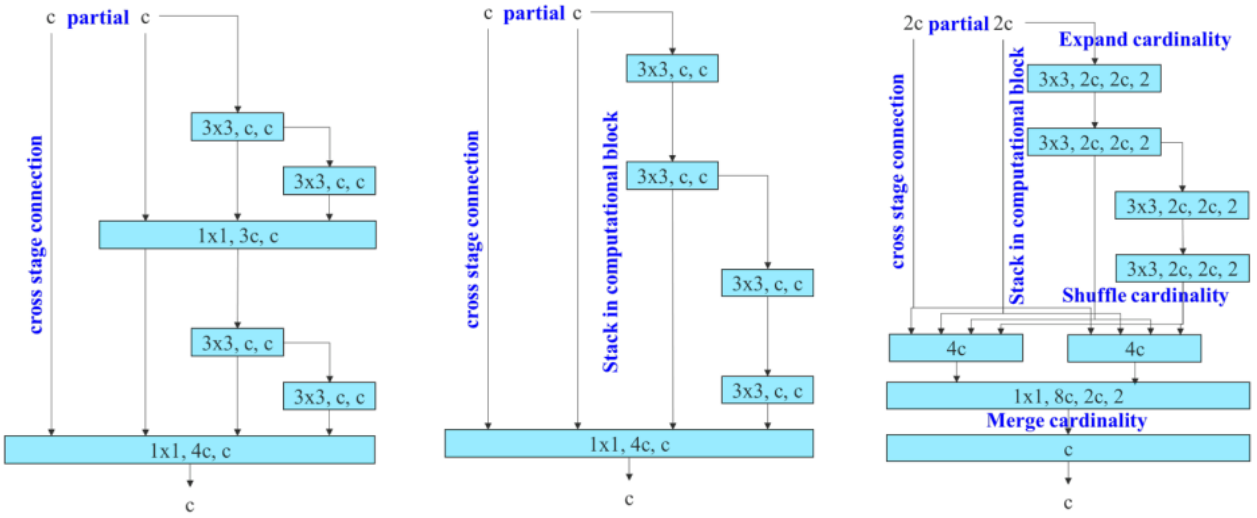
2.5.3. YOLOv7

YOLOv7 [63] corresponde a la versión más nueva de YOLO publicada por Chien-Yao Wang, mismo creador y contribuidor original de YOLOR y YOLOv4. YOLOv7 es la última iteración de este enfoque y destaca por superar a otros detectores tanto en términos de velocidad como en precisión. Se ha entrenado con el conjunto de datos COCO [29], sin utilizar otros conjuntos de datos o pesos pre entrenados, y logra una velocidad y precisión excepcionales en la detección en tiempo real, superando a otros detectores contemporáneos. YOLOv7 introduce varios métodos optimizados que no solo mejoran la precisión de la detección sino que también mantienen bajos los tiempos de inferencia. Estas optimizaciones incluyen la reparametrización del modelo, la asignación dinámica de etiquetas métodos de escalado extendido y escalado compuesto. A continuación, se detallarán estas optimizaciones.

Las contribuciones de esta red se pueden dividir en tres categorías: arquitectura, técnicas de entrenamiento y función de pérdida.

Arquitectura

- **Red basada en módulos E-ELAN**: En la búsqueda de arquitecturas eficientes, tradicionalmente se consideran aspectos como el número de parámetros, la cantidad de cálculos y la densidad computacional. CSPVoVNet [62], una variante de VoVNet [26], incorpora análisis de gradientes para acelerar inferencias y mejorar precisión, permitiendo que los pesos de diferentes capas aprendan características más variadas. Por otro lado, ELAN [60] propone diseñar redes eficientes controlando el camino del gradiente más corto, lo que permite que una red más profunda aprenda y converja eficazmente. YOLOv7 introduce el *Extended-ELAN* (E-ELAN) que, manteniendo el análisis del camino del gradiente, permite una mejora continua en la capacidad de aprendizaje sin destruir el camino original del gradiente. E-ELAN logra esto a través de la expansión, mezcla y agrupación. Específicamente, utiliza la convolución de grupo para expandir canales y cardinalidad, y luego aplica operaciones de barajado y concatenación. Esta estructura guía a diferentes grupos para aprender características más diversificadas, manteniendo a la vez la estructura original de ELAN. Estos modelos se pueden ver en la figura 2.8.
- **Escalado de modelos basados en concatenación**: se refiere a la adaptación de ciertos atributos del modelo para generar variaciones que satisfagan diferentes velocidades de inferencia. EfficientNet, por ejemplo, escala en función del ancho, profundidad y resolución mientras que en Scaled-YOLOv4 [62] se modifican el número de etapas. Estas estrategias de escalado funcionan bien en arquitecturas como PlainNet o ResNet [18]. Sin embargo, al aplicarse a arquitecturas basadas en concatenación, se presentan problemas, si se realiza un escalado en profundidad, el grado de entrada de una capa



(a) CSPVoVNet [62] (b) ELAN [60] (c) E-ELAN [63]

Figura 2.8: Arquitecturas de módulos en los que se basa YOLOv7, (a) CSPVoVNet propone una conexión cruzada parcial para mejorar la eficiencia de la red, (b) ELAN introduce un bloque de conexión en el estado computacional con la intención de optimizar el rendimiento, y (c) E-ELAN amplía la cardinalidad de la red, utilizando operaciones de expansión, barajado y fusión para aumentar la capacidad representacional sin un coste computacional significativo. Imagen obtenida de [63].

de transición inmediatamente después de un bloque computacional basado en concatenación puede variar. De esta observación, se desprende que no se pueden analizar diferentes factores de escalado de manera independiente para un modelo basado en concatenación. La solución propuesta es un método de escalado compuesto específico para modelos basados en concatenación: al escalar el factor de profundidad de un bloque computacional, se debe calcular el cambio en el canal de salida de ese bloque. Luego, se realiza un escalado del factor de ancho en las capas de transición correspondientes. Este método conserva las propiedades originales del modelo, manteniendo una estructura óptima.

Técnicas de entrenamiento

- **Convolución re-parametrizada planeada**: La técnica RepConv [5] ha demostrado ser efectiva con la arquitectura VGG. Sin embargo, al aplicarla directamente a ResNet y DenseNet, su precisión disminuye significativamente. Mediante el análisis del flujo de gradiente, se descubre que la conexión de identidad en RepConv afecta negativamente a las características residuales y de concatenación presentes en ResNet y DenseNet. Como solución, se propone una variante de RepConv sin conexión de identidad, denominada RepConvN, que se adapta mejor a estas arquitecturas. Esta propuesta asegura una integración más armoniosa y efectiva de la convolución re-parametrizada en diversas arquitecturas de red.
- **Supervisión profunda**: esta técnica se utiliza a menudo en la formación de redes profundas, incorporando cabezas auxiliares en capas intermedias para guiar el apren-

dizaje de las redes. En este contexto, se introduce la idea de una cabeza principal (*lead head*) para la salida final y cabezas auxiliares (*auxiliary head*) para asistencia en el entrenamiento.

- **Asignación de etiquetas**: Un desafío emergente es la asignación de etiquetas. Tradicionalmente, se asignaban etiquetas duras basadas en la verdad fundamental, pero recientemente, la tendencia es hacia etiquetas suaves que consideran tanto las predicciones de la red como la verdad fundamental. Se propone una nueva estrategia de asignación de etiquetas que se guía por las predicciones de la cabeza principal, generando etiquetas jerárquicas de grano grueso a fino.
- **Normalización por lotes en la topología conv-bn-activación**: Este enfoque conecta directamente la capa de normalización por lotes a la capa convolucional. La finalidad es integrar la media y varianza de la normalización por lotes en el sesgo y peso de la capa convolucional durante la etapa de inferencia.
- **Conocimiento implícito**: Se refiere a la simplificación de la información implícita en YOLOR [61] a un vector, que puede ser pre calculado en la fase de inferencia. Este vector puede ser combinado con el sesgo y peso de la capa convolucional anterior o posterior.
- **Modelo EMA**: EMA [56] es una técnica usada en el método *mean teacher*. En este sistema, el modelo EMA se utiliza puramente como el modelo final de inferencia.

Función de pérdida

La función de pérdida utilizada en YOLOv7 viene dada por la ecuación (2.14). Esta corresponde a la suma de la función de pérdida de la caja delimitadora (L_{bbox}), la función de pérdida del objeto (L_{obj}) y la función de pérdida de clasificación (L_{class}).

$$L_{YOLOv7} = L_{bbox} + L_{obj} + L_{class} \tag{2.14}$$

- L_{bbox}: es la función encargada de la localización, se utiliza CIoU (*Complete-IoU*) [67]. Es una versión mejorada de la métrica *Intersection over Union* (IoU). Considera no solo la superposición de las cajas delimitadoras predichas y reales, sino también la distancia entre los centros de las cajas y aspectos de proporción. En la fórmula (2.15), b es la caja delimitadora predicha y b_{gt} es la caja delimitadora real (*ground truth box*). El término ρ denota la distancia euclidiana entre los centros de las dos cajas, mientras que c es la diagonal de la caja delimitadora más pequeña que contiene ambas cajas. El αv es un factor de corrección que penaliza las discrepancias en la proporción de las cajas.

$$L_{bbox} = 1 - IoU + \frac{\rho(b, b^{gt})}{c^2} + \alpha v \tag{2.15}$$

- L_{obj}: es la función encargada de la confianza si una predicción es o no objeto. Se divide en 2 partes, una para los objetos reales y otra para los no-objetos (fondo o *background*). En la ecuación (2.16) la primera parte suma sobre todas las celdas y cajas delimitadoras

donde hay un objeto real. I_{ij}^{obj} es una variable indicadora, que es 1 si hay un objeto en la caja j de la celda i, y 0 en caso contrario. $\hat{C}_i$ es la confianza predicha de que hay un objeto, y C_i es la confianza real (1 o 0). La segunda parte suma sobre celdas y cajas donde NO hay un objeto. λ_{noobj} es un factor de escala para penalizar las predicciones falsas. I_{ij}^{noobj} es similar a I_{ij}^{obj} pero para no-objetos.

$$\begin{aligned} L_{obj} = &\sum_{i=0}^{S^2}\sum_{j=0}^{B} I_{ij}^{obj}[\hat{C}_i log(C_i) + (1-\hat{C}_i) log(1-C_i)] \\ -\lambda_{noobj} &\sum_{i=0}^{S^2}\sum_{j=0}^{B} I_{ij}^{noobj}[\hat{C}_i log(C_i) + (1-\hat{C}_i) log(1-C_i)] \end{aligned} \tag{2.16}$$

- L_{class}: esta función de pérdida de clasificación se basa en la entropía cruzada binaria. Esta función suma las discrepancias entre las probabilidades predichas $p_i(c)$ y las reales para cada clase c dentro de una celda. El término $log(p_i(c))$ penaliza las predicciones erróneas cuando un objeto de clase c está presente, mientras que el término $log(1-p_i(c))$ hace lo mismo cuando el objeto no está presente. La suma se lleva a cabo solo para celdas con objetos, denotado por I_{ij}^{obj} y se extiende a través de todas las clases y celdas de la imagen.

$$L_{class} = \sum_{i=0}^{S^2} \mathrm{I}_{ij}^{obj} \sum_{c \in classes} [\hat{p}_i(c) log(p_i(c)) + (1-\hat{p}_i(c)) log(1-p_i(c))] \tag{2.17}$$

2.5.4. Detectores de instancias de objetos

Mientras que los detectores de clases de objetos están diseñados para identificar y localizar objetos que pertenecen a categorías predefinidas (como *perro*, *coche*, *persona*), los detectores de instancias de objeto tienen la tarea de localizar y reconocer objetos individuales, independientemente de si estos objetos pertenecen o no a una clase específica vista durante el entrenamiento. Esta capacidad es especialmente útil en aplicaciones como la búsqueda visual, donde el objetivo es encontrar un objeto específico en una imagen, incluso si ese objeto no es parte de las clases con las que el modelo fue entrenado.

La metodología común detrás de estos sistemas suele tener los siguientes pasos:

1. **Detección de propuestas**: Antes de identificar un objeto particular, el sistema necesita saber dónde podría estar en la imagen. Usando un detector de propuestas, el modelo sugiere múltiples regiones o propuestas que podrían contener el objeto de interés. Estas propuestas se generan basándose en diferentes características de la imagen, como bordes, texturas y colores.
2. **Extracción de características**: Una vez que se tienen las propuestas, se extraen características de cada una de ellas. Esta extracción convierte las propuestas en vectores numéricos que representan la información contenida en esa región de la imagen. Estos vectores capturan detalles importantes que permiten al sistema diferenciar una instancia de objeto de otra.

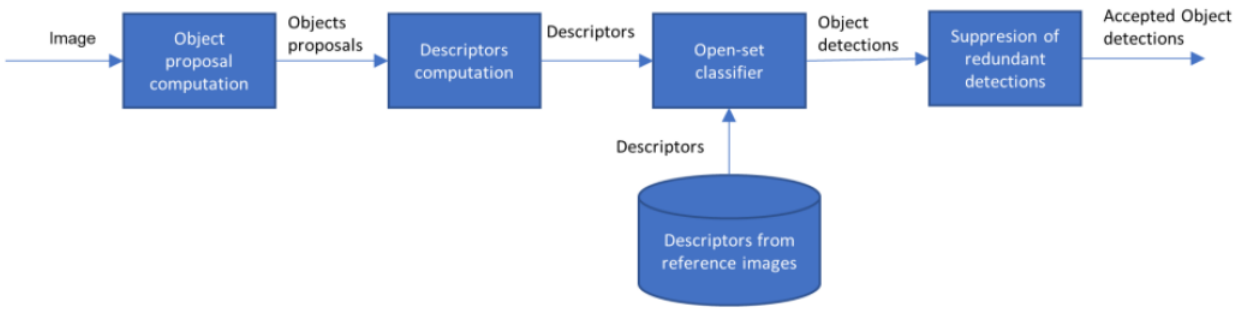


Figura 2.9: Diagrama de bloques del sistema YOLOSPoC. Imagen obtenida de [33].

3. **Comparación de vectores**: Finalmente, los vectores extraídos de las propuestas se comparan con los vectores de imágenes de referencia. Estas imágenes de referencia son representaciones de los objetos que se quieren detectar. Al calcular la similitud o la distancia entre el vector de la propuesta y el vector de referencia, el sistema puede determinar si la propuesta contiene o no el objeto de interés.

2.5.5. YOLOSPoC

YOLOSPoC [33] es un método propuesto por los académicos Patricio Loncomilla y Javier Ruiz-del-Solar, para el reconocimiento de instancias de objetos particulares. YOLOSPoC tiene el objetivo de permitir a un robot realizar tareas como la búsqueda de objetos del hogar en un lugar específico de la casa, permitiendo al robot resolver las problemáticas de *¿dónde está mi taza?* o *¿dónde está mi bebida?*.

La metodología YOLOSPoC se basa en cuatro bloques principales, que se visualizan en la figura 2.9 y se presentan a continuación:

- **Generación de propuestas de objetos**: Se utiliza YOLOv3 previamente entrenado en el conjunto de datos COCO, para generar propuestas de objetos. Sin embargo, se reduce el umbral de confianza con el fin de abarcar un espectro más amplio de detecciones, lo que facilita la identificación de objetos que no se encuentran entre las 80 clases predefinidas en COCO.
- **Cómputo de descriptores**: Para representar las propuestas generadas, se recurre a descriptores globales basados en las capas convolucionales de ResNet-101. Estos descriptores son robustos y ofrecen tolerancia ante variaciones leves en traslación y escala, aspectos fundamentales para el reconocimiento consistente de objetos bajo distintas condiciones. La metodología de entrenamiento empleada para optimizar estos descriptores es la de redes siamesas, mientras que para la generación de los descriptores finales se utiliza el enfoque SPoC (*Sum-Pooled Convolutional Features)*).
- **Clasificador de vecino más cercano en conjunto abierto**: Una vez que se tienen los descriptores, un clasificador vecino más cercano los compara con un conjunto de referencia. Sin embargo, para manejar objetos desconocidos, se emplea un enfoque de conjunto abierto (en inglés, *open-set classifier*), que puede rechazar objetos que no estén

en el conjunto de entrenamiento. La regla del clasificador viene dado por la ecuación (2.18), donde d_1 es la distancia euclidiana entre el descriptor de prueba y el descriptor más cercano en la base de datos, y d_2 es la distancia euclidiana entre el descriptor de prueba y el descriptor más cercano de un objeto diferente en la base de datos. En caso de que no se rechace el objeto, se identifica a este con la clase del descriptor de referencia más cercano.

$$\frac{d_2}{d_1} < \text{umbral} \rightarrow \text{rechazado} \tag{2.18}$$

- **Supresión de detecciones redundantes**: Para manejar múltiples detecciones del mismo objeto, YOLOSPoC implementa un proceso de post procesamiento que elimina detecciones redundantes, conservando solo las más relevantes. Las detecciones relacionadas con un mismo objeto se ordenan por su tamaño. Si el cuadro delimitador de una detección dada está completamente contenido dentro de otro cuadro delimitador del mismo objeto, se elimina. Se utiliza un umbral de tolerancia de 30 píxeles para decidir cuándo una detección de objeto está contenida dentro de otro objeto.

2.6. Aprendizaje métrico supervisado

En muchos problemas de aprendizaje supervisado, como la clasificación de imágenes o la recuperación de objetos en imágenes (en inglés *object retrieval*), la representación resultante de los datos obtenida por los modelos puede no ser adecuada o informativa para realizar la tarea deseada. El aprendizaje métrico supervisado surge como una solución a este problema al proporcionar una nueva representación de los datos que resalta las características relevantes para la tarea en cuestión. Este es un enfoque en el que se utiliza información etiquetada para aprender una métrica o distancia entre pares de instancias en un conjunto de datos. El objetivo principal es aprender una función de distancia de tal manera que las instancias similares se acerquen entre sí en el espacio transformado y las instancias no similares se alejen. Esta función de distancia aprendida puede utilizarse posteriormente para tareas como la clasificación, la recuperación de información y agrupación. Entre los métodos más reconocidos están *Siamese Neural Networks* [25], que utiliza pares de instancias y aprende a diferenciar entre pares similares y no similares; *Triplet Network* [21], que se basa en tripletes de instancias, una instancia positiva, una negativa y otra de referencia, donde el objetivo es que las representaciones de la instancia de referencia estén más cercanas a las de la instancia positiva que de la negativa; y los métodos basados en margen.

2.6.1. Métodos basados en margen

Estos métodos de aprendizaje métrico supervisado se enfocan en ajustar el margen entre diferentes clases o instancias en un espacio latente o de características. Estos enfoques consideran que es esencial no solo aprender una representación donde instancias similares estén cerca entre sí y las diferentes estén alejadas, sino también garantizar un margen claro entre ellas para mejorar la generalización del modelo. Esencialmente, estos métodos buscan

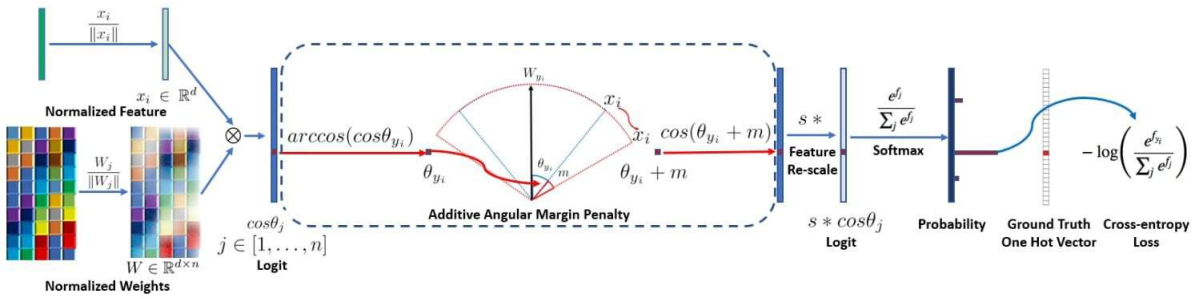


Figura 2.10: Entrenamiento supervisado de una DCNN con ArcFace. Imagen obtenida de [4].

encontrar un equilibrio entre acercar las instancias similares y alejar las diferentes, garantizando un margen claro para una decisión más confiable. El método más destacado de este tipo de enfoques ha sido *ArcFace* [4] y su variante *Sub-Center ArcFace* [3] que son explicados a continuación.

ArcFace

ArcFace [4] es una técnica diseñada específicamente para mejorar la precisión en tareas de reconocimiento facial. Esta introduce una función de pérdida aditiva angular, que penaliza directamente la diferencia entre los ángulos de las características y sus respectivas etiquetas de clase. El objetivo es mejorar el poder discriminativo la función de pérdida *softmax*, es decir, aumentar la compactibilidad intra-clases (cercanía de ejemplos de una clase al centro de esa clase, dada por el *target*) y la distancia inter-clases (lejanía entre clases u identidades distintas).

La inspiración detrás de ArcFace es la observación de que los parámetros de la última capa completamente conectada de una red neuronal convolucional profunda que se entrena con la función de pérdida *softmax* tienen correlación con cada una de las clases de salida.Estos centros pueden entenderse como una representación promedio de todas las características dentro de una clase particular. Para mejorar la discriminación del modelo, se introduce una penalización en el margen angular entre las características extraídas y los centros de clase. Específicamente, penaliza el arco-coseno del *logit*, que es el resultado del producto punto entre el vector de características de la penúltima capa y los pesos de la última capa, ambos normalizados. Este penalizador es aditivo y está modulado por un factor m. La razón detrás de la eficacia de ArcFace es su propiedad geométrica única, ya que al penalizar el margen en términos angulares, el modelo introduce un margen que tiene una correspondencia directa con la distancia geodésica en el espacio de características.

Los pasos del algoritmo para su entrenamiento se visualizan en la figura 2.10 y se explican a continuación:

1. Se normaliza el vector de características $x_i \in \mathbb{R}^d$ resultante de la penúltima capa y los pesos de la última capa del clasificador $W \in \mathbb{R}^{dxn}$, de tal forma que $\|x_i\| = 1$ y $\|W\| = 1$, donde d es la dimensión del vector y n la cantidad de clases.

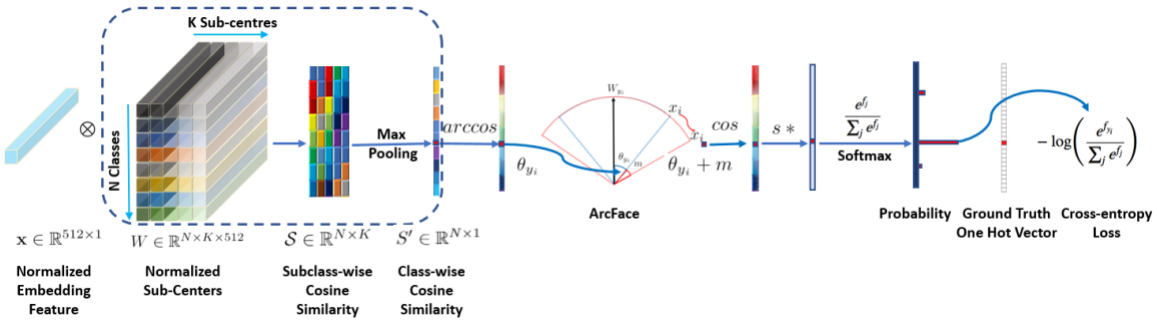


Figura 2.11: Entrenamiento supervisado de una DCNN con Sub-center ArcFace. Imagen obtenida de [3].

2. Se calcula el *logit* como $cos(\theta) = x_i \otimes W$, donde $cos(\theta) \in \mathbb{R}^{1xn}$. Cabe mencionar que $cos(\theta_j) = W_j^T x_i$ corresponde al *logit* de la clase j.
3. Se aplica *arc* a *logit*, se le suma una penalización m y se aplica coseno para volver a obtener el *logit*, el *logit* resultante queda $cos(\theta_j + m)$.
4. Se re-escala por un factor s, y se realizan los típicos pasos de *softmax loss*.

Juntando todo lo anterior, la función de pérdida resultante de ArcFace se muestra en la ecuación (2.19), donde y_i corresponde a la clase correcta de la característica x_i, s a un escalar, m al margen de penalización y $cos(\theta_j)$ el logit de la clase j.

$$L_{ArcFace} = -\frac{1}{N}\sum_{i=1}^{N} log(\frac{e^{s(cos(\theta_{yi}+m))}}{e^{s(cos(\theta_{yi}+m))} + \sum_{j=1,j\neq y_i} e^{s(cos(\theta_j))}}) \tag{2.19}$$

Sub-center ArcFace

Sub-center ArcFace [3] es una extensión del método ArcFace en el aprendizaje métrico. Mientras que ArcFace introduce una penalización angular para mejorar la discriminación entre clases en la clasificación facial, Sub-center ArcFace lleva esto un paso más allá al considerar múltiples sub-centros para cada clase. La idea es capturar la diversidad intrínseca dentro de cada clase facial. Cada clase facial puede tener múltiples modos debido a variaciones en iluminación, pose, expresión, y otros factores. En lugar de tener un único centro para cada clase, como en ArcFace, Sub-center ArcFace introduce múltiples sub-centros para cada clase para representar estas variaciones. Durante el entrenamiento, el modelo se entrena para que cada característica se acerque más a uno de estos sub-centros y se aleje de los sub-centros de otras clases.

Los pasos del algoritmo para su entrenamiento se visualizan en la figura 2.11 y se explican a continuación:

1. Se normaliza el vector de características $x_i \in \mathbb{R}^{dx1}$ resultante de la penúltima y los pesos de la última capa del clasificador $W \in \mathbb{R}^{dxkxn}$, de tal forma que $\|x_i\| = 1$ y $\|W\| = 1$,

donde d es la dimensión del vector, k la cantidad de sub-centros y n la cantidad de clases. En 2.11 se considera $d = 512$.

2. Se calcula un *subclass-wise logit* como $cos(\theta)_{subclass} = x_i \otimes W$, donde $cos(\theta) \in \mathbb{R}^{kxn}$. Posteriormente se aplica *max pooling* en la dimensión de k, obteniendo el *class-wise logit* $cos(\theta)_{class} \in \mathbb{R}^{1xn}$.

3. Se aplica arc al *class-wise logit*, se le suma una penalización m y se aplica coseno para volver a obtener el *logit*, el *logit* resultante queda $cos(\theta_j + m)$.

4. Se re-escala por un factor s, y se realizan los típicos pasos de *softmax loss*.

Juntando todo lo anterior, la función de pérdida resultante de Sub-center ArcFace se muestra en la ecuación (2.20), donde $\theta_{i,j} = arccos(max_k(W_{j_k}^T x_i))$, $k \in \{1, ..., K\}$.

$$L_{Sub-centerArcFace} = -\frac{1}{N}\sum_{i=1}^{N} log(\frac{e^{s(cos(\theta_{i,yi}+m))}}{e^{s(cos(\theta_{i,yi}+m))} + \sum_{j=1,j\neq y_i} e^{s(cos(\theta_j))}}) \tag{2.20}$$

Esta variación posee las siguientes ventajas con respecto a su versión original

- **Mejor representación**: Al tener múltiples sub-centros, se captura mejor la variabilidad dentro de cada clase, lo que lleva a representaciones más ricas y discriminativas.
- **Mayor robustez**: La introducción de sub-centros puede hacer que el modelo sea más robusto a las variaciones dentro de la clase, como cambios en la iluminación o la pose.
- **Mejora en la discriminación**: Al incentivar a las características a acercarse a uno de los sub-centros y alejarse de los sub-centros de otras clases, se mejora la capacidad de discriminación del modelo.

2.7. Segmentación de instancias de objetos

La segmentación de instancias de objetos es otra de las tareas fundamentales en el campo de la visión por computadora y el aprendizaje profundo. Esta tarea no busca únicamente identificar y categorizar objetos en una imagen, sino también delinear y distinguir instancias individuales de cada objeto. A diferencia de la segmentación semántica, que etiqueta cada píxel según su categoría o clase general, por ejemplo *gato*, la segmentación de instancias se esfuerza por separar y etiquetar objetos individuales, por ejemplo, *gato 1*, *gato 2*. Esta tarea es clave en aplicaciones como vehículos autónomos, medicina, agricultura y robótica. El modelo que más ha destacado en la segmentación de instancias de objetos, ha sido Mask R-CNN [17], el cual se basa en Faster R-CNN pero agregando un cabezal (en inglés *head*) de segmentación post detección. Un modelo reciente que ha revolucionado esta tarea, es el desarrollado la empresa *Meta*, llamado SAM por sus siglas en inglés *Segment Anything Model* [24], el cual se explicará a continuación.

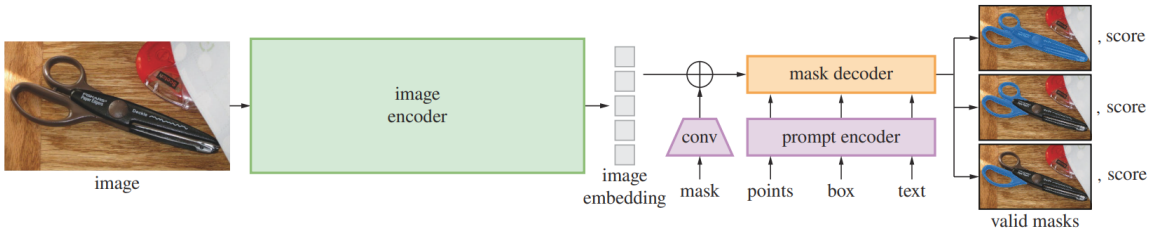


Figura 2.12: La arquitectura de SAM cuando se le proporciona una imagen como entrada. Imagen obtenida de [24].

2.7.1. Segment anything model

Segment Anything Model (SAM) [24], desarrollado por la empresa Meta, representa un avance revolucionario en el campo de la segmentación de imágenes mediante la integración de técnicas de aprendizaje profundo y arquitecturas basadas en transformadores (en inglés *transformers*). A diferencia de los enfoques tradicionales que dependen de extensos conjuntos de datos especializados y horas de entrenamiento, SAM ofrece una solución generalizable y direccionable para la segmentación.

La arquitectura del modelo SAM se muestra en la figura 2.12 y se compone de los siguientes elementos:

- **Image encoder**: Un codificador de imágenes que procesa la imagen de entrada.
- **Prompt encoder**: Un codificador de instrucciones, diseñado para comprender y traducir las instrucciones del usuario, que pueden ser texto, puntos, cuadros delimitadores o máscaras aproximadas dibujadas sobre un objeto.
- **Mask decoder**: Un decodificador de máscaras que, utilizando la potencia del aprendizaje profundo y los transformadores, genera la segmentación deseada con base en la combinación de los *embeddings* de la imagen de entrada y las instrucciones.

SAM tiene la capacidad de interpretar instrucciones específicas del usuario, ya sean textuales, puntos, cuadros delimitadores o máscaras aproximadas sobre el objeto de interés en la imagen. Con estas indicaciones, SAM entrega una segmentación precisa del objeto señalado. Por ejemplo, para segmentar un gato presente en una imagen, el usuario puede señalarlo con un punto, encerrarlo en un cuadro, dibujar una máscara aproximada o simplemente describirlo textualmente. SAM usará esta información junto con la imagen original para enfocarse y segmentar al gato deseado. Esto se ilustra en la figura 2.13, donde se busca segmentar un gato con orejas negras.

Además, para una segmentación completa de todos los objetos presentes en la imagen, el usuario puede proveer al modelo una grilla de puntos de muestreo por defecto, permitiendo que cada objeto en la imagen sea representado mediante uno de estos puntos.

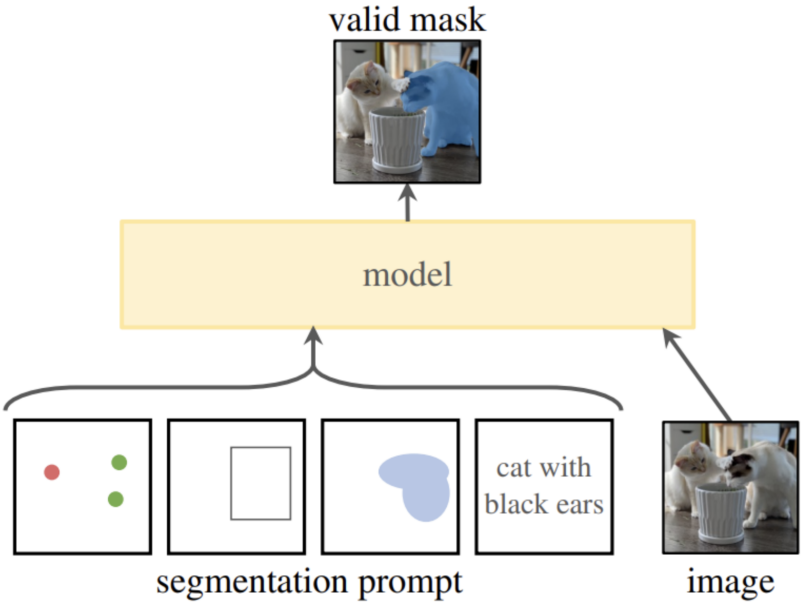


Figura 2.13: Representación de uso del modelo SAM mediante instrucciones específicas. Imagen obtenida de [7].

Capítulo 3

Detector integrado de instancias de objetos

Tal como se mencionó en la introducción, el propósito de esta investigación es desarrollar e implementar un detector integrado de instancias de objetos que funcione en tiempo real y que elimine la necesidad de re-entrenamientos extensivos para reconocer nuevas instancias de objetos.

Para que el detector alcance este objetivo, se integra en un sistema compuesto por tres bloques principales: (i) la red de detección integrada de instancias de objetos, que localiza y genera descriptores para cada objeto; (ii) un reconocedor de instancias que, mediante un esquema de clasificación de conjunto abierto (*Open Set Classification* en inglés), compara cada descriptor obtenido en (i) con los descriptores de las instancias de referencia, siendo capaz de identificar objetos desconocidos; (iii) un bloque que elimina detecciones redundantes, descartando aquellas que se repiten para una instancia específica de objeto. La figura 3.1 ilustra los tres módulos que conforman el sistema.

En las siguientes secciones, se detallará cada uno de estos tres bloques. No obstante, dado que el principal enfoque de esta tesis es la red de detección de instancias de objetos integrada, se le dedicará una explicación más exhaustiva.

3.1. Red de detección integrada de instancias de objetos

Como se mencionó, este bloque se compone de una red integrada de detección de instancias de objetos. Su objetivo principal es proporcionar las ubicaciones de todos los objetos en la imagen de entrada y, adicionalmente, retornar descriptores discriminativos para cada uno de ellos.

Para alcanzar este propósito, en primer lugar se requiere una red convolucional de detección capaz de generar propuestas de ubicación para todos los objetos en la imagen. Esta red

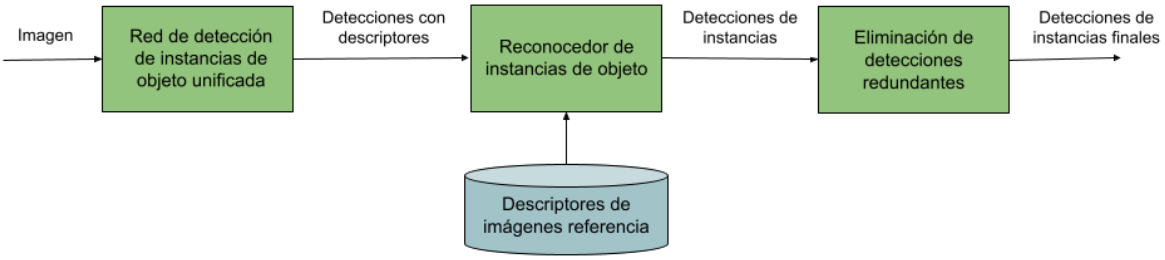


Figura 3.1: Diagrama de bloques del sistema para la detección de instancia de objetos.

no se debe limitar a detectar solo ciertas clases de objetos, sino que debe tener la capacidad de identificar cualquier objeto presente en la imagen, es decir, funcionar como un detector de objetos genérico.

Una vez que el detector proporciona las propuestas de detección, se extraen descriptores para cada una de estas. La estrategia propuesta es la integración de una red convolucional en una etapa temprana del *backbone* del detector y añadir un proceso de extracción de regiones de interés. Este proceso facilita la obtención de características precisas de los objetos según las ubicaciones proporcionadas por el detector en los mapas de características, permitiendo así retornar los descriptores globales de cada objeto.

A continuación se describirá en mayor detalle la solución tomada para el desarrollo de la red detectora de objetos genéricos, seguida por el extractor de descriptores de objetos.

3.1.1. Detector de objetos genéricos

En el marco de esta investigación, se adopta la red *YOLOv7* como detector base, ya que es reconocido por ser uno de los más avanzados en el estado del arte actual. No obstante, para alinear este detector con las necesidades específicas del presente estudio, se ha realizado una modificación clave en su función de pérdida. Tal como se expone en la sección 2.5.3 del marco teórico, detectores como el *YOLOv7* se especializan en identificar un conjunto específico de clases de objetos, determinado por el conjunto de datos con el que se entrenan.

La innovación que se propone consiste en omitir el aprendizaje de clases, priorizando el aprendizaje del concepto de *objectness* y la localización. Es decir, el objetivo del entrenamiento se centra en mejorar la localización y la capacidad del detector para discernir si una propuesta de detección efectivamente representa un objeto o no, sin importar la clase de objeto a la cual pertenezca.

Dicha modificación se representa eliminando la función de pérdida de clases (L_{class}) definida en la ecuación (2.17) de la función de pérdida original de *YOLOv7* (ecuación (2.14)), quedando finalmente la función de pérdida definida en la ecuación (3.1), donde L_{bbox} es la función encargada de la localización de los cuadros limitantes y L_{obj} es la función encargada de la confianza si una predicción es objeto o no.

$$L_{\text{YOLOv7 modificado}} = L_{bbox} + L_{obj} \tag{3.1}$$

La función de pérdida de localización L_{bbox} corresponde a la misma usada en *YOLOv7*, la cual queda definida en la ecuación (3.2), como una función de $CIoU$, donde b es la caja delimitadora predicha y b_{gt} es la caja delimitadora real (ground truth). El término ρ denota la distancia euclidiana entre los centros de las dos cajas, mientras que c es la diagonal de la caja delimitadora más pequeña que contiene ambas cajas. El αv es un factor de corrección que penaliza las discrepancias en la proporción de las cajas.

$$L_{bbox} = 1 - IoU + \frac{\rho(b, b^{gt})}{c^2} + \alpha v \tag{3.2}$$

Con respecto a la función de pérdida de objeto L_{obj}, esta corresponde a la utilizada originalmente por *YOLOv7*, la cual se encarga del *objectness* y se define en la ecuación (3.3). La primera parte de la ecuación suma sobre todas las celdas y cajas delimitadoras que contienen un objeto real. I_{ij}^{obj} es una variable indicadora que vale 1 si hay un objeto en la caja j de la celda i, y 0 en caso contrario. $\hat{C}i$ representa la confianza predicha de la presencia de un objeto, mientras que C_i es la confianza real (1 o 0). La segunda parte suma sobre las celdas y cajas donde no hay un objeto. $\lambda noobj$ actúa como un factor de escala para penalizar las predicciones falsas. I_{ij}^{noobj} es similar a I_{ij}^{obj}, pero se aplica a los "no-objetos".

$$\begin{aligned} L_{obj} = &\sum_{i=0}^{S^2}\sum_{j=0}^{B} I_{ij}^{obj}[\hat{C}_i log(C_i) + (1-\hat{C}_i) log(1-C_i)] \\ -\lambda_{noobj} &\sum_{i=0}^{S^2}\sum_{j=0}^{B} I_{ij}^{noobj}[\hat{C}_i log(C_i) + (1-\hat{C}_i) log(1-C_i)] \end{aligned} \tag{3.3}$$

Como resultado final de esta modificación, el detector que resulta de dicho proceso de entrenamiento, ya no retorna localizaciones con las clases de objetos, si no solo retorna las localizaciones de los objetos presenten en la imagen de entrada. A este nuevo detector se le denominará como *YOLOv7 modificado*.

3.1.2. Extractor de descriptores de objetos

El objetivo del módulo extractor de descriptores es generar representaciones distintivas de cada objeto detectado por *YOLOv7 modificado*. Para lograr esto, la base consiste en una red

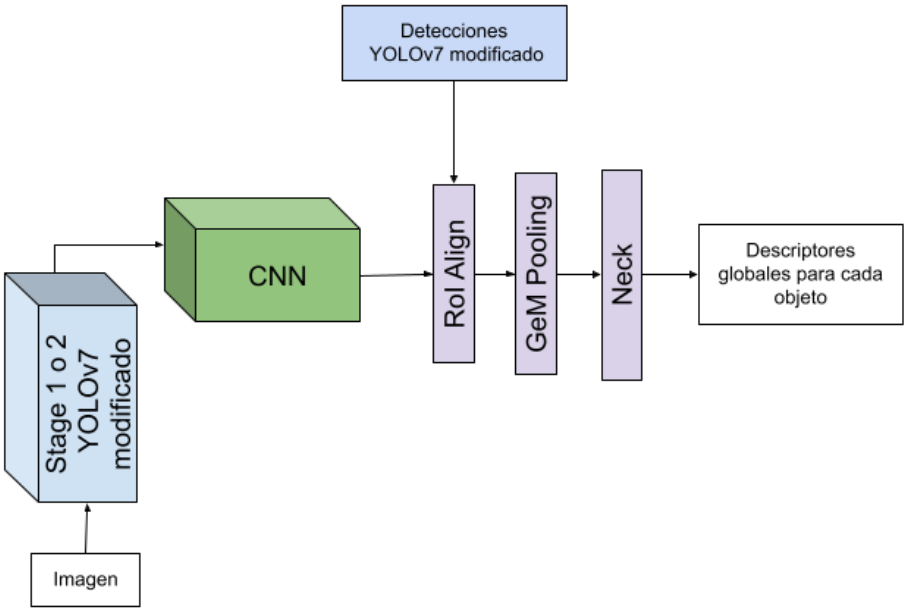


Figura 3.2: Estructura del módulo de extracción de descriptores globales para cada detección de objetos.

neuronal convolucional que se conecta a una de las etapas del *backbone* del detector de objetos genérico, uniendo así la *CNN* con la arquitectura del detector. Gracias a esta integración, se obtiene un mapa de características (o *feature map* en inglés) especializado que abarca toda la escena de entrada.

Posteriormente, a esta *CNN* se le incorpora una etapa de *RoI Align*. Esta etapa es crucial para extraer las regiones de interés del *feature map* generado por la *CNN*. Las regiones de interés que se extraen son determinadas por las detecciones del propio detector genérico. Una vez obtenido el mapa de características de cada región, se implementa una operación de *GeM Pooling* seguida de una capa *Neck* completamente conectada. El producto de estas dos operaciones resulta en la generación de descriptores globales para cada objeto detectado. La estructura completa de este módulo se ilustra en la figura 3.2.

Para asegurar que estos descriptores globales sean verdaderamente representativos, se entrena el módulo como un problema de clasificación. Aquí es donde entra en juego la función de pérdida de *Sub-center ArcFace*, ya que permite modelar la red como una función generadora de *embeddings*. Este enfoque permite, a su vez, el mapeo de objetos a un espacio métrico definido ideal para su comparación.

A continuación, se detallará cada uno de los componentes que integran el módulo de extracción de descriptores.

CNN

La estructura que se une al detector genérico es una red convolucional compuesta por múltiples capas. El objetivo principal de esta *CNN* es generar un mapa de características tridimensional $H_{fm} \times W_{fm} \times C_{fm}$ adaptado a la generación de descriptores de objetos. Aquí, H_{fm} y W_{fm} corresponden a la altura y el ancho, respectivamente, mientras que C_{fm} representa la profundidad o canales del mapa de características.

Este nuevo módulo del detector se ubica en la salida de una *stage* temprana del *backbone* de la red de detección. La razón de esta ubicación es que, en etapas tempranas, la red aún conserva características e información de alta resolución que no están excesivamente especializadas. Si se decidiera situar esta extensión en una etapa posterior de la red, la información que la *CNN* extraería tendría una resolución menor y estaría más enfocada a la detección de objetos, lo cual no es lo deseado para este módulo.

Las arquitecturas que se estudiarán corresponden a las siguientes:

- **JPNet**: La arquitectura de esta red propuesta por el autor, consiste en 5 capas, cada una compuesta por una convolución 2D con un *kernel* de 3x3 y un *stride* de 1. Cada capa convolucional está seguida de una capa de *Batch Normalization* 2D y una función de activación *SiLU*. A lo largo de la CNN, las dimensiones espaciales de altura y anchura del *feature map* de entrada se mantienen constantes, resultando en un *feature map* de salida con una profundidad de 512, independientemente del *stage* del *backbone*.
- **Módulos de VoVNet**: En el caso del modelo que utiliza la arquitectura VoVNet [26], se seleccionó VoVNet27 Slim, empleando únicamente los módulos de los *stages* 2 y 3 de esta red. Debido a las características de las operaciones en estos *stages* de VoVNet, las dimensiones de altura (H) y anchura (W) del *feature map* de salida se reducen a la mitad en comparación con las dimensiones de entrada. El *feature map* resultante mantiene una profundidad constante de 512, independientemente del *stage* específico del *backbone* de YOLOv7 modificado utilizado.
- **Baseline**: Para este enfoque, se emplea el método de Identidad. Esto significa que las dimensiones del *feature map* de salida son exactamente las mismas que las del *feature map* de entrada, extraídas del *stage 1* o *stage 2* del *backbone* de YOLOv7 modificado. De esta manera, se mantiene la estructura original del *feature map* sin aplicar cambios adicionales.

En el próximo capítulo, centrado en experimentos y validación, se examinará la relevancia de la CNN elegida. Para ello, se evaluarán las arquitecturas JPNet, VoVNet y *baseline*. Además, se procederá al entrenamiento y análisis de la integración de estas CNN al término del *stage 1* y *stage 2*.

RoI Align

Después de obtener el mapa de características resultante de la CNN y las propuestas de detección del detector, es necesario extraer las regiones de interés propuestas de este mapa.

Estas regiones se corresponden con las áreas de la imagen donde el detector ha identificado posibles objetos. La operación encargada de esta extracción es conocida como *RoI* (por sus siglas en inglés *Region of Interest*) *Align* [17].

Esta técnica busca superar las limitaciones de los métodos anteriores como *RoI Pooling*, que a menudo distorsionan las características de las regiones de interés debido a su enfoque de agrupamiento de tamaño fijo. Por otro lado, *RoI Align* evita cualquier cuantización del espacio de la región propuesta, preservando así la información espacial exacta. La idea es tomar secciones flotantes del *feature map* y aplicar interpolación bilineal para obtener valores precisos en ubicaciones no enteras, lo que da como resultado características más precisas para cada objeto propuesto.

El resultado de la operación *RoI Align* es un tensor de dimensiones $N \times H_{RoI} \times W_{RoI} \times C_{RoI}$, donde N es el número de propuestas de objetos por el detector genérico, H_{RoI} y W_{RoI} se refieren respectivamente a la altura y ancho del *feature map* de cada objeto detectado, mientras que C_{RoI} representa la profundidad, que se corresponde con la cantidad de canales en el mapa.

Este proceso garantiza que, independientemente del tamaño y la forma de las propuestas de detección, se obtenga un tensor de características de dimensiones consistentes para cada objeto, preparado para el posterior procesamiento y análisis.

GeM Pooling

Una vez que se han extraído las características de cada objeto propuesto mediante *RoI Align*, es necesario condensar esta información en un descriptor global que sea representativo para el análisis y comparaciones posteriores. Para esto se ocupa la operación *GeM Pooling*, que corresponde a técnica de agrupación que supera las limitaciones de los enfoques tradicionales de *pooling*.

El *GeM Pooling* [41] se diferencia de otros métodos de *pooling* en su capacidad para capturar información más representativa y diversa del mapa de características, utilizando un promedio ponderado potenciado de las características. Matemáticamente, para un mapa de características $X \in \mathbb{R}^{H_{RoI} \times W_{RoI} \times C_{RoI}}$ la operación *GeM Pooling* se define en la ecuación (3.4), donde $Y(c)$ es el resultado del *pooling* para el canal $c \in C$ y p_k es un parámetro que determina el grado de ponderación aplicado a las características. Cabe destacar que un $p_k = 1$ corresponde a la operación *average pooling*.

$$Y(c) = \left(\frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} X(i,j,c)^{p_k}\right)^{\frac{1}{p_k}} \tag{3.4}$$

El resultado final del bloque *GeM Pooling* es un tensor $Y \in \mathbb{R}^{NxC}$, donde N denota el número de objetos detectados y C representa la profundidad del vector descriptor, que es el número de canales o características.

En este trabajo, es crucial señalar que el parámetro p_k se mantiene fijo y no es objeto

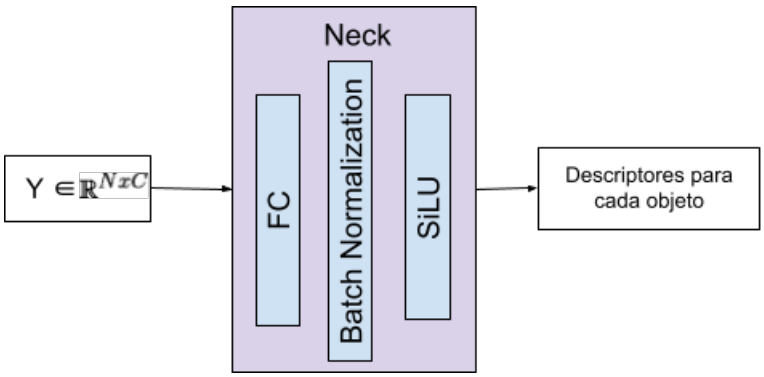


Figura 3.3: Diagrama de la estructura del *Neck*.

de entrenamiento. Esta decisión garantiza una consistencia en la transformación aplicada a los mapas de características, permitiendo comparaciones más directas entre los descriptores resultantes de diferentes objetos.

Neck

Siguiendo el trabajo realizado en [16], después de la operación de *GeM Pooling*, se incorpora una etapa adicional de procesamiento para afinar y optimizar el descriptor resultante. Esta etapa se denomina *Neck*, ya que sirve como un puente entre la extracción de características y la producción de descriptores finales.

El vector obtenido del *GeM Pooling* se pasa a través de una capa totalmente conectada (FC). Esta capa es necesaria para transformar y adaptar las dimensiones del vector, al mismo tiempo que permite capturar interacciones complejas entre las características. A continuación, una capa de normalización por lotes (BN) se aplica para garantizar que las características estén en una escala uniforme y para ayudar en la estabilización y aceleración del proceso de entrenamiento. Finalmente, se utiliza la función de activación SiLU (*Sigmoid Linear Unit*) para introducir no linealidades en el descriptor, permitiendo que el modelo capture patrones más complejos y sutiles en los datos. En la figura 3.3 se visualiza este módulo de forma completa.

Función de pérdida *Sub-Center ArcFace* para generación de descriptores

Para el entrenamiento del extractor de descriptores, es necesario representar este módulo como una función, que dada una imagen de entrada con instancias de objetos, esta retorne *embeddings* representativos por cada objeto. Tomando como referencia el trabajo realizado en [16], donde se trata el problema de recuperación de imágenes (conocido en inglés como *image retrieval*) en escenas de lugares, se propone la utilización de *Sub-center ArcFace* (explicada en la sección 2.6.1) para el entrenamiento del modelo final que genera los descriptores de estas escenas. Este enfoque obtuvo el 3° lugar en la competencia *Google Landmark Recognition 2020*. Debido a esto, se considera el uso de un cabezal *Sub-center ArcFace* para entrenar el módulo de generador de descriptores.

La función de pérdida *Sub-center ArcFace* ocupada para el entrenamiento del módulo generador de descriptores se muestra en la ecuación (2.20), donde $\theta_{i,j} = arccos(max_k(W_{j_k}^T x_i))$, $k \in \{1, ..., K\}$ y m es el margen fijo.

Una vez finalizado el proceso de entrenamiento, se retira el cabezal de entrenamiento *Sub-center ArcFace* del módulo extractor de descriptores, así este se encuentra habilitado para hacer inferencias de *embeddings* por cada imagen.

3.2. Reconocedor de instancias de objetos

Tras la entrega de las detecciones y sus correspondientes descriptores por parte de la red integrada de instancias de objetos, es necesario que estos sean procesados por un clasificador. Dicho clasificador debe ser capaz de asignar etiquetas comparando los descriptores de los objetos detectados con descriptores de referencia previamente calculados. Además, este clasificador deberá tener la capacidad de reconocer objetos no previstos, identificando aquellos cuyos descriptores de prueba no coincidan con ningún descriptor de referencia, es decir, debe corresponder a un clasificador de conjunto abierto. Los descriptores de referencia son representaciones calculadas de aquellos objetos que el sistema está configurado para identificar según los requerimientos del usuario.

Para cumplir con estos objetivos, el clasificador diseñado para el reconocimiento de instancias sigue el siguiente procedimiento:

1. **Comparación de descriptores**: Dado un descriptor de prueba que necesita ser clasificado, se calcula la distancia euclidiana entre el descriptor de prueba y todos los descriptores de referencia.
2. **Identificación de descriptores**: Con la comparación de descriptores realizada, se identifica la distancia al descriptor de referencia más cercano d_1. Siguiendo la condición 3.5, si la distancia d_1 es menor que un umbral de distancia predefinido λ, se asume que el descriptor es conocido y se le asigna la etiqueta del descriptor de referencia más cercano.

   $$d_1 < \lambda \rightarrow \text{conocido} \tag{3.5}$$

   En el caso contrario, si d_1 es mayor o igual al umbral de distancia λ se considera el descriptor como desconocido.

   $$d_1 \geq \lambda \rightarrow \text{desconocido} \tag{3.6}$$

El resultado de este procedimiento es la identificación de cada uno de los objetos como una de las instancias de clases de referencia o como una instancia desconocida. Adicionalmente, a cada objeto se le asigna la distancia del descriptor de referencia más cercano d_1, para ser utilizada en el siguiente bloque de eliminación de detecciones redundantes.

3.3. Eliminación de detecciones redundantes

Con las identificaciones de cada objeto detectado obtenidas, es necesario diseñar un proceso de limpieza para eliminar detecciones redundantes. Este proceso es requerido debido a que el sistema puede generar múltiples propuestas de detección para un único objeto. Adicionalmente, debido a que el sistema emplea un detector de objetos genéricos, pueden generarse falsos positivos de detección, lo que termina resultando en múltiples identificaciones desconocidas por el clasificador.

El proceso de limpieza diseñado consiste en filtros aplicados en forma de cascada, con el objetivo de que a medida que las detecciones vayan siendo procesadas en cada etapa, se eliminen cada vez más detecciones redundantes. Los filtros diseñados e implementados se presentan a continuación:

1. **Filtro por clase:** Este filtro agrupa las detecciones obtenidas por clase y compara el IoU de cada una con el resto de detecciones de la misma clase asignada. Si en la comparación se encuentra un IoU que supera un umbral establecido, se descarta la detección que posea la mayor distancia d_1 a su etiqueta de referencia respectiva.
2. **Filtro por IoU**: Al igual que en el caso anterior, se hace un filtrado comparando los IoU de las detecciones, sin embargo, en este caso no se tiene en consideración las clases. Si existe el caso donde hay detecciones con un IoU mayor a un umbral, se elimina la detección que tenga la mayor distancia d_1 a su etiqueta respectiva. Es importante, mencionar que para evitar filtrado de detecciones correctas debido a un agrupamiento real de objetos, es necesario definir un umbral mayor a 0,5.
3. **Filtro de objeto desconocido que contiene a objeto conocido**: Se descartan las detecciones de objetos clasificados como desconocidos si contienen completamente a un objeto conocido dentro de su cuadro delimitador. Como este filtro únicamente elimina detecciones reconocidas como desconocidas, no tiene influencia en las métricas de precisión.
4. **Filtro de objeto desconocido contenido en objeto conocido**: Inversamente al filtro anterior, se eliminan las detecciones de objetos desconocidos que estén completamente contenidos dentro del cuadro delimitador de un objeto conocido. Como este filtro únicamente elimina detecciones reconocidas como desconocidas, no tiene influencia en las métricas de precisión.

3.4. Arquitectura final del sistema

Finalmente, la arquitectura del sistema completo de detección de instancias de objeto se ilustra en la Figura 3.4. Esta arquitectura detalla el flujo operativo completo, partiendo de una imagen inicial, pasando por el detector *YOLOv7 modificado* que genera propuestas de detección. Estas propuestas son procesadas por el módulo extractor que obtiene sus descriptores correspondientes. A continuación, se realiza la etapa de reconocimiento de instancias y, finalmente, el sistema termina con el módulo de eliminación de detecciones redundantes.

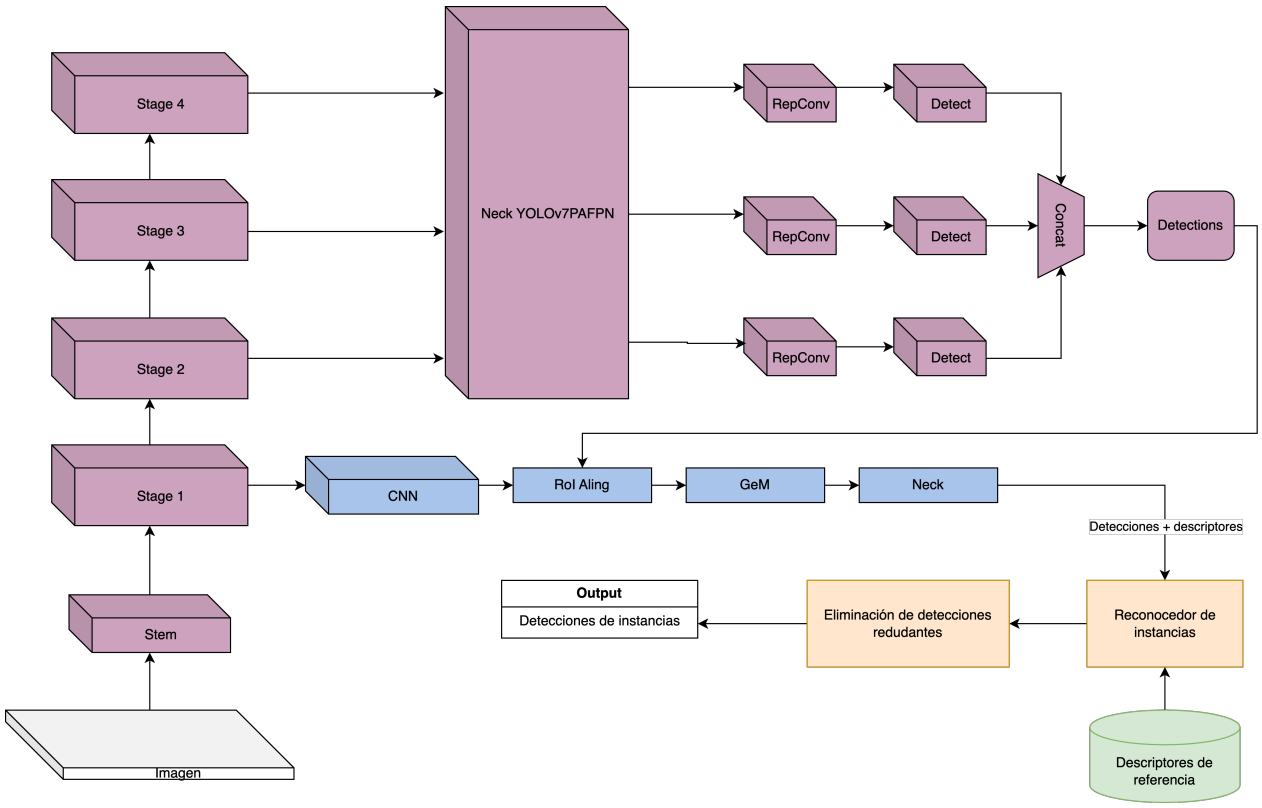


Figura 3.4: Arquitectura final del sistema detección de instancias de objetos. En color rosado se representa los bloques de la red *YOLOv7 modificado* y en color azul los bloques de la red extractora de descriptores.

Capítulo 4

Metodología de evaluación

En este capítulo se exponen en detalle los entrenamientos y experimentos efectuados para validar la hipótesis presentada al comienzo de este trabajo. Se estructura en cinco secciones principales, cada una dedicada a una fase específica del proceso de validación. La sección 4.1 se enfoca al entrenamiento del detector *YOLOv7 modificado*, mientras que la sección 4.2 se dedica a la evaluación de estos entrenamientos. Posteriormente, la sección 4.3 detalla el entrenamiento de la red extractora de descriptores, y la sección 4.4 se centra en su respectiva evaluación. El capítulo concluye con la sección 4.5, donde se lleva a cabo la validación final que mide el rendimiento del sistema completo.

4.1. Entrenamiento del detector YOLOv7 modificado

Esta sección detalla los procedimientos seguidos en el entrenamiento de la red YOLOv7 modificada propuesta. Inicialmente, se describirán las bases de datos utilizadas para el entrenamiento de la red, detallando sus características y la relevancia para el estudio. Posteriormente, se expondrán las técnicas de *data augmentation* implementadas con el objetivo de mejorar el rendimiento del modelo. Para concluir, se presentará la configuración específica adoptada en el entrenamiento, incluyendo una explicación de los hiperparámetros seleccionados.

4.1.1. Base de datos para detección

Que la red pueda lograr detectar objetos genéricos sin importar la clase, depende en gran medida del conjunto de datos utilizado durante el entrenamiento. Con este fin, se opta por iniciar el entrenamiento con el conjunto de datos de detección MS COCO [29], pero ignorando la etiqueta de clase de cada objeto. Como segundo entrenamiento, debido a que MS COCO únicamente posee etiquetas de 80 clases de objetos a pesar de que en sus imágenes existan objetos de otras clases, se propone un nuevo etiquetado completo de detección de las imágenes mediante un modelo del estado del arte de segmentación de objetos.

Tabla 4.1: Distribución de conjuntos de datos en COCO 2017.

Conjunto	N° de imágenes	% de imágenes
Entrenamiento	118K	82,5 %
Validación	5K	3,5 %
Prueba	20K	14 %

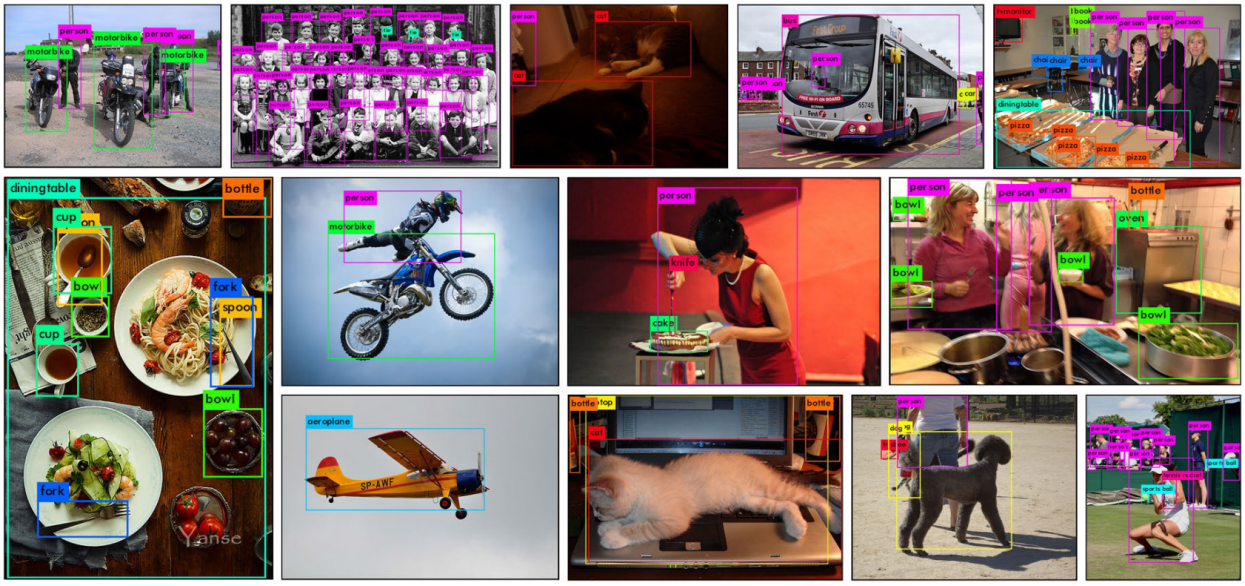


Figura 4.1: Ejemplo de imágenes de COCO. Imagen obtenida de [35].

COCO

Una de las bases de datos más ampliamente utilizadas en la tarea de detección de objetos es la base de datos COCO (por sus siglas en inglés, Common Objects in Context) [29]. Este conjunto de datos fue creado por Microsoft y se caracteriza por proporcionar etiquetas para la detección, segmentación y clasificación de objetos. COCO contiene más de 200,000 imágenes etiquetadas, abarcando un total de más de 1.5 millones de instancias de objetos distribuidas en 80 clases distintas. Estas categorías van desde personas y vehículos hasta una amplia variedad de objetos domésticos y elementos urbanos.

Las imágenes de COCO no solo están acompañadas de anotaciones de cajas delimitadoras, sino que también incluyen segmentaciones de instancias, lo que permite llevar a cabo entrenamientos para tareas relacionadas con la segmentación de objetos. COCO presenta una gran diversidad de imágenes, que incluye escenarios urbanos y domésticos, con objetos que presentan diferentes escalas, orientaciones y grados de obstrucción. Lo anterior convierte a COCO en un conjunto de datos sumamente desafiante y completo para el desarrollo de sistemas de visión artificial de alto rendimiento, en particular, sistemas de detección de objetos.

La distribución de los conjuntos de entrenamiento (*train set*), validación (*val set*) y prueba (*test set*) se presentan en la tabla 4.1. En la figura 4.1 se pueden observar ejemplos de imágenes etiquetadas del conjunto de datos de COCO.

(a) Ejemplo de la falta de etiquetado en objetos de la categoría *umbrella*, a pesar de estar incluida en las categorías de COCO.

(b) Ejemplo de la presencia de objetos en una imagen de COCO que no están incluidos en las 80 categorías predefinidas.

Figura 4.2: Ejemplos que muestran problemas de etiquetado en el conjunto de datos COCO para el entrenamiento del YOLOv7 modificado.

Si bien el conjunto de datos contiene 80 categorías de objetos comúnmente encontrados en entornos urbanos y domésticos, estas no son suficientes para cubrir la amplia variedad de objetos que pueden aparecer en dichos escenarios. Una solución habitual para extender la capacidad de detección de clases de objeto es entrenar con las categorías originales del conjunto de datos y, posteriormente, reducir el umbral de confianza durante la inferencia. Este ajuste permite que el detector intente reconocer objetos con formas similares a las definidas por las categorías preexistentes, ampliando así la gamma de objetos que el modelo puede identificar.

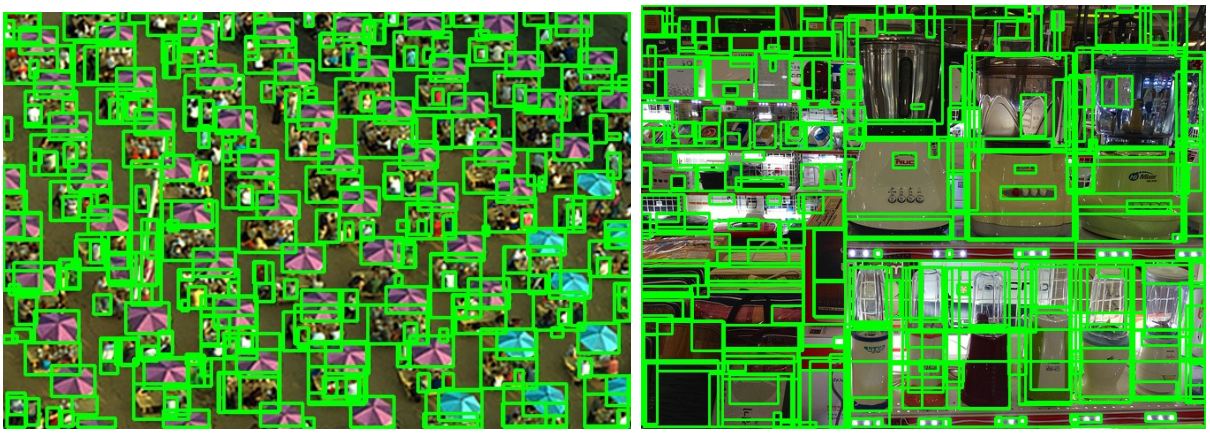
COCO etiquetado con SAM

A pesar de que COCO es uno de los conjuntos de datos de detección más utilizados, no está libre de errores en su etiquetado. Se han identificado imágenes que contienen objetos no etiquetados, a pesar de pertenecer a las 80 categorías predefinidas. Además, en las imágenes de COCO no solo aparecen objetos de estas 80 clases, si no que también hay elementos que no pertenecen a ellas. Si un detector es entrenado exclusivamente con este conjunto de datos, estos objetos podrían ser erróneamente clasificados como fondo o *background*, lo cual es contrario al objetivo que se busca con un detector de objetos genéricos. La figura 4.2 ilustra estos dos problemas: la figura 4.2a muestra la falta de etiquetas en un objeto que está dentro de las categorías definidas, mientras que la figura 4.2b evidencia la presencia de objetos en el conjunto de datos que no forman parte de las clases establecidas.

Como solución a este problema, se propone la utilización de las imágenes de COCO, pero re-etiquetadas con un enfoque que reconozca todos los objetos presentes en las imágenes, independientemente de su clase. Un modelo para este propósito es la red de segmentación de objetos SAM [24]. Este modelo no solo realiza la segmentación de todos los objetos en una imagen, sino que también permite enfocarse en aquellos específicos indicados por el usuario. Además, SAM proporciona recuadros de detección para cada objeto identificado, facilitando

Tabla 4.2: Parámetros seleccionados para el modelo *ViT-H SAM*.

Parámetro	Definición	Valor
`points_per_side`	Número de puntos muestreados por lado en la imagen	32
`pred_iou_thresh`	Umbral para la calidad de la máscara predicha	0,88
`box_nms_thresh`	Umbral de IoU para cajas utilizado en el algoritmo NMS	0,7



(a) Ejemplo 1 de resultado de etiquetado. (b) Ejemplo 2 de resultado de etiquetado.

Figura 4.3: Resultados del etiquetado de imágenes COCO utilizando SAM para el entrenamiento del detector de objetos genéricos. Se destacan las detecciones adicionales que no están presentes en las etiquetas originales de COCO.

así un etiquetado más completo y acorde a los requisitos de un detector de objetos genéricos.

Para el etiquetado, se seleccionó la variante más avanzada del modelo SAM, específicamente el *ViT-H SAM*. Utilizando esta versión, se procedió a inferir cada imagen de los tres subconjuntos del dataset COCO (entrenamiento, validación y prueba), generando así los recuadros de detección para cada una de ellas. Los parámetros específicos empleados en el proceso de etiquetado con el modelo SAM se detallan en la tabla 4.2.

Finalmente, cada recuadro delimitador obtenido se convierte al formato YOLO, un proceso que se detalla en la sección 4.1.2. Estos recuadros se almacenan en archivos de texto (`.txt`) con el mismo nombre de sus imágenes de origen. La figura 4.3 muestra ejemplos de los resultados obtenidos con el etiquetado realizado por SAM. En estos ejemplos, se aprecia un aumento significativo en el número de detecciones en comparación con las etiquetas originales de COCO, lo cual significa una mayor captura de objetos dentro de las imágenes. Este incremento en las detecciones es crucial para el entrenamiento del detector de objetos genéricos.

4.1.2. Formato YOLO

Para el entrenamiento del detector YOLOv7 modificado, es esencial que los conjuntos de datos utilizados, tanto el COCO original como el COCO etiquetado con SAM, se conviertan al formato YOLO [42]. En este formato, cada objeto detectado en la imagen se representa

mediante un conjunto de coordenadas normalizadas que definen el recuadro delimitador, acompañadas de la clase del objeto.

Distribución de imágenes y etiquetas

En el formato YOLO, las imágenes y sus correspondientes etiquetas se almacenan en directorios separados. Cada imagen del conjunto de datos tiene un archivo de etiqueta asociado, que lleva el mismo nombre, pero con una extensión distinta (`.txt` para las etiquetas, en comparación con `.jpg` o `.png` para las imágenes). Por ejemplo, si existe una imagen denominada `image1.jpg`, su archivo de etiqueta correspondiente será `image1.txt`.

Para ilustrar, la estructura de distribución de imágenes y etiquetas para el dataset COCO en preparación para el entrenamiento se organiza de la siguiente manera:

```
instance-object-detector
├── data
│   └── coco.yaml
└── coco
    ├── images
    │   ├── train
    │   │   ├── image_train_1.jpg
    │   │   └── image_train_2.jpg
    │   └── valid
    │       ├── image_val_1.jpg
    │       └── image_val_2.jpg
    ├── labels
    │   ├── train
    │   │   ├── image_train_1.jpg
    │   │   └── image_train_2.jpg
    │   └── valid
    │       ├── image_val_1.jpg
    │       └── image_val_2.jpg
    ├── train.txt
    └── val.txt
```

En esta distribución, es necesario definir el archivo `coco.yaml` que contiene la información global del conjunto de datos. Este archivo incluye las rutas a los archivos `train.txt` y `val.txt`, así como el número total de clases presentes en el conjunto de datos y una lista con los nombres de cada una de estas clases. Los archivos `train.txt` y `val.txt` contienen las ubicaciones de todas las imágenes correspondientes a los conjuntos de entrenamiento y validación, respectivamente.

Formato de etiquetas

Cada archivo de etiqueta en el conjunto de datos YOLO contiene una o más líneas, con cada línea representando un objeto detectado en la imagen. El formato para estas líneas se estructura de la siguiente manera:

```
<class> <x_center> <y_center> <width> <height>
```

Los componentes de este formato son:

- `<class>`: un índice entero que identifica la clase a la que pertenece el objeto.
- `<x_center>` y `<y_center>` son las coordenadas del centro del objeto en relación con las dimensiones de la imagen, normalizadas entre 0 y 1.
- `<width>` y `<height>` son el ancho y la altura del objeto también normalizados en relación con las dimensiones de la imagen.

Un ejemplo visual de cómo se representa el formato YOLO en una imagen se muestra en la figura 4.4. En esta figura, se destaca que el punto de origen para las coordenadas se sitúa en la esquina superior izquierda de la imagen. El archivo de etiqueta en formato `.txt` asociado se presenta en la Figura 4.5, donde se detallan las coordenadas normalizadas y la clase del objeto detectado en la imagen.

4.1.3. Data augmentation

Para fortalecer la robustez y capacidad de generalización de la red YOLOv7 modificada, se han empleado varias técnicas de *data augmentation*. Estas técnicas están diseñadas para introducir variabilidad a las condiciones de detección dentro del conjunto de entrenamiento, imitando así las posibles variaciones que podrían encontrarse en escenarios reales. La selección de técnicas utilizadas están basadas en los trabajos de YOLOv4 [1] y YOLOv7 [63], sin embargo, los valores asignados a cada una son propios del autor.

A continuación se presentan las técnicas de *data augmentation* seleccionadas y utilizadas, junto con la descripción de sus parámetros:

Figura 4.4: Ejemplo visual del formato YOLO en imagen. Imagen obtenida de [47].

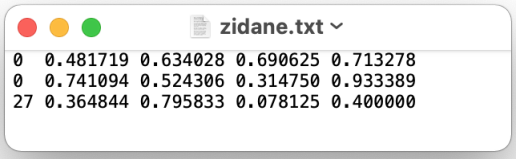


Figura 4.5: Ejemplo visual del formato YOLO en archivo `.txt`. Imagen obtenida de [47].

- **HSV-Hue**: Ajusta el matiz (*Hue*) de la imagen en una fracción de 0.015. Esta variación ligera en el color de los objetos incrementa la robustez del modelo ante cambios cromáticos.
- **HSV-Saturation**: Modifica la saturación de los colores en una fracción de 0.7, permitiendo que el modelo se adapte a variaciones en la intensidad cromática.
- **HSV-Value**: Altera el valor (*Brightness*) de la imagen en una fracción de 0.4, preparando al modelo para trabajar con imágenes de distintos niveles de luminosidad.
- **Traslación**: Implementa traslaciones aleatorias de hasta un 20 % de la dimensión de la imagen, tanto horizontal como verticalmente.
- **Escalado**: Cambia el tamaño de las imágenes aleatoriamente en un rango de ±9 %, lo que facilita la identificación de objetos en diversos tamaños.
- **Volteo Horizontal**: Aplica una inversión horizontal a la imagen con una probabilidad del 50 %, mejorando la capacidad del modelo para reconocer objetos en orientaciones variadas.
- **Mosaico**: Combina varias imágenes en una sola con una probabilidad del 100 %, ampliando el contexto visual para el entrenamiento del modelo [1].
- **Mixup**: Superpone imágenes con una probabilidad del 15 %, lo que ayuda al modelo a discernir objetos con traslapes parciales [1].
- **Paste-In**: Introduce objetos adicionales en las imágenes con una probabilidad del 15 %, similar a la técnica Copy-Paste [12], para incrementar la complejidad de las escenas y el desafío al modelo [12].

Las técnicas de *data augmentation* mencionadas han sido implementadas con el fin de simular las variaciones naturales y artificiales que el modelo podría encontrar en un entorno real de aplicación. Para la implementación de estas transformaciones, se utilizó *Pytorch*, que ofrece herramientas avanzadas para la manipulación y transformación eficiente de imágenes durante el proceso de entrenamiento.

4.1.4. Configuración de entrenamiento

Para el entrenamiento del modelo YOLOv7 modificado, tanto con el conjunto de datos COCO original como con COCO etiquetado con SAM, se mantuvo el mismo diseño de red y los mismos hiperparámetros. Los entrenamientos se llevaron a cabo utilizando una GPU Tesla V-100 de 32 GB en un servidor DGX. A continuación, se detallan los hiperparámetros utilizados:

- **Número de Épocas:** 100 épocas.
- **Optimizador:** Se utilizó el optimizador SGD.

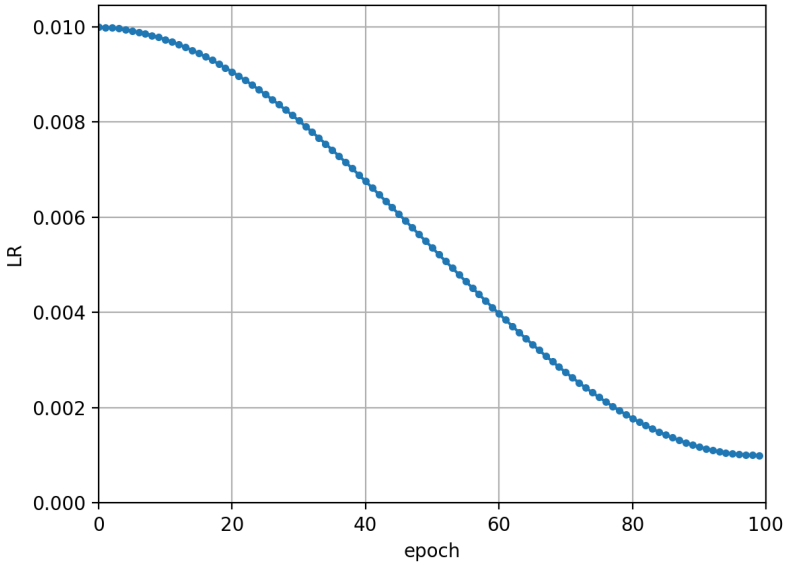


Figura 4.6: Decaimiento *OneCycleLR* [19] utilizado en el entrenamiento del YOLOv7 modificado.

- **Learning Rate:** Inicialmente se estableció en 0,01, reduciéndose según un esquema de decaimiento específico. Se aplicó el método *OneCycleLR* [19], donde la tasa disminuye en un factor de 0,1, finalizando en 0,001. La Figura 4.6 ilustra la evolución de la tasa de aprendizaje durante el entrenamiento.
- **Momentum:** Fijado en 0,937.
- **Batch Size:** Tamaño de lote de 32.
- **Peso de Regularización:** Regularización L2 con un peso de 0,0005.
- **Umbral de Entrenamiento de IoU:** Umbral de IoU establecido en 0,20.
- **Umbral de Anclaje:** Umbral de anclaje múltiple configurado en 4,0.

Es importante mencionar que, dado que la función de pérdida del YOLOv7 modificado omite el aprendizaje de clases, cualquier hiperparámetro relacionado con clases se excluye y el entrenamiento se establece en modo *single-class*.

Siguiendo la metodología del trabajo [63], el entrenamiento en ambos conjuntos de datos se realizó desde cero, sin necesidad de modelos pre-entrenados. Esto fue posible gracias a la eficiencia en la propagación de gradientes que ofrece YOLOv7 a través de sus módulos E-ELAN.

4.2. Evaluación YOLOv7 modificado

Una vez completado el entrenamiento de la red YOLOv7 modificado, resulta necesario evaluar su rendimiento en un conjunto de pruebas para determinar cuál de los dos enfoques adoptados resuelve de mejor manera el problema de detección de objetos genéricos, requerido para la identificación de instancias de objetos. Para ello, esta sección comienza con una revisión breve de los conceptos clave necesarios para comprender las métricas de evaluación que se utilizarán. A continuación, se describen y detallan las bases de datos empleadas en la evaluación. Finalmente, en la subsección dedicada a las métricas, se detallará las métricas finales de evaluación y se definirá el criterio para la selección del modelo definitivo.

4.2.1. Definición de términos

Para el análisis y cálculo de métricas en la detección de objetos, se definen los siguientes términos [57]:

- **Verdadero Positivo (TP):** Una detección donde el modelo identifica correctamente un objeto real presente en la imagen.
- **Falso Positivo (FP):** Una detección en la que el modelo señala incorrectamente un objeto que no existe.
- **Falso Negativo (FN):** Un objeto real presente en la imagen que el detector de objetos no logra identificar.
- **Verdadero Negativo (TN):** Corresponde a regiones de fondo que el modelo correctamente no identifica como objetos. Esta métrica generalmente no se utiliza en la detección de objetos, ya que las regiones de fondo no suelen estar anotadas de manera explícita en los conjuntos de datos.

Intersección sobre Unión (IoU)

Mide el grado de superposición entre el recuadro de verdad g_t y la predicción p_d. Su cálculo viene dado por la ecuación (2.8).

IoU [46] varía entre 0 y 1, donde 0 muestra que no hay superposición y 1 significa superposición perfecta. Para el *umbral IoU* en α, un Verdadero Positivo (TP) es una detección para la cual $\text{IoU}(g_t, p_d) \geq \alpha$ y un Falso Positivo (FP) es una detección para la cual $\text{IoU}(g_t, p_d) < \alpha$. Un Falso Negativo (FN) es un objeto real que se pierde junto con g_t para el cual $\text{IoU}(g_t, p_d) < \alpha$. Es importante mencionar, que la decisión de marcar una detección como TP o FP y un objeto real como FN depende completamente de la elección del *umbral IoU* α.

Precision

Precision [39] se refiere a la proporción de predicciones correctas de objetos (verdaderos positivos) en relación con todas las predicciones realizadas por el modelo, que incluyen tanto los verdaderos positivos como los falsos positivos. La fórmula para calcular el *precision* es la siguiente:

$$\text{Precisión} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.1}$$

Recall

Recall [39] mide la capacidad del modelo para identificar correctamente todos los objetos reales presentes en las imágenes. Se calcula como la proporción de verdaderos positivos respecto al total de objetos reales, que es la suma de los verdaderos positivos y los falsos negativos. La fórmula para calcular el *recall* es la siguiente:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.2}$$

El *precision* es clave para minimizar falsas alarmas en aplicaciones sensibles, mientras que el *recall* es importante para garantizar de que todos los objetos reales presentes en la imagen sean detectados por el modelo. Una métrica que combina tanto la *precision* y el *recall* es el *F1-score*.

F1-score

F1-score [39] es una métrica que proporciona un equilibrio entre la *precision* y el *recall*, siendo útil cuando las clases están desequilibradas. En detección de objetos, donde la exactitud en la localización es tan importante como la correcta clasificación, el *F1-score* se calcula de la siguiente manera:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.3}$$

Esta métrica alcanza su mejor valor en 1 (precision y recall perfectos) y el peor en 0. Es especialmente útil cuando se requiere un equilibrio entre *precision* y *recall*, y es una métrica relevante en la evaluación de modelos de detección de objetos.

4.2.2. Métrica de evaluación

Con las definiciones anteriores, se puede concluir que la *precision* es crucial para minimizar las falsas alarmas en contextos de aplicación sensibles, mientras que el *recall* es importante

para garantizar que todos los objetos reales sean detectados en las imágenes. En el contexto de la detección de objetos genéricos, el objetivo es que la red sea capaz de localizar la mayor cantidad de objetos posibles presentes en la imagen. Por lo tanto, el *recall* se convierte en una métrica más apropiada para evaluar este problema. Sin embargo, una detección excesiva de objetos puede ser contraproducente, ya que el aumento de falsos positivos podría inducir errores de reconocimiento en el sistema completo y aumentar el tiempo de procesamiento.

Por lo tanto, para poder evaluar eficazmente los modelos de detección entrenados, se requiere una métrica que considere ambas medidas, pero que otorgue una mayor importancia al *recall*. Es por esta razón que se ha decidido utilizar la versión extendida del *F1-score* [50], conocida como *Fβ-score*, que permite ajustar el balance entre *precision* y *recall*.

Fβ-score

El Fβ-score [50] es una extensión del F1-score que permite ajustar la importancia relativa entre la *precision* y el *recall*. En esta métrica, el parámetro β determina el peso del *recall* en la combinación de métricas. Un β mayor que 1 da más peso al *recall*, lo que es beneficioso en situaciones donde detectar todos los objetos reales es más importante que minimizar las detecciones incorrectas. La fórmula para calcular el Fβ-score es la siguiente:

$$\text{F}\beta\text{-score} = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} \tag{4.4}$$

Los valores de β seleccionados para evaluar el modelo son 2 y 3, correspondientes al F2-score y F3-score, respectivamente.

Adicionalmente, con el objetivo de analizar una métrica de comparación estándar adicional en la detección de clases de objetos, se empleará el *Mean Average Precision* adaptado a la detección de objetos genéricos, cuyos detalles se explican a continuación.

Mean Average Precision

El *Mean Average Precision* [9], también conocido como *mAP*, es una métrica comúnmente utilizada en la evaluación de modelos de detección de clases de objetos. Proporciona una medida del rendimiento del modelo en términos de *precision* y *recall*, considerando varios umbrales de detección. Para calcular el *mAP*, se siguen estos pasos:

1. Se calcula el *Average Precision*, también conocido como *AP*, para cada clase, que es el promedio de la *precision* lograda a diferentes niveles de *recall*.
2. Como muestra la ecuación (4.5), el mAP es el promedio del AP de todas las clases. Se obtiene sumando los AP de todas las clases y dividiendo por el número total de clases.

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \tag{4.5}$$

Donde N representa el número de clases y AP_i es el *Average Precision* para la clase i. En el contexto específico de la detección de objetos genéricos, donde no se consideran distintas clases, se asume que $N = 1$. Bajo esta condición, el valor de mAP es equivalente a AP, ya que solo se está evaluando un único conjunto general de objetos en lugar de clases específicas.

4.2.3. Base de datos

Tras definir las métricas de evaluación, es necesario seleccionar los conjuntos de datos adecuados para las pruebas de evaluación. Estos conjuntos deben cumplir con los siguientes requisitos para garantizar una evaluación efectiva:

- Deben contener etiquetas para todos los objetos presentes en cada imagen.
- Incluir objetos de categorías que no estén presentes en el conjunto de datos COCO.
- Contar con más de una instancia de objeto por cada clase.
- Ofrecer variación en el número de objetos por imagen.

Los conjuntos de datos que satisfacen estos requisitos son el subconjunto de validación de RPC y el conjunto de evaluación de DSLL. A continuación, se proporcionarán más detalles sobre estos *datasets*.

RPC

El conjunto de datos *Retail Product Checkout* (RPC) [65] es una colección de imágenes enfocada en la detección de productos en entornos de retail, como supermercados y tiendas de autoservicio. Este *dataset* se destaca por su diversidad en la gama de productos típicamente encontrados en el proceso de compra (*checkout*), y es frecuentemente utilizado para evaluar la eficacia de algoritmos de visión computacional en el reconocimiento y detección de objetos en estos escenarios. *Retail Product Checkout* contiene un total de 200 instancias de productos repartidos en 17 meta categorías, que incluyen *puffed food, dried fruit, dried food, instant drink, instant noodles, dessert, drink, alcohol, milk, canned food, chocolate, gum, candy, seasoner, personal hygiene, tissue,* y *stationery*. La figura 4.7 muestra ejemplos de tres productos de 15 de estas meta categorías presentes en el *dataset*. RPC se divide en tres subconjuntos: entrenamiento, validación y prueba. La distribución de los datos en el *dataset* RPC se presenta en la tabla 4.3.

El subconjunto de entrenamiento de RPC incluye 53793 imágenes de alta resolución que exhiben productos desde distintos ángulos y perspectivas, todas capturadas en un entorno controlado y fijo. Cada imagen en este subconjunto está anotada con información sobre el

Figura 4.7: Ejemplo de productos de contenidos en el dataset RPC. Para cada meta categoría, se visualizan 3 productos. Imagen obtenida de [65].

Tabla 4.3: Distribución de sub-conjuntos de datos en RPC.

Conjunto	N° de imágenes	N° de objetos	Categorías/imagen
Entrenamiento (Productos)	53793	53739	1
Validación (Checkout)	6000	73602	6,33
Prueba (Checkout)	24000	294333	6,31

tipo de producto, la meta categoría a la que pertenece, y la localización del objeto dentro de la imagen. Un aspecto destacable de este subconjunto es que cada imagen se centra en la vista de un solo objeto, es decir, hay un producto único por imagen. Las condiciones en las que se capturaron estas imágenes y la variedad de perspectivas que se ofrecen de cada objeto están ilustradas en la figura 4.8.

A diferencia del subconjunto de entrenamiento, los subconjuntos de validación y prueba de RPC no se componen de imágenes que muestran diferentes vistas de un solo producto. En su lugar, incluyen imágenes con múltiples productos, simulando la perspectiva de un punto de compras o áreas de *checkout*. Estos conjuntos contienen 6,000 y 24,000 imágenes, respectivamente. Las imágenes están diseñadas para replicar situaciones reales en las que los productos pueden estar parcialmente ocultos o agrupados, lo que imita las condiciones habituales de una caja de *checkout*. Estas imágenes contienen promedio de más de seis productos diferentes por imagen, y en algunos casos, varios objetos del mismo tipo en una sola imagen. Al igual que en el subconjunto de entrenamiento, el subconjunto de validación ofrece *bounding boxes* precisos y etiquetas de clase para cada producto, facilitando así la evaluación de modelos de detección de objetos. Este subconjunto de validación se clasifica en tres niveles de dificultad: fácil, medio y difícil, basados en la cantidad de productos presentes en la imagen y el nivel de oclusión entre ellos. La figura 4.9 muestra ejemplos de imágenes correspondientes a cada uno de estos niveles de dificultad.

Aunque el subconjunto de entrenamiento de RPC no es ideal para la evaluación debido a su simplicidad (al presentar solo un objeto por imagen), no ocurre lo mismo con los subconjuntos de validación y prueba. Dada la naturaleza compleja y desafiante del problema de *checkout*, estos subconjuntos presentan escenarios más acordes con la realidad y cumplen con todos los requisitos mencionados al principio de esta sección. Por esta razón, se eligen los subconjuntos de validación y prueba de RPC para la evaluación del detector de objetos genéricos.

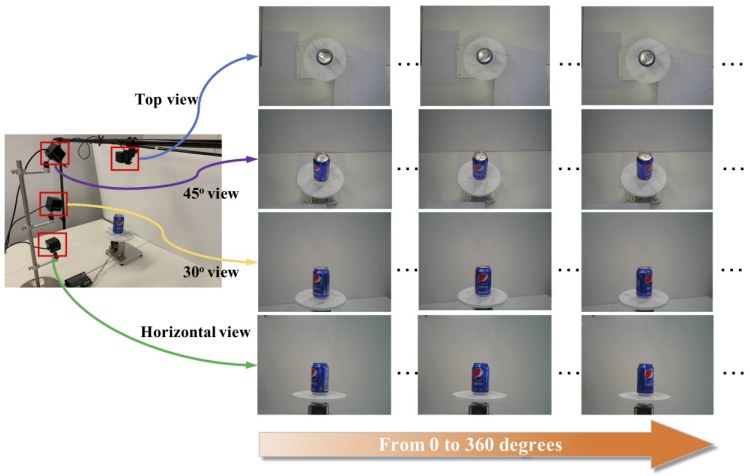


Figura 4.8: Equipos de recolección de imágenes de un solo producto. Imagen obtenida de [65].

(a) Modo fácil.

(b) Modo medio.

(c) Modo difícil.

Figura 4.9: Ejemplo de imágenes de *checkout* (validación) de 3 niveles de dificultad. Imagen obtenida de [65].

Figura 4.10: Conjunto de 40 objetos utilizados en el *dataset* DSLL. Imagen obtenida de [34].

DSLL

El *dataset* DSLL, presentado en [34], fue desarrollado para la evaluación de enfoques de reconocimiento de instancias de objetos desde la perspectiva del robot doméstico Bender, creado en el Laboratorio de Robótica de la Universidad de Chile. Las imágenes de este conjunto tienen una resolución de 1280x720 píxeles y ofrecen un campo de visión angular de 60° horizontal y 45° vertical. Este *dataset* incluye 40 objetos comunes en entornos domésticos, situados sobre una mesa o en un estante. De estos objetos, la mitad presenta texturas visuales y la otra mitad es de superficies sin textura. Es importante destacar que solo el 33 % de estos objetos se encuentran dentro de las 80 categorías del conjunto de datos COCO.

Para el subconjunto de entrenamiento, se capturaron 12 vistas diferentes de cada uno de los 40 objetos, rotándolos 30° entre cada toma consecutiva, lo que resulta en un total de 480 imágenes RGB y de profundidad. La figura 4.10 muestra ejemplos de la captura de cada uno de los 40 objetos incluidos en este *dataset*.

Para el conjunto de evaluación del DSLL, se emplearon diversas configuraciones que imitan condiciones reales de un entorno doméstico. Estas configuraciones se variaron en cuanto a:

- **Distancia al objeto**: 40 cm y 104.1 cm.
- **Fondo de la superficie**: blanco, colorido y desordenado.
- **Iluminación**: normal y baja.
- **Oclusión**: sin oclusión y con oclusión del 50 %.

Los escenarios de prueba configurados bajo estas variables son los siguientes:

- **S1:** Un objeto, fondo blanco, iluminación normal, sin oclusión.
- **S2:** Un objeto, fondo blanco, iluminación baja, sin oclusión.
- **S3:** Un objeto, fondo colorido, iluminación normal, sin oclusión.
- **S4:** Un objeto, varios fondos, iluminación normal, sin oclusión.

- **S5:** Un objeto, fondo blanco, iluminación normal, 50 % de oclusión.
- **M1:** Seis objetos, fondo colorido, iluminación normal.
- **M2:** Seis objetos, varios fondos, iluminación normal.
- **M3:** Seis objetos, fondo blanco, iluminación normal.
- **M4:** Seis objetos, fondo blanco, iluminación baja.

En la Figura 4.11, se presentan ejemplos representativos de imágenes pertenecientes a cada uno de los escenarios de evaluación mencionados. Cada escenario específico comprende 160 imágenes RGB, lo que resulta en un total de 1440 imágenes para el conjunto completo de evaluación de DSLL. Esta variedad en los escenarios y el número significativo de imágenes proporcionan una base sólida y diversa para evaluar de manera más efectiva el modelo en un entorno doméstico realista.

(a) Ejemplo escenario S1. (b) Ejemplo escenario S2. (c) Ejemplo escenario S3.

(d) Ejemplo escenario S4. (e) Ejemplo escenario S5. (f) Ejemplo escenario M1.

(g) Ejemplo escenario M2. (h) Ejemplo escenario M3. (i) Ejemplo escenario M4.

Figura 4.11: Ejemplo de escenarios de evaluación de DSLL.

Finalmente, habiendo definido los conjuntos de evaluación con los datasets RPC y DSLL, se procederá a calcular las métricas *F2-score*, *F3-score* y el *mAP* para estos conjuntos. Basándose en los resultados obtenidos, se seleccionará el modelo que demuestre el mejor

desempeño. Este modelo elegido será utilizado como el detector base en la red integrada de detección de instancias de objetos.

Los detalles de los resultados obtenidos tanto del entrenamiento como de la evaluación se presentarán en la Sección 5.1.

4.3. Entrenamiento extractor de descriptores

Una vez seleccionada la mejor versión de YOLOv7 modificado, el siguiente paso es el entrenamiento del módulo extractor de descriptores de objetos. A continuación, se describirá la metodología adoptada para implementar y entrenar este módulo. Se detallarán aspectos clave como la integración del módulo con la red de detección YOLOv7 modificado, la implementación del cabezal *sub-center ArcFace*, y los detalles relacionados con el entrenamiento, incluyendo las bases de datos utilizadas, las estrategias de *data augmentation* empleadas, y las arquitecturas consideradas para la evaluación del modelo.

4.3.1. Integración del módulo extractor al detector

Para lograr que la red de detección de instancias de objetos funcione como una red integrada, el módulo extractor de descriptores necesita ser una extensión de la red de detección de objetos genéricos YOLOv7 modificado. Esto implica unir una CNN en alguna de las etapas del *backbone* del detector. Los *stage 1* y *stage 2* de YOLOv7 modificado han sido seleccionados para la experimentación, ya que proporcionan características con información de alta resolución y menor nivel semántico.

Cabe mencionar, la importancia de contar con un *feature map* de entrada de alta resolución para el módulo extractor, debido al proceso de RoI Align que se emplea para extraer características más precisas relacionadas con la posición espacial del objeto, lo cual es relevante para captar correctamente objetos de tamaño pequeño y mediano.

Los *feature maps* de salida del *stage 1* del backbone presentan las siguientes dimensiones: una altura de $\frac{H_{entrada}}{4}$, un ancho de $\frac{W_{entrada}}{4}$ y una profundidad de 256 capas. Por otro lado, los *feature maps* resultantes del *stage 2* tienen dimensiones de $\frac{H_{entrada}}{8}$ por $\frac{W_{entrada}}{8}$, con una profundidad de 512 capas.

4.3.2. Arquitecturas CNN utilizadas

Como se mencionó en el capítulo anterior, se evaluarán dos arquitecturas diferentes para la extracción de características para descriptores: una diseñada por el autor, denominada JPNet, y otra basada en la red VoVnet. El objetivo de esta evaluación es analizar si existen diferencias significativas entre los tipos de arquitectura utilizados. Además, como referencia comparativa, se evaluará el rendimiento del módulo extractor de características sin el uso de

una CNN adicional. Esto se realiza con el propósito de examinar la relevancia de emplear una CNN especializada en la extracción de características de descriptores.

4.3.3. Configuración de RoI Align

La configuración específica de *RoI Align* utilizada en los experimentos es la siguiente:

- **Tamaño de Salida**: Se estableció un tamaño de salida de 14×14 para cada RoI, lo que proporciona una cuadrícula detallada de características para una representación más representativa y detallada de cada región propuesta.
- **Escala Espacial**: La escala espacial se ajustó a 1/4, 1/8 o 1/16, dependiendo del stage del *backbone* o de la CNN específica utilizada. Esto significa que el tamaño de la característica de salida corresponde a un cuarto o un octavo del tamaño de la imagen de entrada.
- **Radio de Muestreo**: Se configuró el radio de muestreo en -1, permitiendo que *PyTorch* determine automáticamente el número de puntos de muestreo en función del tamaño de la *RoI* y la operación de *pooling* aplicada.
- **Alineación**: Se activó la alineación precisa de los bordes configurando el parámetro `aligned` en `True`. Esto asegura que los bordes de cada *RoI* se alineen correctamente con los píxeles de la imagen, mejorando la precisión en la extracción de características.

4.3.4. Configuración de GeM Pooling

La configuración específica del *GeM Pooling* utilizada en los experimentos es la siguiente:

- **Parámetro** p: Se fijó el valor del parámetro p en 3. Este valor define la intensidad de la operación de *pooling*. Con $p = 3$, se pone mayor énfasis en los valores de características más altos dentro de cada región del *feature map*. Esta configuración resulta efectiva para resaltar las características más distintivas y relevantes de los objetos dentro de la región analizada.

4.3.5. Configuración del Neck

La configuración específica de la capa *neck* utilizada en los experimentos se detalla a continuación:

- **Capa Fully Connected (FC)**: Se utiliza una capa totalmente conectada (`nn.Linear`) para transformar las características de entrada a un espacio de 512 dimensiones. Esta dimensión se eligió basándose en investigaciones previas sobre la dimensionalidad óptima de los *embeddings* de características [66, 32, 64]. La capa se configura sin sesgo (`bias=False`).

- **Batch Normalization (BN)**: Después de la capa FC, se aplica una normalización por lotes (`nn.BatchNorm1d`) para estandarizar las 512 características.
- **Función de Activación SiLU**: Se utiliza la función de activación *SiLU* para introducir no linealidad sobre las 512 características.

El número de características de entrada de la capa FC (`in_features`) depende de la profundidad del *feature map* generado por las capas anteriores, lo que indica que la capa *neck* se adapta dinámicamente al tamaño del *feature map* de entrada.

4.3.6. Implementación de cabezal Sub-center ArcFace

Para capacitar al módulo extractor de características a generar *embeddings* representativos, es necesario entrenarlo como un problema de clasificación. En este contexto, se utiliza un cabezal personalizado basado en la técnica *Sub-center ArcFace*. A continuación, se detalla la implementación específica de este cabezal:

```
import torch
import torch.nn.functional as F

# Variables
x = ...  # Tensor de descriptores (embeddings)
W = ...  # Pesos de la capa lineal
labels = ...  # Etiquetas de clase
cos_m, sin_m = ...  # Valores precalculados para el margen angular
th, mm = ...  # Umbrales para la aplicaci n del margen
s = ...  # Factor de escalado
sub_centers = ...  # N mero de sub-centros
out_features = ...  # Dimensiones de la salida o n mero de clases

# Normalizar x y W
x = F.normalize(x, dim=1)
W = F.normalize(W, dim=1)

# Calcular cos(theta)
cosine = F.linear(x, W)

# Manejo de sub-centros
if sub_centers > 1:
    cosine = cosine.view(-1, out_features, sub_centers)
    cosine, _ = torch.max(cosine, dim=2)

# Calcular sin(theta)
sine = torch.sqrt(1.0 - torch.clamp(cosine**2, min=0, max=1))

# Calcular cos(theta + m)
phi = cosine * cos_m - sine * sin_m

# Aplicar margen
if easy_margin:
    phi = torch.where(cosine > 0, phi, cosine)
else:
```

```
    phi = torch.where(cosine > th, phi, cosine - mm)

# Crear m scara one-hot para las etiquetas
one_hot = torch.zeros_like(cosine, device=x.device)
one_hot.scatter_(1, labels.view(-1, 1).long(), 1)

# Combinar cos(theta + m) y cos(theta)
output = one_hot * phi + (1 - one_hot) * cosine

# Escalar el output final
output *= s
```

Código 4.1: Implementación del cabezal sub-center arcface

4.3.7. Base de datos

Para el entrenamiento eficaz del módulo extractor de descriptores, es necesario contar con una base de datos que no solo incluya las clases de cada objeto en la imagen, sino también las localizaciones de estos objetos. Esta necesidad se debe a que tanto la CNN como la capa *neck* del módulo se entrenan con una capa de *RoI Align* entre ellas. Esto significa que el aprendizaje de clasificación en el módulo no se realiza sobre la imagen completa de manera global, sino que se enfoca de manera local en las regiones específicas donde se encuentran los objetos. A continuación, se detalla el conjunto de datos seleccionado para el entrenamiento del módulo extractor de descriptores.

Para el entrenamiento del módulo extractor de descriptores, se utilizará el subconjunto de entrenamiento del *dataset* RPC [65]. A pesar de que esta base de datos ya fue empleada para evaluar el detector de objetos genéricos, como se menciona en la sección 4.2.3, en este contexto se utiliza su subconjunto de entrenamiento. RPC es particularmente adecuado para entrenar en un contexto de clasificación, ya que incluye imágenes centradas en diferentes vistas de 200 instancias de productos distribuidos en 17 meta categorías. La Figura 4.12 muestra ejemplos de objetos con forma de botella y bolsa del conjunto de entrenamiento.

Para validar el entrenamiento, se empleará el conjunto de validación de RPC, mostrado en la figura 4.9. El uso de un conjunto de validación con una distribución de objetos diferente al de entrenamiento es ventajoso, ya que simula más fielmente la aplicación en un entorno real donde el objetivo es reconocer una variedad de objetos.

4.3.8. Formato de entrenamiento

Aunque la función de pérdida utilizada para el entrenamiento del módulo extractor de descriptores es una función de pérdida de clasificación multiclase, es crucial que el conjunto de datos empleado contenga tanto las clases de los objetos como sus localizaciones. Por lo tanto, el formato de entrenamiento adoptado para el módulo extractor de descriptores es el mismo que se utiliza para el detector, es decir, el formato YOLO. Esta elección asegura que el módulo aprenda no solo a clasificar objetos, sino también a comprender su ubicación espacial en la imagen.

(a) Productos con forma de botella.

(b) Productos con forma de bolsa.

Figura 4.12: Ejemplos de productos con forma de botella y bolsa en el conjunto de entrenamiento RPC.

4.3.9. Data augmentation

Al igual que en el caso de la detección, es fundamental aplicar diversas estrategias de *data augmentation* para garantizar un aprendizaje adecuado del módulo a entrenar. Sin embargo, estas estrategias difieren ligeramente de las usadas en el detector. Por ejemplo, se reduce la variación de color para evitar confusiones en el reconocimiento, se añaden rotación, *shear*, perspectiva y volteo horizontal para promover la invariancia de los descriptores a estos cambios. Además, se incorpora *copy-paste* para aumentar la cantidad de objetos por imagen y la oclusión entre ellos. A continuación se detallan los valores específicos para cada técnica de *data augmentation*:

- **Ajuste de tono HSV:** Variación aleatoria del tono (Hue) en un 1.5 %, manteniendo constante la saturación y el valor.
- **Rotación:** Rotaciones aleatorias en un rango de ±0.2 grados.
- **Traslación:** Traslaciones aleatorias de hasta un 10 % de las dimensiones de la imagen en ambos ejes.
- **Escalado:** Modificación aleatoria del tamaño de las imágenes en un rango de ±10 %.
- **Shear:** Deformaciones de *shear* con un rango de ±0.1 grados.
- **Perspectiva:** Cambios de perspectiva con una fracción máxima de 0.0005.
- **Volteo vertical y horizontal:** Volteo aleatorio de las imágenes en ambas direcciones con una probabilidad del 50 %.
- **Mosaico:** Combinación de múltiples imágenes en una sola con una probabilidad del 100 %. [1].

- **Mixup:** Superposición de imágenes con una probabilidad del 50 %.[1].
- **Copy-Paste:** Incorporación de objetos de una imagen a otra con una probabilidad del 50 %. [12]
- **Paste-In:** Introducción de objetos adicionales en las imágenes con una probabilidad del 50 %. [12]

Cabe destacar, la importancia de la *traslación*, *mosaico*, *mixup*, *copy-paste* y *paste-in* en estos entrenamientos. Sin el uso de estas técnicas, las características aprendidas tenderían a concentrarse solo en la zona central de las imágenes del conjunto de entrenamiento de RPC. Esto podría provocar un rendimiento deficiente en la detección de objetos ubicados en los bordes de la imagen.

4.3.10. Configuración de entrenamiento

El proceso de entrenamiento del módulo extractor de descriptores es la misma, independientemente de la arquitectura de CNN utilizada. Como se ha mencionado anteriormente en esta sección, es esencial contar con etiquetas de clase y de cajas delimitadoras (*bbox*) para el entrenamiento. Dado que el objetivo es enseñar a la red a extraer descriptores de la mejor forma posible, los cuadros delimitadores suministrados directamente a la capa *RoI Align* son los *bbox* de verdad (*ground truth*), y no los generados por la red YOLOv7 modificado ya entrenada. Utilizar los *bbox* predichos por la red de detección podría introducir imprecisiones en la extracción de la región de interés del objeto, lo que afectaría negativamente el rendimiento del entrenamiento del módulo.

Adicional a lo mencionado anteriormente, es necesario realizar la congelación (conocido como freeze en inglés) de todas las capas de la red YOLOv7 modificado, incluyendo los *stages* y la capa de *stem* que se comparte con el módulo reconocedor. Este paso se lleva a cabo para evitar perturbaciones en los resultados obtenidos del entrenamiento de la red de detección de objetos genéricos. Dado que el entrenamiento del módulo reconocedor se realiza sobre la red ya entrenada, congelar las capas permite que los parámetros entrenables del extractor se adapten adecuadamente a las capas iniciales del *backbone* de YOLOv7 modificado, sin alterar el aprendizaje previamente adquirido.

Los entrenamientos del módulo extractor de descriptores se realizaron utilizando una GPU Tesla V-100 de 32GB en un servidor DGX. Los hiperparámetros seleccionados para el entrenamiento son los siguientes:

- **Número de épocas:** Se utilizaron 100 épocas para el entrenamiento.
- **Optimizador:** Se eligió el optimizador SGD.
- **Tasa de aprendizaje (Learning rate):** Se inició con una tasa de aprendizaje de 0,01, que disminuye progresivamente siguiendo un esquema de decaimiento definido por *OneCycleLR* [19]. En este esquema, la tasa de aprendizaje se reduce en un factor de 0,1, finalizando en 0,001. La evolución de la tasa de aprendizaje durante el entrenamiento se ilustra en la figura 4.6.

- **Momentum:** Se estableció un valor de *momentum* en 0,937.
- **Tamaño del lote (Batch size):** Se utilizó un tamaño de lote de 32.
- **Peso de regularización:** Se aplicó una regularización L2 con un peso de 0,0005 para prevenir el sobreajuste.

En relación con el cabezal de *Sub-center ArcFace*, los hiperparámetros se configuraron de la siguiente manera:

- **Número de subcentros**: Se utilizaron 3 subcentros, basándose en las recomendaciones de trabajo [16].
- **Escalar s**: Se eligió un escalar adecuado para las dimensiones del vector de características de entrada al cabezal, que es de 512. De acuerdo con los estudios [4] [3] [16], para esta dimensión se recomienda un escalar de 64.
- **Margen m**: Se estableció un margen m de 0.5, siguiendo las pautas de los trabajos citados [3].
- **Número de Clases**: El número de clases está definido por el dataset RPC, que cuenta con 200 clases diferentes.

Al igual que en el entrenamiento del detector, las capas de la CNN y del *neck* se entrenan desde cero, sin necesidad de un pre-entrenamiento previo. Las únicas capas que conservan entrenamientos previos son las capas congeladas del *backbone* de YOLOv7 modificado.

4.4. Evaluación del extractor de descriptores

Tras finalizar los entrenamientos de las diferentes arquitecturas del módulo extractor de descriptores, la fase de evaluación e inferencia requieren de una adaptación importante: retirar el cabezal *Sub-center ArcFace* utilizado durante el entrenamiento y reemplazarlo por el clasificador de conjunto abierto, descrito previamente en la sección 3.2. Este cambio es fundamental para evaluar tanto la capacidad discriminativa de los descriptores generados por el módulo con respecto a objetos no vistos durante el entrenamiento, como su eficiencia en identificar objetos desconocidos. Para enfocar el análisis en el rendimiento del extractor, las regiones de interés procesadas por la capa *RoI* se corresponden con los *bbox* de verdad (*ground truth*), y no con los generados directamente por el detector de objetos genéricos.

4.4.1. Definición de términos

Aunque el enfoque de esta sección es evaluar específicamente el extractor de descriptores, los conceptos básicos son los mismos mencionados en la sección 4.2.2, con la diferencia que aquí se tiene en consideración las instancias de objetos y la categorización de objetos desconocidos.

Para simplificar la evaluación de los modelos y facilitar su comparación con el sistema completo, cuando un objeto se considera desconocido es equivalente a considerar que el detector no localizo el objeto, por lo cual si el objeto en cuestión corresponde a un objeto conocido dentro de la base de datos de referencia, esta detección corresponderá a un falso negativo.

Accuracy

En el contexto de la detección de instancias de objetos, el *accuracy* [39] es una métrica que mide la proporción de identificaciones correctas, tanto positivas como negativas, realizadas por el modelo en relación con el total de predicciones. Esta métrica proporciona una visión general del rendimiento del modelo, considerando tanto las identificaciones correctas de objetos (verdaderos positivos) como las correctas identificaciones de no objetos (verdaderos negativos).

La fórmula para calcular la *accuracy* se presenta a continuación:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \tag{4.6}$$

La *accuracy* es una métrica útil para obtener una vista rápida de la efectividad general del modelo, pero es importante complementarla con otras métricas como la *precision* y el *recall* para una evaluación más completa.

4.4.2. Métrica de evaluación

Dado que tanto la *precision* como el *recall* son métricas relevantes en este contexto, se opta por utilizar el *F1-score*, una métrica que combina ambas. El *F1-score* es particularmente útil cuando se busca un equilibrio entre la precisión de las detecciones y la capacidad del modelo para identificar todos los objetos relevantes.

A diferencia del *F1-score* de detección sin consideración de clases, aquí se tiene que considerar las instancias de clases, por lo que el *F1-score* se calcula en base al promedio del *F1-score* de cada instancia de clase evaluada.

4.4.3. Base de datos

Para evaluar el rendimiento de discriminación de las arquitecturas entrenadas, se utilizará una base de datos específica. La base de datos a utilizar corresponde a DSLL, detallada previamente en la sección 4.2.3. Este conjunto es adecuado para esta evaluación, ya que incluye objetos que no se presentaron en el conjunto de entrenamiento RPC, ofreciendo así un escenario ideal para evaluar la capacidad del modelo de discriminar nuevos objetos.

Una razón adicional para seleccionar el conjunto de datos DSLL es su uso en el modelo YOLOSPoC, que sirvió de inspiración para este trabajo. Al emplear el mismo conjunto de datos y la métrica de evaluación *F1-score*, se facilita la comparación directa entre los resultados del presente modelo y los de YOLOSPoC.

Los escenarios específicos del conjunto de evaluación DSLL que se utilizarán son S1, M1, M2, M3 y M4. Estos escenarios abarcan una variedad de condiciones y desafíos, lo que proporcionará una evaluación exhaustiva del rendimiento del modelo.

Ya con la métrica de evaluación y el conjunto de datos definido, se pueden realizar los experimentos de evaluación. Sin embargo, aún queda por definir la metodología de la obtención de descriptores de referencia, que utilizará el reconocedor de instancias de objetos.

4.4.4. Obtención de descriptores de referencia

Los descriptores de instancias de objetos de referencia representan aquellas instancias que el sistema está configurado para identificar. Estos descriptores actúan como una base de conocimiento contra la cual se comparan las entradas para realizar identificaciones. Para que el usuario pueda registrar un nuevo objeto a esta base de datos de referencia, se debe seguir los siguientes pasos:

1. **Captura de imágenes**: El usuario debe tomar una o más fotografías del objeto o de los objetos que se desea incorporar al sistema. La cantidad imágenes requeridas dependerá de la asimetría y características distintivas del objeto. Una imagen puede contener más de un objeto.
2. **Etiquetado manual**: Tras la captura, es necesario que el usuario etiquete manualmente las ubicaciones de las instancias del objeto dentro de las imágenes. Además, es importante que cada instancia esté numerada consecutivamente, comenzando con el número 0. El formato de etiquetado necesario es el formato YOLO. El nombre del archivo con las etiquetas debe tener el mismo *filename* que su imagen respectiva. Adicionalmente, es necesario crear un archivo de texto separado donde se enumeren los nombres específicos asociados a cada instancia de objeto etiquetada. Es de suma importancia asegurar que la numeración usada para las etiquetas concuerde con los nombres especificados en este archivo.
3. **Extracción de descriptores**: Se debe ejecutar el script `get_features.py` disponible en el código. Este programa procesará la imagen junto con sus respectivas etiquetas, generando, como resultado, un archivo que consolida y almacena la base de datos de descriptores de referencia.

Una vez finalizado este proceso, el sistema está capacitado para asignar una etiqueta a cada propuesta de detección. Esta etiqueta puede identificar a la propuesta como una instancia de objeto previamente registrada en la base de datos o, en su defecto, como una instancia desconocida.

Finalmente, para la evaluación del extractor de descriptores en todos los escenarios mencionados, se ha seleccionado el conjunto S1 del *dataset* DSLL como la base de datos de referencia para el cálculo de los descriptores de referencia. Esta elección se basa en las vistas que proporciona el conjunto S1, el cual incluye diferentes perspectivas ideales de las 40 instancias de objetos. Utilizar este conjunto específico asegura que los descriptores de referencia se generen a partir de representaciones variadas de cada objeto, lo que es crucial para una evaluación efectiva del rendimiento del extractor de descriptores.

Los detalles de los resultados obtenidos tanto del entrenamiento como de la evaluación se presentarán en la Sección 5.2.

4.5. Evaluación detector de instancias de objetos

Con la selección del mejor modelo para la extracción de descriptores, se cuenta con las herramientas necesarias para llevar a cabo una evaluación exhaustiva del sistema propuesto. Esto implica que las detecciones utilizadas por la capa de *RoI Align* del extractor de descriptores son las producidas por el detector al cual está conectado y con el que fue entrenado. Adicionalmente, se medirá la velocidad de inferencia del sistema completo para validar la hipótesis de que opera en tiempo real, lo cual se define como la habilidad para inferir instancias de objetos a una tasa mínima de 24 fotogramas por segundo. Se procederá también a comparar el rendimiento con el modelo YOLOSPoC, evaluando tanto el *f1-score* como la velocidad de inferencia. Esta comparativa tiene como finalidad establecer si el sistema diseñado en esta tesis presenta ventajas frente a YOLOSPoC. Como paso final, se presentarán visualizaciones gráficas de las detecciones realizadas por el modelo seleccionado en tres escenarios representativos, con el objetivo de corroborar visualmente la representación de las métricas logradas.

En las siguientes sub-secciones, se especificarán los términos y métricas que se aplicarán en la evaluación del sistema completo. Se describirá la base de datos utilizada para las evaluaciones y, por último, se describirán los métodos empleados para llevar a cabo las comparativas y obtener visualizaciones de las detecciones en los escenarios seleccionados.

4.5.1. Métricas de evaluación

Para evaluar el sistema de detección de instancias de objetos, es necesario emplear métricas que permitan medir el rendimiento en términos de precisión del sistema como el rendimiento en términos de velocidad de procesamiento. Estas métricas son claves para determinar la efectividad del sistema en aplicaciones reales y para realizar una comparación con otros modelos como YOLOSPoC.

En cuanto a la precisión del sistema, se utilizarán las métricas ya definidas para la evaluación del extractor de descriptores, es decir, *recall*, *precision* y *f1-score* [39]. Estas métricas son adecuadas ya que fueron definidas ya en el marco de la detección de instancias de objetos.

Para medir el rendimiento en términos de velocidad de procesamiento, se define una

métrica común conocida como fotogramas por segundo o FPS.

Fotogramas por segundo

Es una métrica para medir la velocidad de inferencia de un modelo, corresponde al número de fotogramas por segundo (FPS) [51]. Esta métrica indica la cantidad de imágenes (fotogramas) que el modelo es capaz de procesar en un segundo. Una mayor tasa de FPS significa que el modelo puede procesar imágenes más rápidamente, lo cual es necesario para aplicaciones que requieran modelos en tiempo real. Para los detectores de objetos, se considera que un detector funciona en tiempo real cuando este procesa al menos a 30 FPS.

4.5.2. Base de datos

Para la evaluación del sistema propuesto, se utilizará la misma base de datos mencionada en la sección anterior. Esta elección se debe a que DSLL no solo sirve como *dataset* para el reconocimiento de instancias de objetos, sino también para la detección. Este conjunto es adecuado para esta tarea ya que incluye vistas con múltiples objetos en diversas posiciones, lo que permite evaluar la capacidad del sistema para detectar y reconocer instancias de objetos en diferentes contextos.

Los escenarios seleccionados para evaluar los dos mejores modelos serán los conjuntos S1, S2, S3, S4, M1, M2, M3 y M4. Se han incluido más escenarios en este análisis porque es relevante analizar cómo se desempeña la detección y el reconocimiento de instancias en la mayor cantidad de escenarios posibles, lo que permite una evaluación exhaustiva del sistema bajo distintas condiciones. Utilizar DSLL para esta evaluación también asegura una comparabilidad directa entre las diferentes etapas de evaluación, lo que proporciona una interpretación más clara de los resultados en términos de mejoras o cambios en el rendimiento del sistema. Además, como se mencionó en secciones anteriores, YOLOSPoC ya ha sido evaluado en los escenarios S1, S2, S3, S4, M1 y M2, facilitando así la comparación directa entre el modelo propuesto y este sistema.

4.5.3. Configuración de umbrales de reconocimiento

Para llevar a cabo la evaluación completa del sistema utilizando los dos mejores modelos de extracción de descriptores, es necesario establecer un umbral de reconocimiento específico para cada uno. Este umbral es crucial, ya que permite al sistema discernir si un objeto es conocido o desconocido. El valor del umbral determina la sensibilidad del sistema, estableciendo la distancia máxima aceptable entre una instancia de objeto detectada y su instancia de referencia más cercana. Un umbral más alto hará que el sistema sea más permisivo al clasificar un objeto como conocido, mientras que un umbral más bajo tendrá el efecto opuesto.

Aunque cada modelo podría necesitar un umbral diferente para cada escenario de evaluación, en este trabajo se ha optado por asignar un valor umbral que optimice el *f1-score* promedio en todos los escenarios DSLL. Este enfoque busca establecer un umbral general

que favorezca la generalización del modelo, permitiéndole reconocer instancias de objetos en una amplia variedad de situaciones.

4.5.4. Comparación del sistema completo con los mejores extractores de descriptores

Es posible que el mejor modelo en la evaluación de extracción de descriptores no mantenga el mismo nivel de rendimiento al ser integrado en el sistema completo. Por esta razón, se lleva a cabo una evaluación comparativa del desempeño de los dos mejores modelos. Para ello, se consideran los escenarios S1, S2, S3, S4, M1, M2, M3 y M4. Al evaluar los modelos dentro del sistema completo, se procede a calcular tanto las métricas de precisión como el tiempo de inferencia junto con sus FPS correspondientes.

Es importante destacar que, para la determinación de las métricas de precisión, se buscó para cada modelo el valor umbral de reconocimiento que maximiza el f1-score promedio a través de todos los escenarios evaluados. En cuanto a las pruebas de velocidad, estas se realizaron utilizando una tarjeta gráfica GeForce RTX3070, calculando el tiempo promedio de 100 inferencias con una resolución de imagen de entrada de 640x480. El tiempo medido abarca desde la recepción de la imagen hasta la entrega de los objetos detectados con sus respectivos descriptores. Además, se configuró la cantidad máxima de detecciones procesadas por imagen (max_det) en 100.

Los criterios para seleccionar el mejor modelo se basan, en primer lugar, que el sistema completo funcione en tiempo real, lo que implica un rendimiento superior a 24 FPS. En segundo lugar, se considera el modelo que logra el mejor F1-score promedio en todos los escenarios evaluados.

4.5.5. Comparación mejor modelo propuesto vs YOLOSPoC

Con la selección del mejor modelo propuesto, se procede a compararlo con el modelo de referencia de esta tesis, YOLOSPoC, tanto en términos de precisión como de velocidad. Para ello, se eligen los escenarios de DSLL en los que YOLOSPoC cuenta con métricas de F1-score publicadas [33]. Esta comparativa es viable ya que ambos modelos se someten a evaluación con el mismo conjunto de datos y el mismo número de instancias de objetos, cambiando únicamente en el número de imágenes de referencia consideradas. Las métricas de F1-score del mejor modelo propuesto son las mismas que se obtuvieron en la sección de comparación anterior.

Con respecto al tiempo de inferencia y los FPS de YOLOSPoC, se estiman basándose en los tiempos de inferencia de su detector YOLOv3 y su extractor de descriptores ResNet101, calculados en el mismo entorno de procesamiento con una tarjeta gráfica GeForce RTX3070. El tiempo de inferencia de YOLOv3 se obtiene del promedio de 100 inferencias con una imagen de entrada de resolución 640x480. Por otro lado, el tiempo de inferencia de ResNet101 se estima con base en el promedio de 100 inferencias de una imagen de 60x40, tamaño seleccionado bajo el supuesto promedio de la resolución que podría tener un objeto al calcular

su descriptor. La suma de estos tiempos nos brinda una estimación conservadora del tiempo de inferencia para YOLOSPoC, considerando además que este valor solo contempla una detección por imagen. Bajo condiciones normales, debido a que YOLOv3 utiliza un umbral relativamente bajo, el número de propuestas de objetos tiende a ser alto, lo que incrementaría aún más el tiempo de inferencia del modelo.

4.5.6. Análisis resultados finales sistema propuesto

Como última evaluación, se visualizan los resultados del sistema propuesto en varios escenarios, utilizando el mismo umbral de reconocimiento que se empleó en la comparativa con YOLOSPoC.

Para realizar un análisis completo pero conciso, se optó por seleccionar los escenarios más representativos de DSLL; estos son S1, M3 y M2. El escenario S1, debido a su simplicidad, es ideal para examinar el rendimiento del reconocedor. Por otro lado, M3, que es un escenario similar a S1 pero con seis instancias de objetos en lugar de una, es útil para evaluar la capacidad del detector. Finalmente, el escenario M2, que corresponde al escenario más complejo con seis objetos en diferentes fondos e iluminaciones, es ideal para evaluar las dificultades del sistema y las posibles mejoras que podría tener.

Además, se realiza un análisis de los resultados visuales obtenidos por el modelo en el conjunto de datos RPC, centrado en seis tipos de instancias de objetos como referencia.

Los resultados detallados de la evaluación del detector de instancias de objetos se presentarán en la Sección 5.3.

Capítulo 5

Resultados y análisis

En este capítulo, se presentan y analizan los resultados obtenidos de los experimentos realizados, abarcando tanto los entrenamientos como las evaluaciones del detector de objetos genéricos, el módulo extractor de descriptores y el detector de instancias de objetos final. La estructura del capítulo se organiza en tres secciones principales: (i) resultados y análisis del entrenamiento y evaluación del detector de objetos genérico; (ii) resultados y análisis del entrenamiento y evaluación del módulo extractor de descriptores; (iii) resultados y análisis del detector de instancias de objetos final.

5.1. Detector de objetos genéricos

En esta sección se detallan los resultados de los entrenamientos realizados de la red YOLOv7 modificado, utilizando tanto el conjunto de datos COCO original como el conjunto COCO etiquetado con SAM. Se presentarán gráficos que muestran la evolución del *loss*, la *precision*, el *recall* y el *mAP* a lo largo de las épocas de entrenamiento. Se selecciona la época donde el modelo en entrenamiento alcanza las métricas óptimas de validación y se presenta la curva de *precision-recall*, para observar el comportamiento del modelo ante diferentes umbrales de confianza en la detección

A continuación, se presentan los resultados de la evaluación de los modelos al término del entrenamiento en los conjuntos seleccionados (RPC Eval, DSLL), contrastándolos con un enfoque *baseline*, representado por el detector de clases de objetos YOLOv7 entrenado con COCO original.

5.1.1. Entrenamiento YOLOv7 modificado con COCO original

Durante el proceso de entrenamiento de YOLOv7 modificado con COCO original, se observó cómo las pérdidas y las métricas de rendimiento evolucionan con cada época. En la figura 5.1, se muestra la evolución de las pérdidas de localización *Box* y de *Objectness* en el conjunto de entrenamiento. Además, se presentan las trayectorias de las métricas de

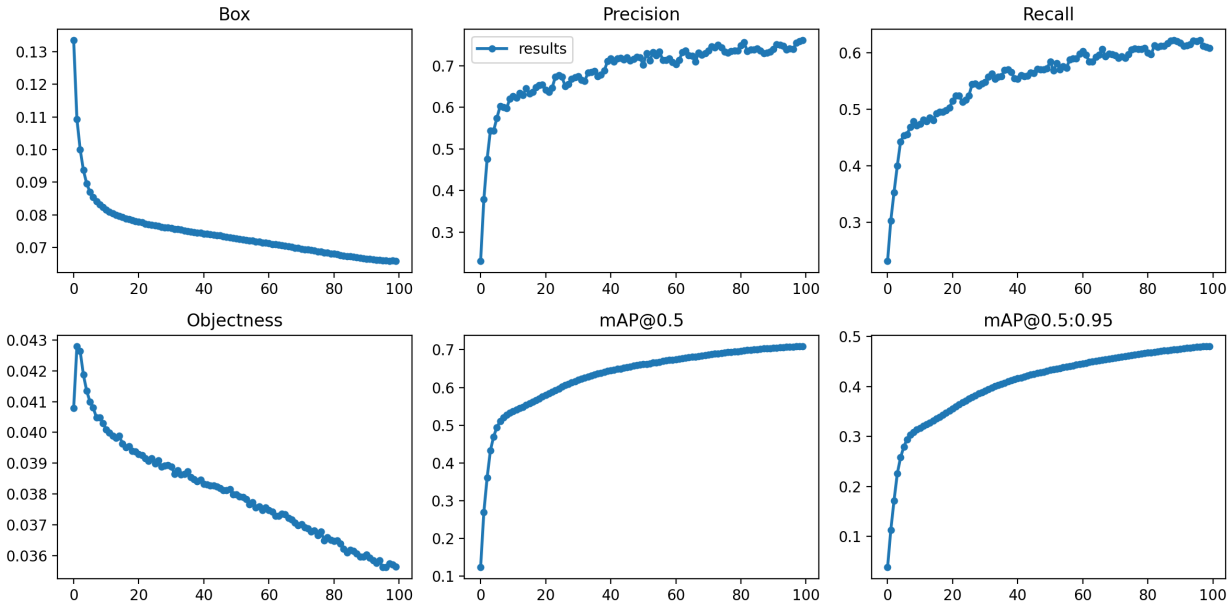


Figura 5.1: Evolución de las pérdidas y las métricas de rendimiento durante el entrenamiento de YOLOv7 modificado con el conjunto de datos COCO original.

Tabla 5.1: Resultados obtenidos de la mejor época de entrenamiento de la red YOLOv7 modificado con COCO original.

Época	***Box***	***Objectness***	**Total**	**P**	**R**	**mAP@.5**	**mAP@.5:.95**
100	0.066	0.036	0.102	0.762	0.609	0.709	0.481

precision, *recall*, *mAP0.5* (umbral *IoU* de 0.5) y *mAP0.5:0.95* (promedio de los umbrales *IoU* de 0.5 hasta 0.95) en el conjunto de validación de COCO.

En la figura 5.1, se evidencia la convergencia del modelo YOLOv7 modificado durante su entrenamiento con el conjunto de datos COCO original. Se observa una disminución significativa en las pérdidas de *Box*, que van de 0.134 a 0.066, y de *Objectness*, que se reducen de 0.041 a 0.036, conforme avanzan las épocas. Simultáneamente, se aprecia un aumento notable en las métricas de rendimiento en las primeras épocas, seguido de una tendencia a la estabilización hacia las últimas. Esto sugiere que el modelo evita el sobreajuste durante el entrenamiento, evidenciado en que las métricas alcanzan su punto máximo en la última época, mientras que las pérdidas continúan decreciendo. Los resultados presentados en la tabla 5.1 corroboran esta observación, destacando que la época 100, que es la última, muestra las métricas más favorables.

Los valores de *P*, *R*, *mAP@.5* y *mAP@.5:.95* que se presentan en la tabla 5.1 demuestran que, en la época 100, el modelo YOLOv7 modificado, que no incluye la clasificación de objetos, realiza detecciones efectivas en el conjunto de validación de COCO original.

La eficacia del modelo se ve reflejada en la curva de *precision-recall* mostrada en la figura 5.2. En esta curva, se aprecia cómo, a diferentes umbrales de confianza, tanto la *precision* como el *recall* del modelo mantienen un equilibrio consistente. En escenarios de alta confianza,

donde el modelo es más riguroso en sus detecciones, se observa que el *recall* es más bajo, pero la precisión se aproxima a 1. En contraste, con umbrales de confianza más bajos, el *recall* se incrementa significativamente, aunque la *precision* disminuye, lo que sugiere un aumento en los falsos positivos. Es importante destacar que en la curva, la *precision* desciende a 0 con un *recall* aproximado de 0.96, indicando que en umbrales bajos la proporción de falsos positivos crece notablemente en comparación con los verdaderos positivos.

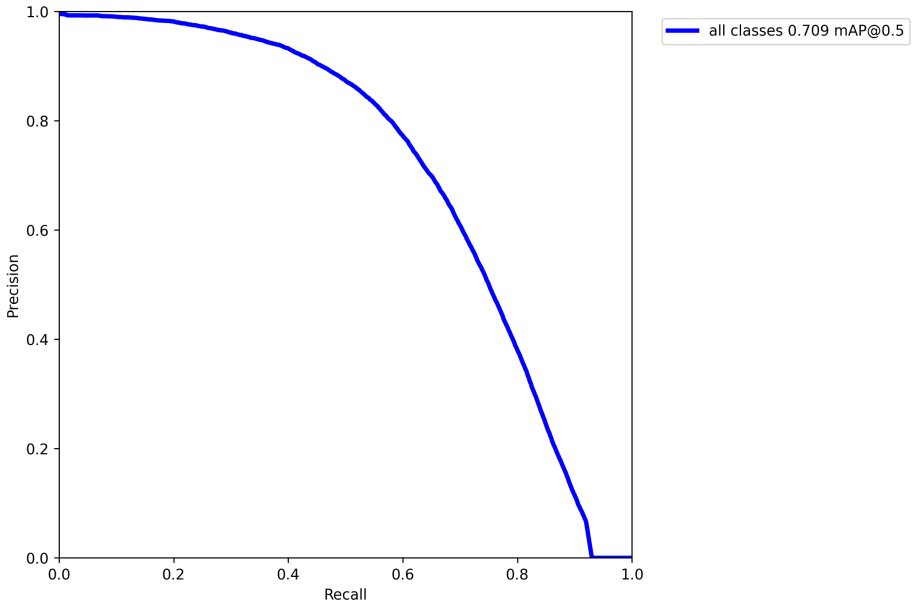


Figura 5.2: Curva de *precision-recall* para el modelo YOLOv7 modificado con COCO original.

Después de evaluar los resultados obtenidos en estas validaciones, se selecciona el modelo de la época 100 de YOLOv7 modificado, entrenado con el conjunto de datos COCO original, como el candidato representante para la evaluación y comparación en la selección del detector de objetos genéricos.

5.1.2. Entrenamiento YOLOv7 modificado con COCO etiquetado con SAM

Para el entrenamiento de YOLOv7 modificado utilizando el conjunto de datos COCO etiquetado con SAM, la evolución tanto de la pérdida como de las métricas de rendimiento se puede apreciar en la figura 5.3. En esta figura, se destaca la convergencia del modelo durante el proceso de entrenamiento, reflejada en una reducción notable de las pérdidas de *Box* (de 0.127 a 0.059) y de *Objectness* (de 0.236 a 0.227). Además, el comportamiento de las métricas de rendimiento en este caso es similar al observado con COCO original. Es importante mencionar que no se detecta evidencia de sobre ajuste en el entrenamiento del modelo.

Los valores correspondientes a la mejor época, que resultaron ser de la época 100, se pueden encontrar en la tabla 5.2. Estos resultados sugieren que el modelo YOLOv7 modificado está adecuadamente capacitado para efectuar destacables detecciones en el conjunto de

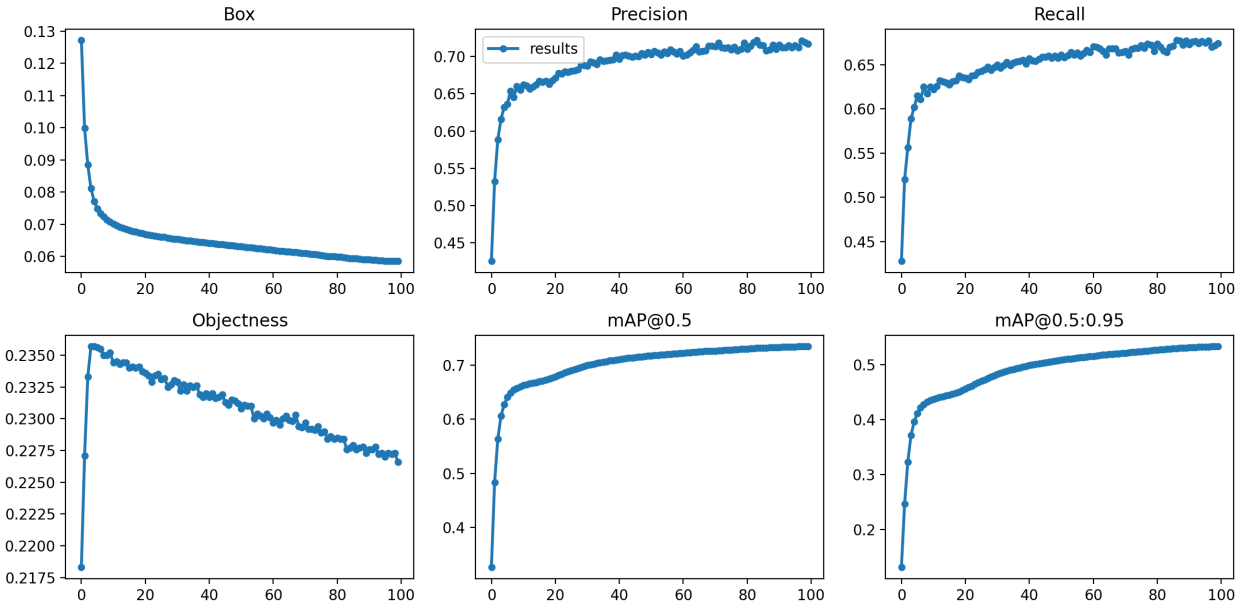


Figura 5.3: Evolución de las pérdidas y las métricas de rendimiento durante el entrenamiento de YOLOv7 modificado con el conjunto de datos de COCO etiquetado con SAM.

Tabla 5.2: Resultados obtenidos de la mejor época de entrenamiento de la red YOLOv7 modificado con COCO etiquetado con SAM.

Época	***Box***	***Objectness***	**Total**	**P**	**R**	**mAP@.5**	**mAP@.5:.95**
100	0.059	0.227	0.285	0.717	0.674	0.735	0.533

validación de COCO etiquetado con SAM.

Finalmente, el rendimiento satisfactorio del entrenamiento se corrobora a través de la curva *precision-recall* mostrada en la figura 5.4. Esta curva demuestra un patrón que es indicativo de un buen detector. No obstante, se observa un fenómeno similar al del modelo entrenado con COCO original: cuando el *recall* alcanza un valor de aproximadamente 0.9, la *precision* desciende a 0. Este resultado implica que, bajo umbrales bajos, la cantidad de falsos positivos aumenta significativamente en comparación con los verdaderos positivos.

5.1.3. Evaluación YOLOv7 modificado

Tras seleccionar los pesos óptimos de los entrenamientos, se procede a evaluar los modelos YOLOv7 modificados en los conjuntos de datos designados. Para establecer un estándar comparativo y analizar la eficacia del enfoque propuesto, se incluye la versión original de YOLOv7, entrenada con el conjunto de datos COCO y una confianza de 0,001, como método *baseline*. Así, los modelos a comparar son tres: YOLOv7 entrenado con COCO (YOLOv7-COCO), YOLOv7 modificado entrenado con COCO (YOLOv7mod-COCO) y YOLOv7 modificado con SAM (YOLOv7mod-SAM). A continuación, se presentan y analizan los resultados de estos tres modelos en los conjuntos de validación RPC y DSLL.

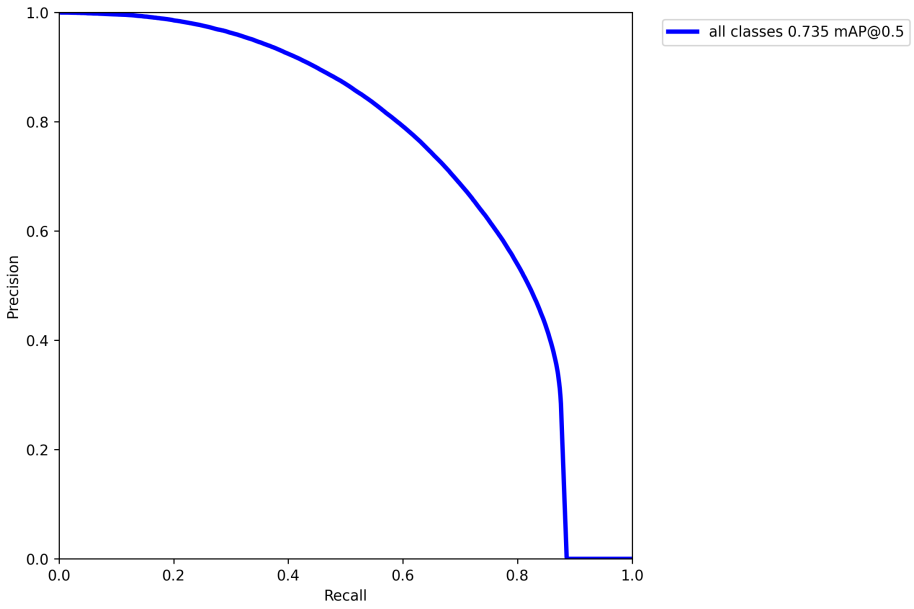


Figura 5.4: Curva de *precision-recall* para el modelo YOLOv7 modificado con COCO etiquetado con SAM.

Tabla 5.3: Resultados de evaluación de modelos en el conjunto de evaluación de RPC.

Modelo	**P**	**R**	**AP@.5**	**AP@.5:.95**	**F1**	**F2**	**F3**
YOLOv7-COCO	0,975	0,096	0,1	0,081	0,175	0,117	0,106
YOLOv7 mod-COCO	0,96	0,129	0,132	0,11	0,228	0,157	0,142
YOLOv7 mod-SAM	0,315	0,854	0,541	0,379	0,549	0,647	0,729

Validación RPC

La tabla 5.3 muestra los resultados de la evaluación de los tres modelos en el conjunto de validación RPC. En ella, se evidencian diferencias significativas en el rendimiento de cada modelo en relación con cada una de las métricas evaluadas.

El modelo YOLOv7 original, entrenado con el conjunto de datos COCO, demuestra una alta *precision* (0.975), pero su *recall* es notablemente bajo (0.096). Este patrón sugiere que, a pesar de la alta precisión en sus detecciones, el modelo tiende a pasar por alto una cantidad significativa de objetos reales. Este comportamiento era de esperar, ya que el conjunto de validación RPC solo incluye aproximadamente un 1.25 % de las clases de COCO (únicamente la clase botella). Como resultado, las métricas *AP*, *F1-score*, *F2-score* y *F3-score* resultan ser relativamente bajas, señalando que este modelo no es el más adecuado para las detecciones en el conjunto de datos RPC.

En contraste, el modelo YOLOv7 modificado entrenado con COCO muestra un incremento en el *recall* (de 0.033) en comparación con el modelo original, lo que implica una capacidad ligeramente mejorada para detectar objetos, aunque con una pequeña disminución en la precisión (de 0.015). Estas variaciones se reflejan en un aumento en las métricas de *AP@0.5* (de 0.032), *AP@0.5:0.95* (de 0.029) y *F-scores* (de 0.053, 0.04, 0.036). Esto indica que el modelo

Tabla 5.4: Resultados de evaluación en escenarios del conjunto de datos DSLL.

Set	Modelo	P	R	AP@.5	AP@.5:.95	F1	F2	F3
M1	YOLOv7-COCO	0,967	0,33	0,332	0,29	0,492	0,38	0,353
	YOLOv7mod-COCO	0,975	0,415	0,416	0,378	0,582	0,469	0,44
	YOLOv7mod-SAM	0,358	0,954	0,54	0,407	0,602	0,727	0,818
M2	YOLOv7-COCO	0,88	0,381	0,376	0,335	0,531	0,429	0,403
	YOLOv7mod-COCO	0,961	0,511	0,507	0,462	0,667	0,564	0,536
	YOLOv7mod-SAM	0,335	0,949	0,442	0,329	0,556	0,7	0,802
M3	YOLOv7-COCO	0,912	0,356	0,356	0,302	0,512	0,405	0,379
	YOLOv7mod-COCO	0,758	0,456	0,42	0,378	0,569	0,495	0,475
	YOLOv7mod-SAM	0,381	0,983	0,612	0,435	0,645	0,754	0,849
M4	YOLOv7-COCO	0,931	0,324	0,323	0,279	0,48	0,372	0,346
	YOLOv7mod-COCO	0,517	0,39	0,265	0,238	0,445	0,41	0,4
	YOLOv7mod-SAM	0,376	0,972	0,644	0,441	0,645	0,757	0,839
Mean	YOLOv7-COCO	0,923	0,348	0,347	0,302	0,504	0,397	0,37
	YOLOv7mod-COCO	0,803	0,443	0,402	0,364	0,566	0,485	0,463
	YOLOv7mod-SAM	0,363	0,965	0,56	0,403	0,612	0,735	0,827

modificado logra un mejor equilibrio entre *precision* y *recall* en comparación con el *baseline*.

Finalmente, el modelo YOLOv7 modificado y entrenado con COCO etiquetado con SAM se destaca por lograr un alto *recall* de 0.854, aunque experimenta una disminución en la precisión comparado con los modelos anteriores. Este elevado *recall* sugiere que el modelo es eficaz en detectar la mayoría de los objetos en la imagen, pero esto conlleva un aumento en los falsos positivos, reflejado en una *precision* de 0.315. Los valores significativamente más altos en las métricas de *AP* y *F-scores*, particularmente en *F3-score*, indican que este modelo es el más adecuado para situaciones como las presentadas en el escenario *checkout* de RPC.

DSLL

En la Tabla 5.4 se muestran los resultados obtenidos por los tres modelos en diferentes escenarios del conjunto de datos DSLL. Al igual que en el conjunto de validación RPC, se aprecian diferencias notables entre los modelos en términos de *precision*, *recall*, *average precision* para los umbrales de .5 y .5:.95, así como en las métricas compuestas *F-scores*.

En los escenarios M1, M2, M3 y M4, el modelo YOLOv7 original entrenado con COCO muestra generalmente una alta *precision* pero un *recall* bajo, indicando que, aunque la red es eficiente en detectar los objetos, tiende a omitir muchos de ellos. Los valores de AP y los *F-scores* son regulares, con un AP@5 promedio de 0.347 y un AP@.5:.95 de 0.302, lo cual se considera relativamente bajo para un buen detector.

Por otro lado, el modelo YOLOv7 modificado con COCO muestra un mejor *recall* en comparación con el modelo original, lo que implica una mayor capacidad para detectar objetos. A diferencia de la evaluación con RPC, esta mejora en el *recall* no conlleva necesariamente una disminución en la *precision* en todos los escenarios, ya que en M1 y M2 se observa un aumen-

to en esta última. En promedio, este modelo logra mejores métricas de AP y *F1-scores* que el *baseline*, destacando especialmente en los incrementos de *F2-score* y *F3-score* de 0.088 y 0.093, respectivamente. Esto sugiere que este modelo se adapta mejor a los objetivos definidos que la versión original de YOLOv7 con COCO.

El modelo YOLOv7 modificado con SAM, por su parte, demuestra un incremento significativo en el *recall* en todos los escenarios, alcanzando un promedio sobresaliente de 0.965. Sin embargo, este modelo muestra una pérdida más marcada en la *precision*, con un promedio de 0.363. Respecto a los valores de *average precision*, este modelo supera a los dos anteriores, indicando un mejor rendimiento en la detección general de objetos. En cuanto a los *F-scores*, este modelo alcanza los mejores valores en las tres métricas, resaltando en *F2-score* y *F3-score*, con un promedio de 0.735 y 0.827 en los diferentes escenarios, respectivamente.

5.1.4. Selección del mejor modelo

Teniendo en cuenta que el sistema completo incorpora una etapa de reconocimiento y eliminación de detecciones redundantes, las cuales permiten descartar objetos que corresponden a instancias desconocidas o que resultan ser detecciones falsas o repetidas, el criterio principal para seleccionar el mejor detector de objetos genéricos se centra en su capacidad de capturar la mayor cantidad de objetos presentes en la imagen. Por ende, se busca un detector con un alto nivel de *recall*.

En este contexto, el modelo más adecuado es aquel que logre el mejor *F3-score*, que valora más el *recall* que la *precision*. El YOLOv7 modificado con SAM destaca en este aspecto, obteniendo un *F3-score* de 0.729 en RPC y 0.827 en DSLL. Además, este modelo sobresale en comparación con los otros en el resto de las métricas de evaluación de detección, incluyendo *AP@0.5*, *AP@.5:.95*, *F1-score* y *F2-scores*.

5.2. Módulo de extracción de descriptores

En esta sección, se exponen los resultados obtenidos de los entrenamientos realizados al módulo extractor de descriptores, empleando distintas arquitecturas de CNN y conexiones al backbone del detector de objetos genéricos (Stage 1 y Stage 2). Se mostrarán gráficos que reflejan la evolución del *loss* durante el entrenamiento y las métricas de evaluación en el conjunto de validación, junto con detalles de la mejor época alcanzada en cada configuración. A continuación, se evaluarán los resultados de las diversas arquitecturas en el conjunto de datos DSLL. Esta subsección incluirá estudios y comparaciones importantes para determinar la configuración óptima y así proceder a la comparación final entre las arquitecturas examinadas.

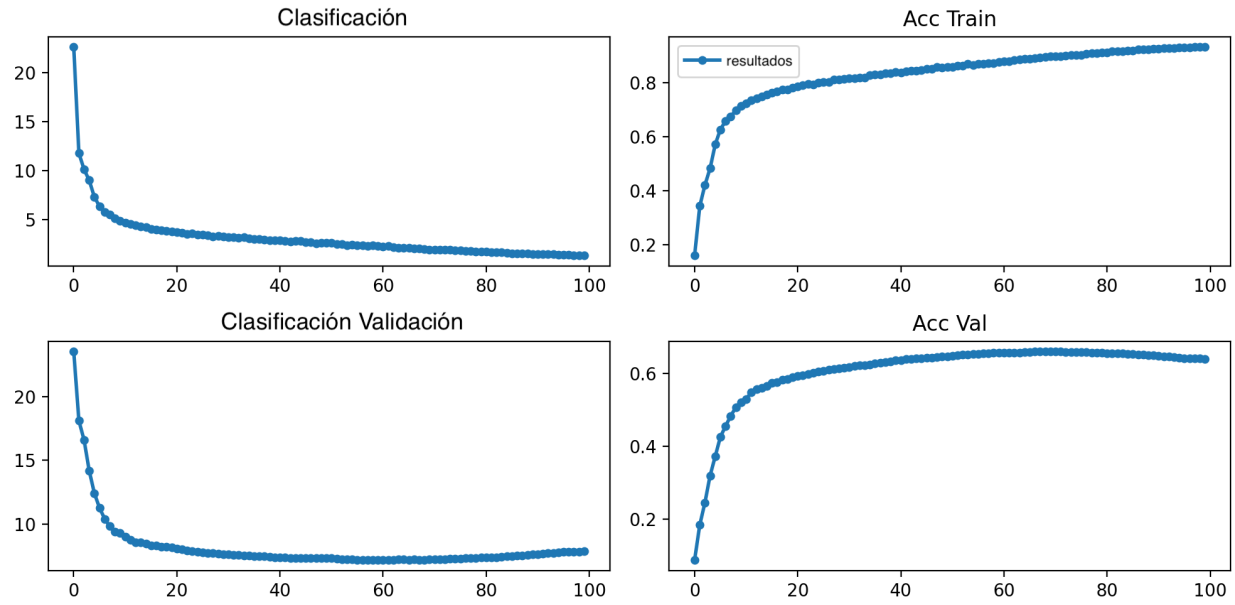


Figura 5.5: Evolución de las pérdidas y las métricas de rendimiento durante el entrenamiento de JPNet conectado al *stage 1*.

5.2.1. Entrenamiento con JPNet

En esta subsección, se detallan los resultados obtenidos durante el entrenamiento del módulo extractor de descriptores utilizando JPNet como arquitectura de CNN. Los resultados se analizan tanto para la conexión en el Stage 1 como en el Stage 2 del backbone de la red detectora de objetos genéricos. Se presentarán gráficos que ilustran la evolución del *loss* de entrenamiento y las métricas de evaluación en el conjunto de validación, proporcionando una visión completa del rendimiento del módulo en distintas etapas del proceso de entrenamiento.

Stage 1

La figura 5.5 muestra la evolución de las métricas de entrenamiento al integrar JPNet en el stage 1 del backbone. Se aprecia una disminución constante en el *loss* de clasificación de entrenamiento (Clasificación) a lo largo de las épocas. Por otro lado, el *loss* de clasificación de validación muestra un patrón similar inicialmente, pero a partir de la época 78 empieza a incrementarse, lo que podría ser un indicativo de un posible sobreajuste en el extractor. Este fenómeno se ve reforzado al examinar las curvas de *accuracy* de clasificación en el entrenamiento y en la validación. En la primera, se observa una tendencia ascendente que se estabiliza en las últimas épocas, mientras que en la segunda, especialmente en las últimas etapas, se nota una disminución en el *accuracy* conforme avanza el entrenamiento.

Los mejores valores de *accuracy* de validación se presentan en la tabla 5.5, donde se destaca que en la época 71 el modelo logra un *accuracy* de validación de 0.666. La discrepancia entre el *accuracy* de entrenamiento y la de validación se debe a las diferencias entre ambos conjuntos; uno está enfocado en la clasificación y el otro en la detección.

Tabla 5.5: Resultados obtenidos de la mejor época de entrenamiento de JPNet conectado al stage 1.

Época	Clasificación	Accuracy entrenamiento	Accuracy validación
71	1.9	0.898	0.660

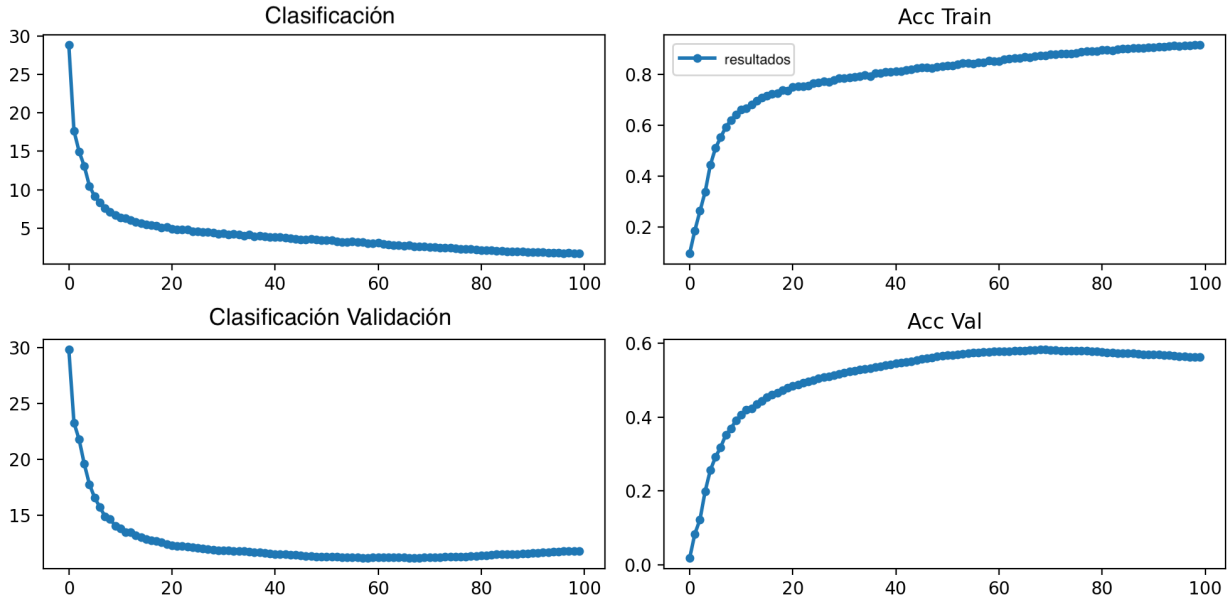


Figura 5.6: Evolución de las pérdidas y las métricas de rendimiento durante el entrenamiento de JPNet conectado al *stage 2*.

Stage 2

Respecto al entrenamiento con JPNet conectado al stage 2, los resultados de su evolución se pueden visualizar en la figura 5.6. Se observa un comportamiento similar tanto en los *loss* como en los *accuracy*. No obstante, en la tabla 5.6 se aprecia un *accuracy* inferior en la mejor época de entrenamiento del modelo: un *accuracy* de 0.874 en el conjunto de entrenamiento, en comparación con el 0.898 alcanzado con la conexión al stage 1. Además, en la tabla 5.6 se presenta una disminución aún más notable en el conjunto de validación, donde se logra un *accuracy* de 0.583 frente al 0.660 obtenido con el stage 1. Esta reducción sugiere que la arquitectura se adapta mejor al mapa de características del stage 1, debido a su mayor resolución y a la información más detallada que esto proporciona.

Tabla 5.6: Resultados obtenidos de la mejor época de entrenamiento de JPNet conectado al stage 2.

Época	Clasificación	Accuracy entrenamiento	Accuracy validación
68	2.6	0.874	0.583

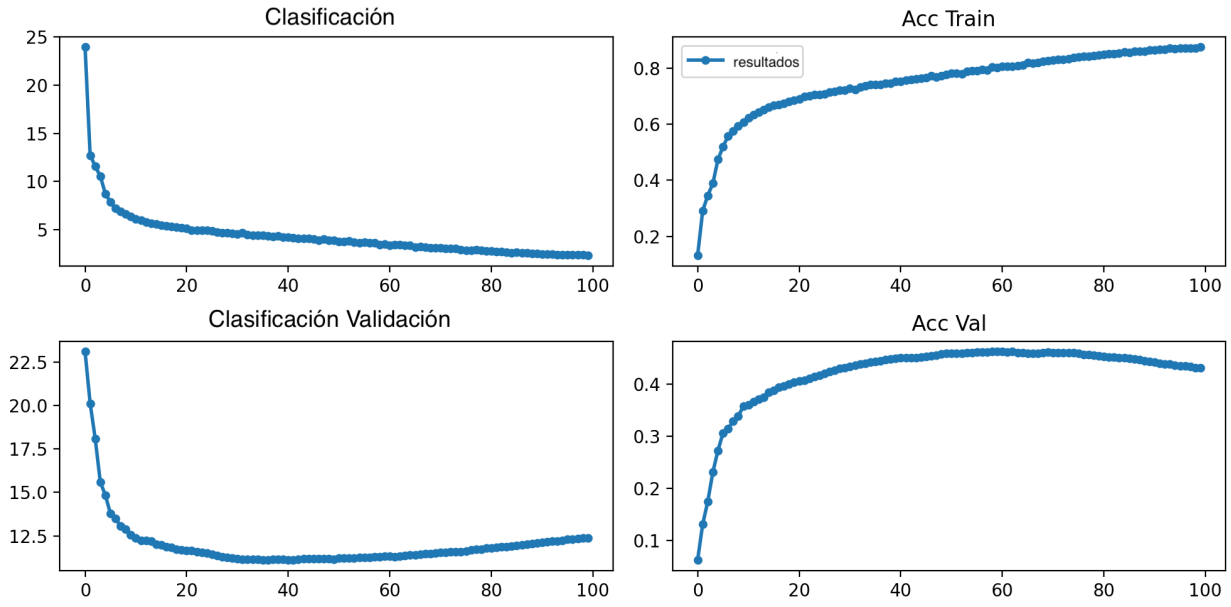


Figura 5.7: Evolución de las pérdidas y las métricas de rendimiento durante el entrenamiento de VoVNet conectado al *stage 1*.

Tabla 5.7: Resultados obtenidos de la mejor época de entrenamiento de VoVNet conectado al stage 1.

Época	**Clasificación**	**Accuracy entrenamiento**	**Accuracy validación**
62	3.455	0.806	0.462

5.2.2. Entrenamiento con módulo VoVNet

Esta subsección aborda los resultados obtenidos al entrenar el módulo extractor de descriptores usando la arquitectura basada en la red del estado del arte VoVNet como CNN. Se examinan los resultados para ambos stages, el 1 y el 2.

Stage 1

La figura 5.7 ilustra la evolución del entrenamiento al integrar módulos de VoVNet en el stage 1 del backbone. Aquí se puede ver cómo el *loss* de clasificación disminuye a lo largo del entrenamiento. Sin embargo, el *loss* de validación solo disminuye hasta la época 50, mostrando un sobreajuste más evidente del modelo después de la época 60 en comparación con los modelos entrenados con JPNet. Este fenómeno se ve reflejado en el *accuracy* de validación, que, según la tabla 5.7, alcanza su mejor valor en la época 62 con un *accuracy* de validación de 0,462.

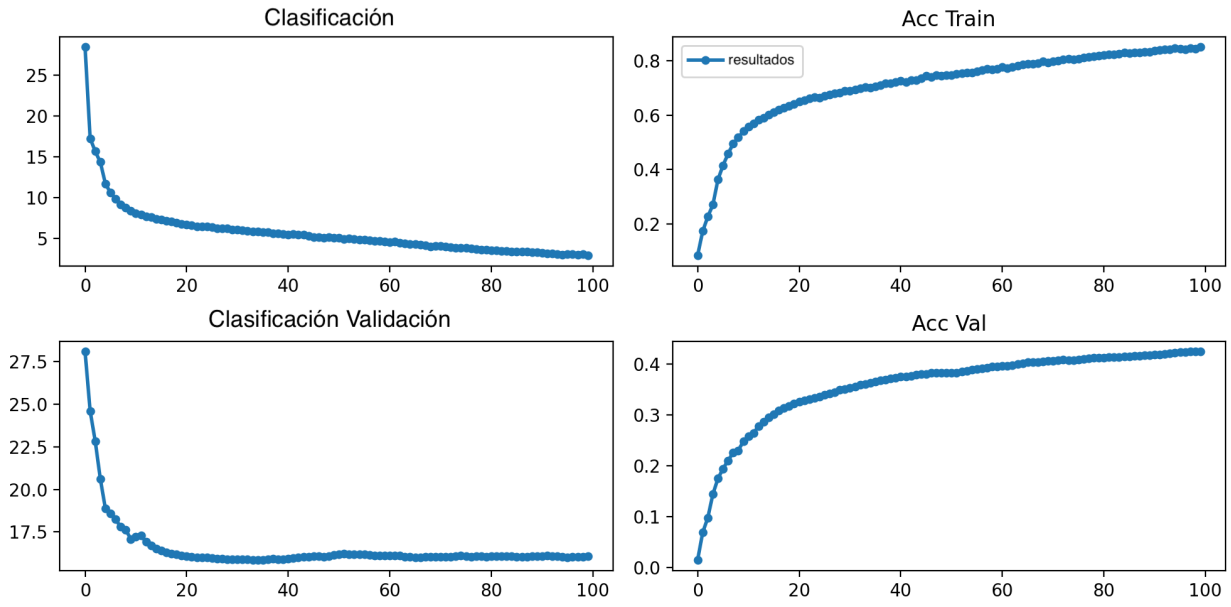


Figura 5.8: Evolución de las pérdidas y las métricas de rendimiento durante el entrenamiento de VoVNet conectado al *stage 2*.

Tabla 5.8: Resultados obtenidos de la mejor época de entrenamiento de VoVNet conectado al stage 2.

Época	**Clasificación**	**Accuracy entrenamiento**	**Accuracy validación**
99	2.907	0.851	0.425

Stage 2

En la figura 5.8, se observa una tendencia similar en la disminución del *loss* de clasificación de entrenamiento. A diferencia del stage 1, aquí no se detecta un patrón de sobreajuste, dado que el *loss* de validación no aumenta a lo largo del entrenamiento. La precisión en el conjunto de validación, mostrada en la tabla 5.8, llega a su punto más alto en la época 100 con un valor de 0.425. Este resultado, siendo inferior al máximo alcanzado en el stage 1, indica que la conexión en el stage 2 resulta menos efectiva para la extracción de descriptores en comparación con el stage 1.

5.2.3. Entrenamiento con Baseline

En esta subsección, se abordan los resultados y el análisis del entrenamiento del módulo extractor sin emplear una CNN. Esta configuración de *baseline* actúa como referencia para evaluar las ventajas de incorporar una capa CNN en el aprendizaje de un mapa de características especializado para la extracción de descriptores. Al igual que con los módulos anteriores, se lleva a cabo el entrenamiento tanto en el stage 1 como en el stage 2. Es importante señalar que, debido a un sobreajuste temprano en ambos entrenamientos, se limitó la cantidad de épocas a las iniciales para facilitar el análisis.

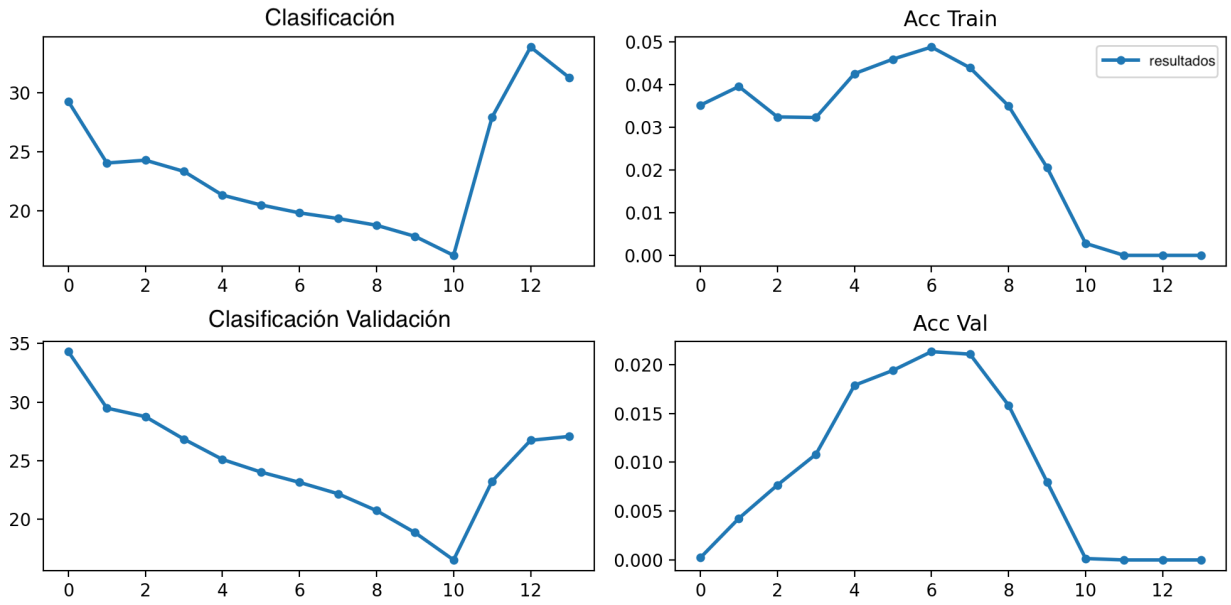


Figura 5.9: Evolución de las pérdidas y las métricas de rendimiento durante el entrenamiento de baseline conectado al *stage 1*.

Tabla 5.9: Resultados obtenidos de la mejor época de entrenamiento de *baseline* conectado al stage 1.

Época	**Clasificación**	**Accuracy entrenamiento**	**Accuracy validación**
7	19.81	0.049	0.021

Stage 1

La figura 5.9 muestra la evolución del modelo *baseline* al estar conectado al *Stage 1* del backbone, exhibiendo curvas de aprendizaje limitadas. Aunque las curvas de *loss* de entrenamiento y validación disminuyen adecuadamente, a partir de la época 10 el *loss* comienza a aumentar significativamente. Las curvas de *accuracy* muestran un comportamiento aceptable durante las primeras 6 épocas, pero al igual que el *loss*, desde la época 7 el *accuracy* experimenta una caída notable. Este fenómeno podría deberse a la ausencia de capas convolucionales entrenables que puedan adaptarse mejor a la tarea de clasificación de objetos. Cabe mencionar que los únicos parámetros entrenables en el *baseline* corresponden a las capas *fully connected* y de normalización por lotes (*batch normalization*) de la capa *Neck*.

Este comportamiento refleja la necesidad de una arquitectura que pueda ajustarse a la variabilidad en las características de las instancias de objetos en las imágenes. La tabla 5.9 detalla las métricas de la mejor época, que corresponde a la época 7, con un *loss* de clasificación de 19.81, un *accuracy* de entrenamiento de solo 0.049, y un *accuracy* de validación aun más bajo de 0.021.

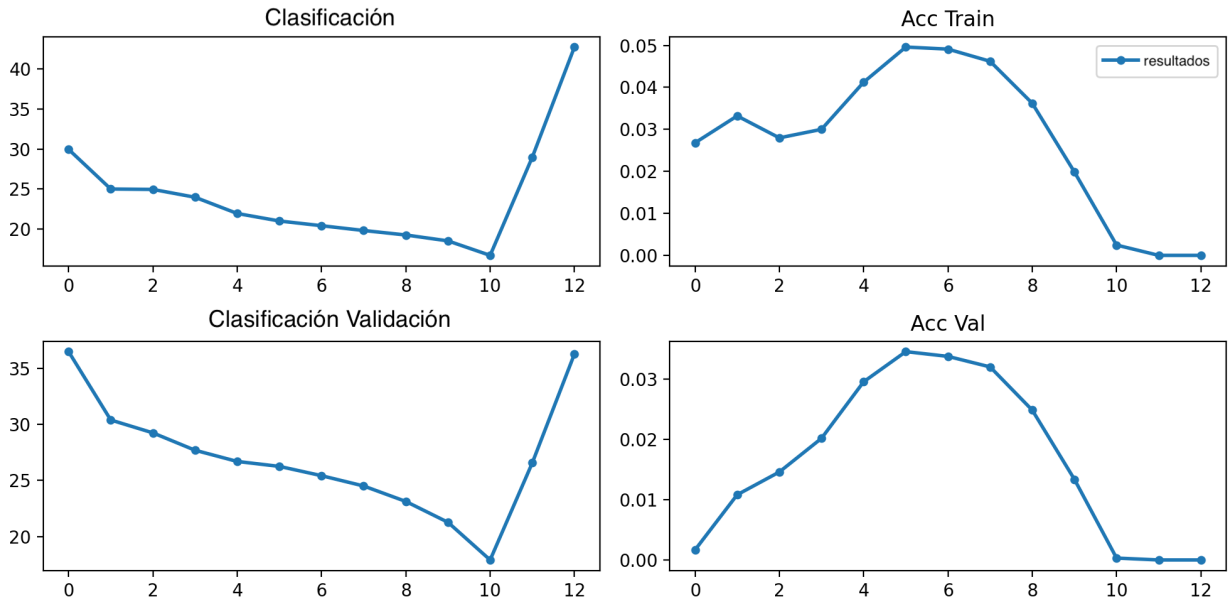


Figura 5.10: Evolución de las pérdidas y las métricas de rendimiento durante el entrenamiento de baseline conectado al *stage 2*.

Tabla 5.10: Resultados obtenidos de la mejor época de entrenamiento de *baseline* conectado al stage 2.

Época	**Clasificación**	**Accuracy entrenamiento**	**Accuracy validación**
6	20.9	0.05	0.035

Stage 2

El enfoque *baseline* conectado al *Stage 2* del backbone tiene la misma tendencias de curvas que el conectado al stage 1. En la figura 5.10 se muestra la evolución del loss y las métricas a medida que se realiza el entrenamiento.

La tabla 5.10 presenta los resultados de la mejor época, que corresponde a la época 6, con un *loss* de clasificación alto de 20.9. Los valores de accuracy para el entrenamiento y la validación, 0.05 y 0.035 respectivamente, son la prueba de que la ausencia de una CNN compleja impide que el modelo capture las características necesarias para una clasificación precisa.

5.2.4. Evaluación extractor de descriptores

Esta subsección está enfocada en la evaluación de los modelos de extracción de descriptores que fueron desarrollados y entrenados previamente. Para llevar a cabo esta evaluación, cada modelo se evalúa con distintas configuraciones en los escenarios seleccionados del conjunto de evaluación DSLL. Inicialmente, se abordan dos casos de estudio: el primero compara el impacto de usar 1 frente a 4 imágenes de referencia y el segundo examina la influencia de la

Tabla 5.11: Comparación de resultados entre usar 1 imagen de referencia vs 4 imágenes de referencia con JPNet Stage 1 y JPNet Stage 2.

Set	Modelo	N° imágenes	P	R	F1-score
S1	JPNet Stage 1	1	0,977	0,995	0,986
	JPNet Stage 2	1	0,98	0,987	0,983
	JPNet Stage 1	4	0,995	0,995	0,995
	JPNet Stage 2	4	0,995	0,995	0,995
M1	JPNet Stage 1	1	0,762	0,71	0,735
	JPNet Stage 2	1	0,724	0,694	0,709
	JPNet Stage 1	4	0,756	0,729	0,742
	JPNet Stage 2	4	0,736	0,693	0,714
M2	JPNet Stage 1	1	0,671	0,36	0,469
	JPNet Stage 2	1	0,654	0,532	0,587
	JPNet Stage 1	4	0,62	0,461	0,529
	JPNet Stage 2	4	0,625	0,632	0,628
M3	JPNet Stage 1	1	0,914	0,91	0,912
	JPNet Stage 2	1	0,939	0,926	0,932
	JPNet Stage 1	4	0,939	0,934	0,936
	JPNet Stage 2	4	0,945	0,938	0,941
M4	JPNet Stage 1	1	0,863	0,833	0,848
	JPNet Stage 2	1	0,843	0,814	0,828
	JPNet Stage 1	4	0,86	0,843	0,851
	JPNet Stage 2	4	0,865	0,829	0,847
MEAN	JPNet Stage 1	1	0,837	0,762	0,79
	JPNet Stage 2	1	0,828	0,791	0,808
	JPNet Stage 1	4	0,834	0,792	0,811
	JPNet Stage 2	4	0,833	0,817	0,825

capa *neck* en el proceso de inferencia. Finalmente, se realiza una comparación de los modelos, tomando en cuenta las conclusiones derivadas de los estudios anteriores. El objetivo de esta comparativa es analizar y seleccionar el modelo más eficaz para la extracción de descriptores de instancias de objetos.

Estudio 1: una imagen vs cuatro imágenes de referencia

Este primer estudio se enfoca en evaluar el impacto de utilizar diferentes cantidades de imágenes de referencia en el desempeño del modelo a través de los escenarios. Se compararon los resultados de usar una única imagen de referencia frente a cuatro imágenes de referencia. Para ello, se utilizó el escenario S1, que contiene distintas vistas de 40 instancias de objetos. En el caso de una imagen, se seleccionó una vista del objeto de frente, mientras que para el caso de cuatro imágenes se usaron todas las vistas disponibles por objeto. Los resultados de estos experimentos en cada escenario se muestran en la tabla 5.11.

En el conjunto S1, se observa que tener cuatro imágenes de referencia mejora el *precision*

y el *recall* en comparación con el uso de una sola imagen, por ende se tiene un mejor *F1-score*. Esto puede deberse a que, como el escenario S1 es el mismo utilizado como referencia, el promedio de todas las vistas por objeto resulta más representativo que una sola vista para el reconocimiento.

Para los conjuntos M1, M2, M3 y M4, donde las condiciones ambientales varían en términos de número de objetos, iluminación o fondo, los resultados de *precision* fluctúan tanto positiva como negativamente entre usar una y cuatro imágenes. Sin embargo, el *recall* en todos los escenarios se ve beneficiado. Aunque la *precision* disminuye en ciertos escenarios, el aumento en *recall* siempre compensa esta caída, como se refleja en el *F1-score*, donde en todos los escenarios el uso de cuatro imágenes supera al uso de una tanto para JPNet Stage 1 como JPNet Stage 2. Un ejemplo de esto es el escenario M2 para JPNet Stage 1, donde la *precision* disminuye en 0.051 pero el *recall* aumenta en 0.101, resultando en un incremento del *F1-score* de 0.06.

En general, el rendimiento de ambos enfoques se refleja en las métricas promedio (MEAN), donde la *precision* disminuye ligeramente en 0.003 para el Stage 1 y aumenta en 0.005 para el Stage 2; el *recall* aumenta en 0.03 para el Stage 1 y en 0.026 para el Stage 2, lo que lleva a un aumento en el *F1-score* de 0.021 y 0.017 respectivamente. Esta mejora general se debe a que considerar el promedio de varias vistas en lugar de una única ofrece una representación estadísticamente más robusta, capturando una mayor variabilidad de los objetos, ayudando a mitigar características atípicas y suavizando el ruido que puede presentar una sola imagen de referencia.

Estudio 2: inferencia con y sin capa neck

Este estudio analiza la diferencia en el rendimiento de inferencia de los modelos de extracción de descriptores con y sin la capa *neck* en distintos escenarios de DSLL. Se utilizó el modelo JPNet en los stages 1 y 2, evaluando su rendimiento en las métricas de *precision*, *recall* y *F1-score*. Los resultados se presentan en la tabla 5.12.

En el escenario S1, no se detectaron diferencias al incluir o excluir la capa *neck* en los modelos JPNet de los stages 1 y 2. En M1, todas las métricas mejoraron al no utilizar la capa *neck* en la inferencia tanto para JPNet Stage 1, donde el *F1-score* aumentó en 0.05, como para JPNet Stage 2 con un aumento de *F1-score* de 0.047. En M2, la eliminación del *neck* en la inferencia resultó en un notable aumento del *F1-score*, incrementándose en 0.144 para JPNet Stage 1 y 0.086 para JPNet Stage 2. En M3, no se observó un patrón similar en el *F1-score*; para JPNet Stage 1, se redujo ligeramente en 0.027 al inferir sin *neck* y para JPNet Stage 2, en ausencia de *neck*, el *F1-score* aumentó levemente en 0.011. En M4, el rendimiento en *F1-score* disminuyó ligeramente al inferir sin *neck* en ambos stages. Finalmente, al promediar los rendimientos en cada escenario, se observa que, en general, el rendimiento al inferir sin la capa *neck* aumenta para ambos enfoques.

Por lo tanto, se puede concluir que la capa *neck* en la inferencia tiene un impacto variable en el rendimiento dependiendo del contexto de uso. En escenarios donde los objetos tienen mayor similitud con los de referencia, como S1, M3 (mismo fondo e iluminación) y M4 (mismo fondo e iluminación), el uso del *neck* no ofrece ventajas significativas. Sin embargo, en

Tabla 5.12: Comparación de resultados de inferencia con y sin neck.

Set	**Modelo**	**Neck**	**P**	**R**	**F1-score**
S1	JPNet Stage 1	SI	0,995	0,995	0,995
	JPNet Stage 1	NO	0,995	0,995	0,995
	JPNet Stage 2	SI	0,995	0,995	0,995
	JPNet Stage 2	NO	0,995	0,995	0,995
M1	JPNet Stage 1	SI	0,697	0,687	0,692
	JPNet Stage 1	NO	0,756	0,729	0,742
	JPNet Stage 2	SI	0,699	0,638	0,667
	JPNet Stage 2	NO	0,736	0,693	0,714
M2	JPNet Stage 1	SI	0,4	0,371	0,385
	JPNet Stage 1	NO	0,62	0,461	0,529
	JPNet Stage 2	SI	0,536	0,548	0,542
	JPNet Stage 2	NO	0,625	0,632	0,628
M3	JPNet Stage 1	SI	0,96	0,96	0,96
	JPNet Stage 1	NO	0,94	0,934	0,937
	JPNet Stage 2	SI	0,933	0,927	0,93
	JPNet Stage 2	NO	0,945	0,938	0,941
M4	JPNet Stage 1	SI	0,856	0,854	0,855
	JPNet Stage 1	NO	0,86	0,843	0,851
	JPNet Stage 2	SI	0,866	0,843	0,854
	JPNet Stage 2	NO	0,865	0,829	0,847
MEAN	JPNet Stage 1	SI	0,782	0,773	0,777
	JPNet Stage 1	NO	0,834	0,792	0,811
	JPNet Stage 2	SI	0,806	0,79	0,798
	JPNet Stage 2	NO	0,833	0,817	0,825

escenarios con mayor variabilidad, como M1 (fondo colorido diferente) y M2 (varios fondos), la omisión del *neck* mejora el rendimiento del modelo en el reconocimiento. Esto podría deberse a que la capa FC y de Batch Normalization presentes en el *neck* pueden estar sobreajustadas a las características del conjunto de entrenamiento, que no incluía cambios de contexto tan drásticos como los del conjunto de esta evaluación. Además, la capa FC, al estar diseñada para aprender una representación global de los datos, puede otorgar relevancia a partes del área de detección que no contienen el objeto detectado.

Comparación final entre modelos

A partir de los estudios anteriores, se concluye que para lograr un rendimiento óptimo, los modelos deben utilizar como descriptores el promedio de 4 imágenes por objeto y omitir la capa *neck* durante la inferencia, especialmente en escenarios con alta variabilidad. El objetivo de esta sección es evaluar y comparar el rendimiento de todos los extractores de descriptores entrenados en los escenarios de DSLL. Los resultados obtenidos se presentan en la tabla 5.13.

En el escenario S1, los modelos muestran un rendimiento similarmente alto. Tanto los modelos JPNet como el *baseline* alcanzan la misma medida de *F1-score*. VoVNet presenta una pequeña diferencia en su *F1-score*, donde con el stage 1 alcanza un valor perfecto de 1, mientras que con el stage 2 llega a 0.966. Estos valores casi ideales se justifican debido a que S1 es el mismo conjunto del cual se toman las imágenes de referencia, minimizando la variabilidad.

En los conjuntos M1 y M2, donde hay mayor variabilidad en términos de fondos y número de objetos, el rendimiento disminuye considerablemente. Por ejemplo, en M1, JPNet Stage 1 cae a 0.742, mientras que en M2, desciende aún más a 0.528. Estas caídas reflejan una sensibilidad del sistema a los cambios de fondo y ambiente.

En los escenarios M3 y M4, con menor variabilidad que M1 y M2, se observa un mejor rendimiento. Por ejemplo, JPNet Stage 1 alcanza un *F1-score* de 0.936 en M3 y de 0.852 en M4. El rendimiento en M4 es menor que en M3 debido al cambio en la iluminación, lo que sugiere ciertas dificultades en el reconocimiento de instancias de objetos, aunque estos cambios afectan en menor medida que los cambios de fondo.

En los escenarios M1, M2, M3 y M4, se observan variaciones en el rendimiento entre los diferentes modelos. En M1, JPNet Stage 1 destaca con un *F1-score* de 0.742. En M2, el sistema más eficaz es el *baseline* stage 2, con un *F1-score* de 0.7. En M3, VoVNet stage 1 sobresale con un impresionante *F1-score* de 0.945, y en M4, el mejor rendimiento lo tiene VoVNet stage 2 con un *F1-score* de 0.862.

Al promediar el rendimiento en todos los escenarios, JPNet Stage 2 emerge como el modelo con mejor *F1-score* general, con un valor de 0.825, seguido de cerca por JPNet Stage 1 con 0.811. La superioridad de JPNet, a pesar de su arquitectura más simple en comparación con VoVNet, se debe a su diseño específico para el problema en cuestión. JPNet, con sus 5 capas convolucionales densas y sin realizar reducción de escala en los mapas de características, logra preservar la información espacial y mejorar el nivel semántico de los descriptores.

Tabla 5.13: Comparación de resultados con diferentes configuraciones y escenarios DSLL.

Set	**Modelo**	**P**	**R**	**F1-score**
S1	JPNet Stage 1	0,995	0,995	0,995
	JPNet Stage 2	0,995	0,995	0,995
	VoVnet Stage 1	1	1	1
	VoVnet Stage 2	0,969	0,964	0,966
	Baseline Stage 1	0,995	0,995	0,995
	Baseline Stage 2	0,995	0,995	0,995
M1	JPNet Stage 1	0,756	0,729	0,742
	JPNet Stage 2	0,736	0,693	0,714
	VoVnet Stage 1	0,712	0,691	0,701
	VoVnet Stage 2	0,63	0,622	0,626
	Baseline Stage 1	0,542	0,596	0,568
	Baseline Stage 2	0,709	0,679	0,694
M2	JPNet Stage 1	0,62	0,461	0,529
	JPNet Stage 2	0,625	0,632	0,628
	VoVnet Stage 1	0,552	0,537	0,544
	VoVnet Stage 2	0,535	0,534	0,534
	Baseline Stage 1	0,503	0,556	0,528
	Baseline Stage 2	0,72	0,681	0,7
M3	JPNet Stage 1	0,939	0,934	0,936
	JPNet Stage 2	0,945	0,938	0,941
	VoVnet Stage 1	0,948	0,942	0,945
	VoVnet Stage 2	0,811	0,811	0,811
	Baseline Stage 1	0,832	0,82	0,826
	Baseline Stage 2	0,811	0,738	0,773
M4	JPNet Stage 1	0,86	0,844	0,852
	JPNet Stage 2	0,865	0,829	0,847
	VoVnet Stage 1	0,876	0,848	0,862
	VoVnet Stage 2	0,7	0,708	0,704
	Baseline Stage 1	0,66	0,605	0,631
	Baseline Stage 2	0,742	0,666	0,702
MEAN	JPNet Stage 1	0,834	0,793	0,811
	JPNet Stage 2	0,833	0,817	0,825
	VoVnet Stage 1	0,818	0,804	0,81
	VoVnet Stage 2	0,729	0,728	0,728
	Baseline Stage 1	0,706	0,714	0,71
	Baseline Stage 2	0,795	0,766	0,773

VoVNet, a pesar de su eficiencia energética y optimización en el entrenamiento y ser la base de los módulos E-ELAN de YOLOv7, no logra superar a JPNet. Esto se debe a que los stages de VoVNet seleccionados para el modelo involucran reducción de escala, lo que conlleva a la pérdida de información espacial y reduce la exactitud de la capa RoI Align al seleccionar regiones de interés.

Por otro lado, la CNN basada en VoVnet a pesar de ser una arquitectura más eficiente en términos de energía y optimización de los gradientes en el entrenamiento y de tener mejor acople a los módulos E-ELAN del backbone de YOLOv7 modificado, que están basados en esta misma red, no logra superar a JPNet, y esto es debido a que el modelo considerado, solo corresponde a 2 stage del modelo completo, debido a que las operaciones que realizan los stages de VoVNet contienen downsampling, y a medida que se avanza en sus stages la resolución se va disminuyendo a la mitad, significando pérdidas de información espacial y menor exactitud de la capa RoI Align al momento de seleccionar la región de interés.

Sorprendentemente, el enfoque *baseline* demuestra ser competitivo, a pesar de no contar con una CNN para un mapa de características especializado. El *baseline* stage 2 incluso supera a VoVNet stage 2 y al *baseline* stage 1. Esto podría explicarse por el mayor nivel semántico de los features en el stage 2, que, aunque menos especializados, son más efectivos en la generación de descriptores.

Aunque JPNet Stage 2 muestra los mejores resultados, también se considerará JPNet Stage 1 para la evaluación final del sistema completo, dado que podría ser menos susceptible a problemas de localización del detector de objetos genéricos, lo que podría influir en un rendimiento más equilibrado en términos de métricas.

5.3. Detector de instancias de objetos

Esta sección se dedica a la presentación de los resultados finales del sistema completo seleccionado, que combina el detector de objetos genéricos con el módulo extractor de descriptores, así como con el módulo de reconocimiento y eliminación de detecciones redundantes. La sección se enfocará en evaluar el rendimiento del sistema en términos de precisión y velocidad de inferencia.

Para establecer un punto de comparación, los resultados del mejor modelo obtenido en la evaluación comparativa entre JPNet Stage 1 y Stage 2, se contrastarán con los de YOLOSPoC. Esta comparación abordará tanto la precisión como la velocidad de inferencia, proporcionando un análisis detallado y comparativo del desempeño de ambos sistemas.

Por último, se realizará un análisis visual de las detecciones de instancias de objetos obtenidas en los distintos escenarios del conjunto de datos DSLL. Este análisis visual tiene como objetivo ofrecer una comprensión más clara de las fortalezas y limitaciones del sistema propuesto, resaltando las áreas donde se destaca y aquellas en las que enfrenta desafíos.

Tabla 5.14: Resultados obtenidos de YOLOv7 modificado con JPNet Stage 1 y JPNet Stage 2.

Set	Modelo	P	R	F1-score
S1	YOLOv7mod & JPNet Stage 1	0,936	0,994	0,964
	YOLOv7mod & JPNet Stage 2	0,787	0,988	0,876
S2	YOLOv7mod & JPNet Stage 1	0,853	0,86	0,857
	YOLOv7mod & JPNet Stage 2	0,733	0,846	0,785
S3	YOLOv7mod & JPNet Stage 1	0,773	0,75	0,761
	YOLOv7mod & JPNet Stage 2	0,728	0,729	0,728
S4	YOLOv7mod & JPNet Stage 1	0,751	0,75	0,751
	YOLOv7mod & JPNet Stage 2	0,644	0,63	0,637
M1	YOLOv7mod & JPNet Stage 1	0,687	0,484	0,568
	YOLOv7mod & JPNet Stage 2	0,489	0,356	0,412
M2	YOLOv7mod & JPNet Stage 1	0,743	0,39	0,512
	YOLOv7mod & JPNet Stage 2	0,452	0,372	0,408
M3	YOLOv7mod & JPNet Stage 1	0,865	0,785	0,823
	YOLOv7mod & JPNet Stage 2	0,71	0,829	0,765
M4	YOLOv7mod & JPNet Stage 1	0,802	0,632	0,707
	YOLOv7mod & JPNet Stage 2	0,691	0,627	0,657
MEAN	YOLOv7mod & JPNet Stage 1	0,732	0,708	0,742
	YOLOv7mod & JPNet Stage 2	0,654	0,672	0,659

5.3.1. YOLOv7 modificado con JPNet Stage 1 y JPNet Stage 2

Evaluación de métricas de precisión

La tabla 5.14 muestra una comparación detallada de los mejores resultados obtenidos por el sistema completo, que integra las redes YOLOv7 modificadas con JPNet en los Stage 1 y Stage 2. Es importante destacar que, aunque el rendimiento de los modelos se evaluó en términos de *recall*, *precision* y *F1-score*, la configuración de hiper-parámetros se orientó principalmente a optimizar el *F1-score* promedio. Para JPNet Stage 1, se encontró que el valor umbral óptimo para el reconocimiento de descriptores es de 0.37, mientras que para JPNet Stage 2, el umbral óptimo se estableció en 0.5.

En los escenarios S1 a S4, que presentan una instancia de objeto por imagen, se observa que YOLOv7 modificado con JPNet Stage 1, en general, muestra un rendimiento superior al Stage 2. Esta superioridad es particularmente evidente en el escenario S1, donde el Stage 1 alcanza un *F1-score* de 0.964, frente a 0.876 del Stage 2. Este resultado contrasta con los obtenidos en la tabla 5.13, donde ambos modelos demostraban un rendimiento similar. Esto podría indicar que el Stage 2 es más sensible a errores en la localización de objetos en comparación con el Stage 1, dado que en este escenario las detecciones se basan directamente en los recuadros generados por el detector y no en las posiciones reales de los objetos.

En los escenarios M1 a M4, que representan escenarios más desafiantes con seis instancias de objetos, tanto para el Stage 1 como para el Stage 2 de YOLOv7 modificado con JPNet,

Tabla 5.15: Resultados de tiempos de inferencias obtenidos de YOLOv7 modificado con JPNet Stage 1 y Stage 2.

Resolución	Modelo	Tiempo inferencia [ms]	FPS
640 x 480	YOLOv7mod & JPNet Stage 1	31,25	32
	YOLOv7mod & JPNet Stage 2	22,73	44

se observa una disminución esperada en el rendimiento en comparación con los escenarios S1 a S4. Sin embargo, el Stage 1 continúa mostrando un rendimiento general superior.

Esta ventaja del Stage 1 se refleja en la métrica promedio del *F1-score*, donde alcanza un valor de 0.742, superando al 0.659 del Stage 2. Un *F1-score* promedio más alto indica que el modelo mantiene un equilibrio más consistente entre un *recall* y una *precision* más alta en todos los escenarios.

Que JPNet con Stage 1 supere en *precision* al Stage 2 en la evaluación del sistema completo, a pesar de que el Stage 2 tuviera un mejor rendimiento en la evaluación del extractor, sugiere que este último es más susceptible a errores de localización. Esta sensibilidad puede atribuirse a que el Stage 2 trabaja con un mapa de características de menor resolución (mitad de la resolución que el Stage 1), lo que puede amplificar los errores en la extracción de la región de interés y resultar en descriptores con mayor ruido del ambiente.

Evaluación tiempos de inferencia

Respecto a los tiempos de inferencia, presentados en la tabla 5.15, el sistema con JPNet Stage 2 muestra un tiempo de inferencia menor que el Stage 1 (22,73 ms frente a 31,25 ms), lo que se traduce en una mayor capacidad de procesamiento de imágenes por segundo (44 FPS frente a 32 FPS). Esta diferencia se debe a la menor resolución del mapa de características de entrada en la capa CNN del Stage 2 en comparación con el Stage 1, implicando un número menor de parámetros a calcular. Es importante destacar que ambos sistemas califican como sistemas en tiempo real, ya que funcionan a más de 30 FPS. Cabe mencionar que las pruebas de velocidad fueron realizadas con una tarjeta gráfica GeForce RTX3070.

Finalmente, se puede concluir que, aunque el sistema con JPNet Stage 2 muestra un rendimiento inferior en varios escenarios en comparación con JPNet Stage 1, tiene la ventaja de poseer un menor tiempo de inferencia. La elección del modelo más adecuado dependerá del caso de uso específico. Sin embargo, para los propósitos de este trabajo, y considerando que ambos modelos cumplen con los requisitos de ser sistemas en tiempo real, se opta por seleccionar la versión con JPNet Stage 1 para realizar la evaluación comparativa final frente a YOLOSPoC.

5.3.2. Comparación entre el sistema final y YOLOSPoC

En esta sección, se presenta una comparativa entre el sistema final propuesto, YOLOv7 modificado con JPNet Stage 1, y YOLOSPoC, un sistema consolidado de detección de instan-

Tabla 5.16: Resultados obtenidos de YOLOv7 modificado con JPNet Stage 1 y YOLOSPoC.

Set	Modelo	F1-score
S1	YOLOv7mod & JPNet Stage 1	0,964
	YOLOSPoC	0.861
S2	YOLOv7mod & JPNet Stage 1	0,857
	YOLOSPoC	0,84
S3	YOLOv7mod & JPNet Stage 1	0,761
	YOLOSPoC	0,902
S4	YOLOv7mod & JPNet Stage 1	0,751
	YOLOSPoC	0,923
M1	YOLOv7mod & JPNet Stage 1	0,568
	YOLOSPoC	0,863
M2	YOLOv7mod & JPNet Stage 1	0,512
	YOLOSPoC	0,814
MEAN	YOLOv7mod & JPNet Stage 1	0.736
	YOLOSPoC	0,843

cias de objetos que utiliza YOLOv3 y ResNet101 para la detección y extracción de descriptores, respectivamente. Esta comparación se centrará en evaluar la precisión y la velocidad de inferencia de ambos modelos.

Evaluación de métricas de precisión

Los *F1-scores* del sistema final y YOLOSPoC se detallan en la tabla 5.16. Al examinar los resultados de *F1-score* en cada escenario, se observa que YOLOv7 modificado con JPNet Stage 1 supera a YOLOSPoC en los escenarios S1 y S2, logrando un *F1-score* de 0.964 en comparación con 0.861 de YOLOSPoC y de 0.857 frente a 0.84 de YOLOSPoC, respectivamente. Sin embargo, en los escenarios S3, S4, M1 y M2, YOLOSPoC alcanza un mejor rendimiento.

El mejor desempeño del sistema final en los escenarios S1 y S2 se debe a que estos escenarios presentan una baja variabilidad en relación con las imágenes de referencia, donde la generación de descriptores funciona de manera más efectiva. Esto, combinado con una mayor detección de objetos que logra un detector de objetos genéricos en comparación con YOLOv3 entrenado solo con las clases etiquetadas de COCO, utilizado por YOLOSPoC, permite al sistema propuesto obtener un mejor *F1-score* que YOLOSPoC.

En los escenarios también con un único objeto, S3 (con un fondo colorido) y S4 con múltiples fondos, el rendimiento F1-score del sistema propuesto decae en un poco mas 0.2 de F1-score con respecto a S1, lo cual como se analizo en secciones anteriores, corresponde a la variacion de fondos que afectan sensiblemente el modulo generador de descriptores. Cosa que YOLOSPoC aborda de mejor manera debido a que ocupa una red ResNet101 dedicada a la extracción de descriptores, compensando la dificultad en la deteccion de objetos muy distinto a las clases etiquetadas de COCO.

En los escenarios también con un único objeto, S3 (con un fondo colorido) y S4 (con múltiples fondos), el *F1-score* del sistema propuesto disminuye en poco más de 0.2 en comparación con S1, lo cual, como se analizó en secciones anteriores, se debe a la variación de fondos que afecta significativamente el módulo generador de descriptores. Esta situación es mejor manejada por YOLOSPoC, que utiliza una red ResNet101 dedicada a la extracción de descriptores, compensando así su dificultad en la detección de objetos que son muy diferentes a las clases etiquetadas de COCO.

Esta sensibilidad se ve reforzada en los escenarios M1 y M2, que incluyen múltiples objetos por imagen, donde el sistema propuesto con JPNet Stage 1 obtiene un *F1-score* inferior al de YOLOSPoC. Estos escenarios son también los que YOLOSPoC presenta un menor rendimiento; sin embargo, su mejor capacidad para crear descriptores le permite obtener mejores resultados que el sistema YOLOv7 modificado con JPNet.

En términos de rendimiento promedio, YOLOSPoC demuestra ser más eficaz en general, con un *F1-score* medio de 0.843 en comparación con 0.736 de YOLOv7 modificado, marcando una diferencia aproximada de 0.1. Esto se debe a que YOLOSPoC utiliza una red de extracción de descriptores más especializada, separando las tareas de detección y cálculo de descriptores, lo que permite dedicar una CNN completa a procesar cada objeto detectado por YOLOv3, facilitando su entrenamiento y la generación de descriptores representativos. Por otro lado, YOLOv7 modificado con JPNet es un sistema integrado donde cada objeto detectado obtiene su descriptor de la misma red a través de una capa de RoI Align, en lugar de la imagen completa del objeto.

A pesar de estas diferencias, hay escenarios en los que el sistema propuesto iguala o incluso supera el rendimiento de YOLOSPoC, como en el escenario S1. Esto se debe principalmente a que este sistema utiliza un detector de objetos genéricos, que posee una mejor capacidad de detección que un detector de clases de objetos convencional con baja confidencia, como YOLOv3 o incluso YOLOv7, como se observó en la tabla 5.4.

Evaluación tiempos de inferencia

Los resultados obtenidos de la evaluación de los tiempos de inferencia se detallan en la tabla 5.17. YOLOv7 modificado con JPNet Stage 1 registra un tiempo de inferencia de 31,25 ms, alcanzando 32 FPS, lo que es bastante cercano a los 32,3 ms y 31 FPS de YOLOv3 de YOLOSPoC para imágenes de 640 x 480. Por otro lado, el extractor de descriptores ResNet 101, utilizado en YOLOSPoC, muestra un tiempo de inferencia de 45,17 ms y 22 FPS para imágenes de 60 x 40.

Al considerar el rendimiento combinado de YOLOv3 y ResNet101 en YOLOSPoC para la detección y el cálculo de descriptores de un único objeto, el tiempo de inferencia más rápido que se podría lograr es de 77,47 ms con una tasa de 12 FPS. Esto es más del doble del tiempo que requiere el sistema completo YOLOv7 modificado con JPNet Stage 1. Es importante destacar que debido a la naturaleza de YOLOv3, que tiende a generar múltiples detecciones por inferencia debido a su configuración de baja confidencia, esto podría incrementar aún más el tiempo de inferencia, ya que requeriría más cálculos para la CNN ResNet 101. En contraste, YOLOv7 modificado con JPNet Stage 1 es capaz de procesar y generar descriptores para al

Tabla 5.17: Resultados de tiempos de inferencias obtenidos de YOLOv7 modificado con JPNet Stage 1, redes de YOLOSPoC (YOLOv3 - ResNet101) y estimado de YOLOSPoC.

Resolución	**Modelo**	**Tiempo inferencia [ms]**	**FPS**
640 x 480	YOLOv7mod & JPNet Stage 1	31,25	32
640 x 480	YOLOv3 (YOLOSPoC)	32,3	31
60 x 40	ResNet101 (YOLOSPoC)	45,17	22
640 x 480	YOLOSPoC	$\geq$77,47	$\leq$12

menos 100 propuestas de detección por inferencia, manteniendo una tasa de 32 FPS.

5.3.3. Resultados visuales finales

Para concluir el análisis del sistema completo, se presentan resultados visuales de tres escenarios clave para entender cómo se refleja el *F1-score* alcanzado por el sistema con JPNet Stage 1 en la detección de objetos. Los escenarios seleccionados son S1 (un objeto en ambiente ideal), M3 (seis objetos en ambiente ideal) y M2 (seis objetos con fondos variados). Además se incluye resultados del conjunto de prueba del conjunto RPC, considerando 6 instancias de objetos conocidas. Los resultados visuales de los demás escenarios de DSLL se incluyen en el Anexo 6.1.

El escenario S1 destaca por ser aquel en el que tanto la evaluación del extractor de descriptores como la del sistema completo lograron las mejores métricas. Esto se debe a que corresponde al mismo ambiente en el que se tomaron las imágenes de referencia. De manera similar, el escenario M3, aunque con seis objetos, mantiene un ambiente idéntico, lo que lo convierte en un caso útil para analizar la detección y reconocimiento en condiciones parecidas. En cambio, M2 es el escenario con el *F1-score* más bajo en evaluaciones cuantitativas, debido a su diversidad de fondos y cambios en la iluminación.

Es importante señalar que el umbral utilizado para estas imágenes es el mismo en todos los escenarios, 0.37, como en las evaluaciones anteriores. Este umbral se establece no por ambiente, sino para maximizar el *F1-score* promedio en todos los escenarios para cada modelo.

Resultados visuales en RPC

Los resultados obtenidos en RPC se muestran en la figura 5.11. Se observa que la detección de objetos se da de buena forma, sin embargo, se ve que existen objetos sin detectar en especial cuando se tiene objetos muy cercanos. Esto se debe a un efecto negativo del bloque de eliminado de detecciones redundantes, donde por efectos del filtro se filtran objetos con IoU superior a 0.1. Por el lado del reconocimiento, se observan pocos errores en la identificación en las detecciones existentes, a pesar de poseer objetos con colores y formas similares, como la lata y coca cola. En el caso del snickers y la barra similar, se etiqueta erróneamente a la barra como un snickers, sin embargo, se observa que la distancia que tiene es de 0.21 a comparación de los verdaderos que tienen un valor de 0.15 y 0.13. Por lo tanto, si es que se utilizará un umbral más bajo y particular al escenario se podría hacer filtrado de este objeto.

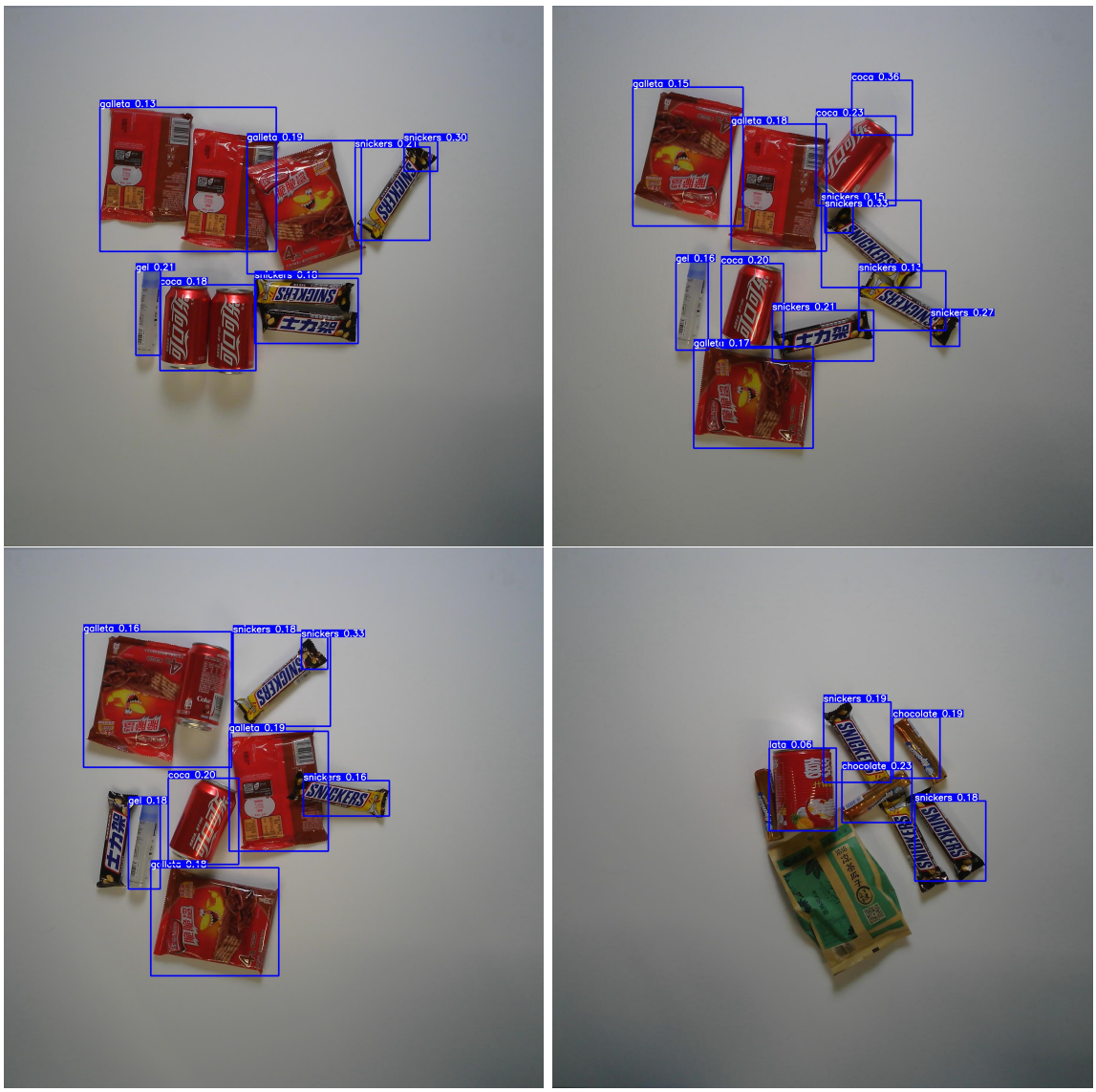


Figura 5.11: Resultados obtenidos del sistema propuesto en RPC.

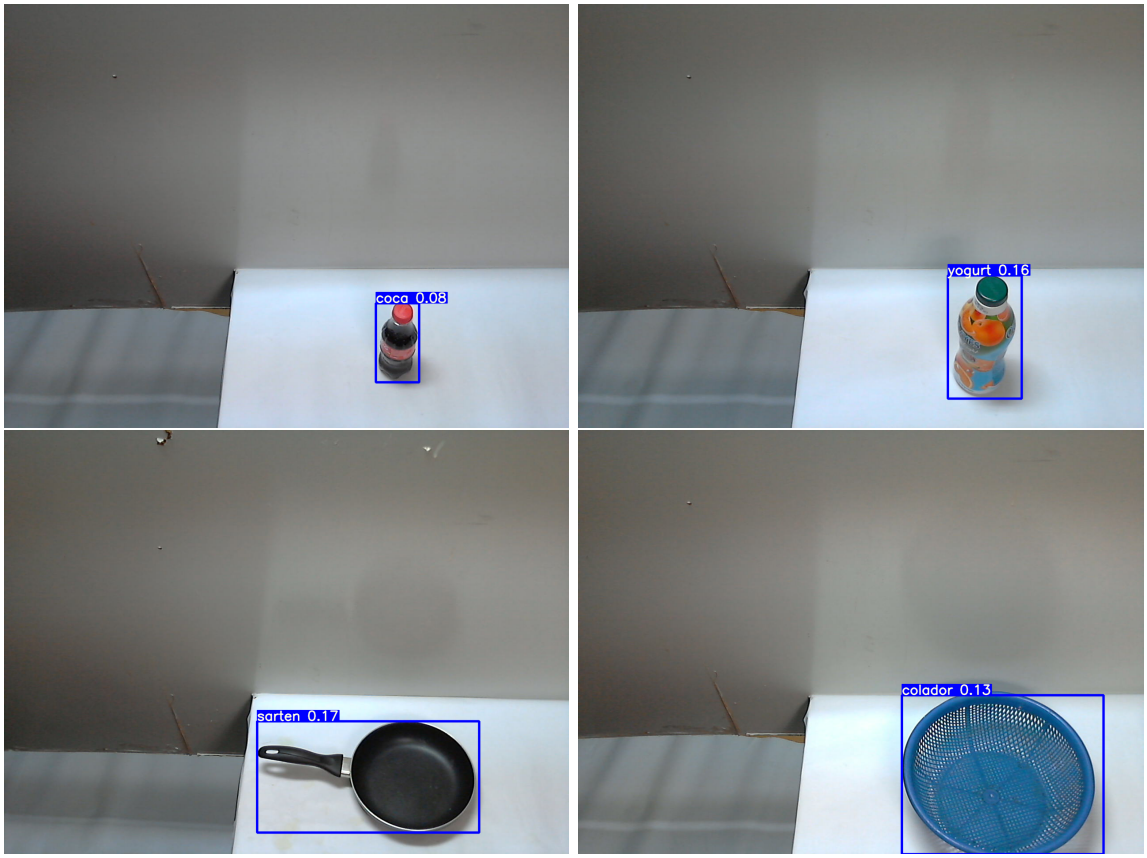


Figura 5.12: Resultados obtenidos del sistema propuesto en el escenario S1.

Resultados visuales en escenario S1

Los resultados obtenidos en S1 se muestran en la figura 5.12. Aquí se observa que la detección de objetos se realiza de forma correcta y precisa, incluso para objetos complicados como el sartén y el colador. Junto a la etiqueta de la instancia de objeto, se muestra la distancia al objeto de referencia, indicando que, cuanto menor es el valor, más confiable es el reconocimiento. En cuatro las imágenes, el objeto con la mayor distancia resulta ser el sartén, lo cual es coherente dada su forma. También se detecta una gran cantidad de supuestos objetos alrededor de la mesa. Sin embargo, como sus distancias son significativamente mayores al umbral de 0.37, se clasifican como objetos desconocidos (*unknown*), siendo ignorados por el sistema. Esta cantidad de detecciones se debe al uso de un detector de objetos genéricos entrenado con SAM, lo cual no afecta negativamente al sistema siempre y cuando el módulo de reconocimiento sea capaz de diferenciar adecuadamente entre objetos conocidos y desconocidos. De no ser así, esto podría llevar a una disminución tanto en las métricas de rendimiento como en la efectividad visual del sistema.

Resultados visuales en escenario M3

En el escenario M3, representado en la figura 5.13, se evidencia la capacidad del sistema para detectar de manera precisa casi todos los objetos. Una excepción es el yogurt en la

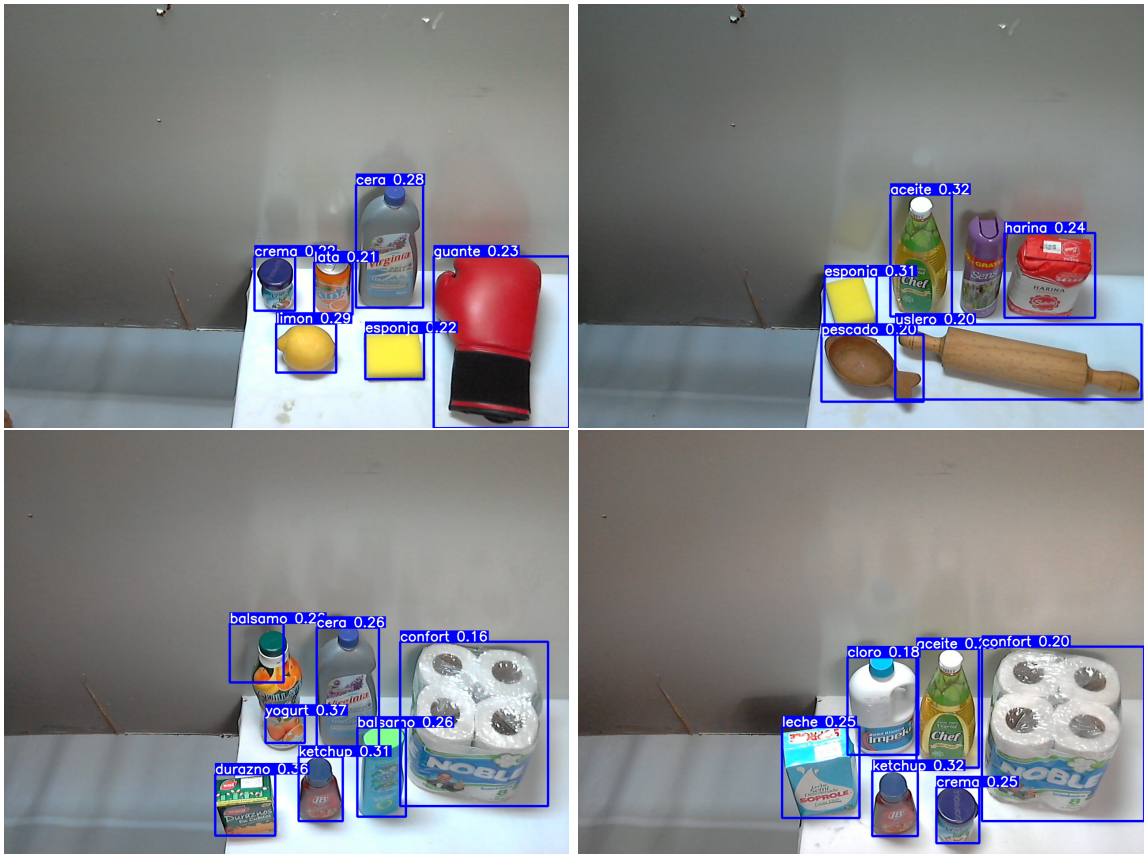


Figura 5.13: Resultados obtenidos del sistema propuesto en el escenario M3.

tercera imagen, donde se generan dos recuadros para abarcar completamente el objeto. Es destacable la detección exitosa de objetos atípicos como el guante de boxeo, el rollo de papel higiénico y el uslero, que no se encuentran entre las 80 clases estándar de COCO.

En términos de reconocimiento, se observa un patrón similar al del escenario S1, con la mayoría de los objetos siendo correctamente identificados con su instancia correspondiente. Sin embargo, se observa un aumento en la distancia promedio de los objetos en cada imagen, alcanzando valores de hasta 0.32 para el ketchup y el aceite, y 0.37 para el yogurt, rozando el umbral de reconocimiento establecido. Este incremento se puede deber a la mayor cantidad de objetos presentes, sus diferentes ubicaciones y la posible superposición de los recuadros de detección, lo que podría afectar la pureza del descriptor generado en comparación con el escenario S1. Un ejemplo claro de esto es el aerosol en la segunda imagen, que no fue reconocido correctamente debido a que su distancia fue de 0.38, ligeramente por encima del umbral.

En cuanto a las detecciones incorrectas de supuestos objetos, todas presentan valores de distancia elevados en relación a los objetos de referencia. Esto permite al sistema de reconocimiento descartar estas detecciones como objetos desconocidos, demostrando la eficacia del reconocedor en filtrar las detecciones incorrectas o irrelevantes.

Figura 5.14: Resultados obtenidos del sistema propuesto en el escenario M2.

Resultados visuales en escenario M2

Finalmente, los resultados visuales del escenario M2, que presentó el rendimiento más bajo, se muestran en la figura 5.14. Aquí se puede ver que, aunque el sistema logra detectar la mayoría de los objetos en la imagen, el reconocimiento se ve significativamente afectado por los cambios de fondo y de tonalidad en la iluminación. Un ejemplo claro se observa en la segunda imagen, donde el sistema no consigue reconocer ningún objeto como conocido, ya que las distancias calculadas superan el umbral establecido. Este incremento en las distancias es previsible dada la variabilidad del ambiente.

No obstante, también se aprecia que aquellos objetos cuya distancia es menor a 0.37 suelen ser reconocidos correctamente, con la excepción notable en la tercera imagen, donde se confunde un bálsamo con una caja de leche, probablemente debido a la similitud en sus colores. Además, en este escenario se presenta una situación no observada en los anteriores: en la tercera y cuarta imagen, una detección en la esquina inferior izquierda se identifica erróneamente como confort, con distancias de 0.37 y 0.34, respectivamente. Estas fallas en el reconocimiento ponen en evidencia una debilidad del sistema, ya señalada en las evaluaciones de las secciones anteriores, que radica en la sensibilidad del sistema en la generación de descriptores frente a cambios significativos en el fondo y el ambiente.

Finalmente, basándonos en los resultados obtenidos y analizados en estos cuatros escenarios, se puede concluir que para mejorar la capacidad del sistema de reconocer objetos

correctamente, sería beneficioso ajustar el valor umbral de reconocimiento de manera específica para cada escenario. En ambientes similares a los de las imágenes de referencia, se podría considerar un umbral más bajo, ya que es probable que las distancias relativas de los objetos a sus referencias sean menores. Por otro lado, en escenarios con ambientes considerablemente diferentes, aumentar el umbral podría ser una estrategia eficaz. Esto se debe a que, en tales situaciones, las distancias relativas de los objetos tienden a ser mayores, y un umbral más alto ayudaría a evitar la identificación incorrecta de objetos conocidos como desconocidos.

Este enfoque de ajustar el umbral de reconocimiento para cada escenario específico puede, efectivamente, mejorar las métricas de precisión del sistema en distintas situaciones. Sin embargo, esta estrategia tiene como consecuencia la pérdida de generalización del sistema para otros escenarios. Esto podría resultar en una experiencia de uso más compleja e incluso poco práctica para el usuario, ya que requeriría manejar un conjunto de umbrales diferentes y conocer con precisión en qué escenario es más adecuado utilizar cada uno.

Capítulo 6

Conclusiones

En esta tesis, se enfrentó el desafío de la detección de instancias de objetos en tiempo real mediante la integración de un detector de objetos y un módulo de extracción de descriptores en una única red.

Se implementó una novedosa modificación en la función de perdida de YOLOv7, obteniendo una nueva red de detección de objetos genéricos, denominada YOLOv7 modificado. El proceso de entrenamiento se llevó a cabo con un conjunto de datos de detección mejorado, basado en MS COCO pero con etiquetas de todos los objetos presentes en las imágenes, utilizando la red de segmentación SAM. Este enfoque propuesto logra ser una contribución, ya que supera el rendimiento tanto de YOLOv7 como de YOLOv7 modificado entrenado con MS COCO original en la detección de objetos genéricos.

En cuanto al módulo extractor de descriptores, se diseñó y desarrolló una arquitectura completa que incluye una CNN, una capa RoI, una operación GeM y una capa Neck para obtener descriptores globales de cada objeto detectado. Se investigó el impacto del stage del backbone al que se integra y se analizó el impacto de usar una CNN especializada. La CNN diseñada y propuesta en este trabajo, JPNet, alcanzó los mejores resultados F1-score, tanto al conectarse al stage 1 como al stage 2, superando a los enfoques baseline y CNN basada en VoVNet. Estos hallazgos permiten deducir que el uso de una CNN especializada es beneficioso para la obtención de mejores descriptores, por ende, un mejor reconocimiento de instancias.

Se corroboró el correcto funcionamiento del sistema completo y se validó su operación en tiempo real. Se concluyó que el extractor que mejor se adaptó al sistema corresponde a JPNet conectado al stage 1, alcanzando un F1-score de 0.742 a una velocidad de 32 FPS, comparado con 0.659 a 44 FPS de JPNet conectado al stage 2. Para validar los beneficios del sistema propuesto, se realizó una comparación con YOLOSPoC, sistema de referencia que motivó este trabajo. YOLOv7 modificado con JPNet logra superar el rendimiento F1-score de YOLOSPoC en los escenarios S1 y S2 de DSLL. Sin embargo, en el rendimiento promedio, el sistema propuesto alcanzó un 0.736 frente a 0.843 de YOLOSPoC. A pesar de esto, al ser YOLOv7 modificado con JPNet un sistema integrado, supera en velocidad al menos al doble a YOLOSPoC, el cual es un sistema no integrado.

6.1. Trabajo futuro

Esta tesis valida a YOLOv7 modificado con JPNet como un sistema innovador y eficiente para la detección de instancias de objetos en aplicaciones que requieran operar en tiempo real, cumpliendo satisfactoriamente con los objetivos propuestos. Sin embargo, aún hay aspectos y líneas de investigación que se pueden explorar para mejorar la precisión del sistema, tanto en detección como en la generación de descriptores. Estas se detallan a continuación.

Aunque YOLOv7 modificado, entrenado con COCO etiquetado mediante SAM, supera en precisión a los detectores entrenados solo con COCO en la detección de objetos genéricos, esta precisión puede incrementarse aún más. Una estrategia sería entrenar el sistema con la combinación de varios conjuntos de datos para ampliar la diversidad de objetos detectables. Incrementar la variedad de objetos en el entrenamiento mejoraría la capacidad del sistema para detectar un rango más amplio de objetos, basándose en un mayor espectro de formas y texturas. Sin embargo, esta mejora vendría con el costo de un entrenamiento más extenso y exigente computacionalmente.

Una línea de investigación interesante para abordar la sensibilidad al cambio de fondo consiste en reemplazar el detector por un segmentador de instancias de objetos. Esto permitiría que, al generar el descriptor global de cada objeto, la atención se centre exclusivamente en el objeto, omitiendo su contorno. Entre las opciones viables se encuentra adaptar una YOLOv7 con un cabezal de segmentación o emplear directamente una versión más ligera de SAM.

Por otro lado, se podría investigar sobre variaciones en la arquitectura del extractor de descriptores propuesto en este trabajo. Una posibilidad es adelantar la etapa de extracción de regiones de interés, pasando los mapas de características por una CNN que procese cada una de estas características y, mediante una operación adecuada, los convierta en descriptores globales. Esto podría simplificar el entrenamiento de la CNN, ya que se propagaría el gradiente de la pérdida que se calculó de toda la CNN en lugar de solo partes de ella. Además, sería beneficioso realizar un análisis sobre qué operaciones de *pooling* son más adecuadas para la detección de instancias de objetos según la arquitectura específica.

Finalmente, en este trabajo, el módulo extractor de descriptores se entrenó con un conjunto de instancias de objetos domésticos, específicamente productos de retail. Como se mencionó en los alcances, para reconocer instancias de objetos de naturaleza diferente, como vehículos o personas, este módulo debería ser reentrenado con un conjunto de datos adecuado. Sería interesante explorar el entrenamiento del extractor con un *dataset* que contenga una cantidad superior a las 200 instancias de objetos de este trabajo, integrando una amplia variedad de instancias de objetos etiquetadas.

Bibliografía

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4321–4330. IEEE, 2020.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[3] Jiankang Deng, Jia Guo, Tongliang Liu, Mingming Gong, and Stefanos Zafeiriou. Sub-center arcface: Boosting face recognition by large-scale noisy web faces. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 741–757. Springer, 2020.

[4] Jiankang Deng, Jia Guo, Jing Yang, Niannan Xue, Irene Kotsia, and Stefanos Zafeiriou. ArcFace: Additive angular margin loss for deep face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):5962–5979, oct 2022.

[5] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13733–13742, 2021.

[6] E. M. Dogo, O. J. Afolabi, N. I. Nwulu, B. Twala, and C. O. Aigbavboa. A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks. In *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, pages 92–99, 2018.

[7] Utkarsh Doshi. Segment Anything Model (SAM) explained, 2023. URL: `https://medium.com/@utkarsh135/segment-anything-model-sam-explained-2900743cb61e`[Último acceso: 14/04/2024].

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7):2121–2159, 2011.

[9] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[10] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2012.

[11] Ali Farhadi and Joseph Redmon. Yolov3: An incremental improvement. In *Computer Vision and Pattern Recognition*, volume 1804, pages 1–6, Berlin/Heidelberg, Germany, June 2018. Springer.

[12] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2918–2928, 2021.

[13] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[16] Qishen Ha, Bo Liu, Fuxu Liu, and Peiyuan Liao. Google landmark recognition 2020 competition third place solution. *DeepAI*, 2020.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 558–567, 2019.

[20] Jeff Heaton. Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618. *Genetic programming and evolvable machines*, 19(1):305–307, 2018.

[21] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*, pages 84–92. Springer, 2015.

[22] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*, pages 84–92. Springer, 2015.

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015*, pages 1–13, San Diego, CA, USA, 2015.

[24] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, and Ross Girshick. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4015–4026, October 2023.

[25] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, pages 1–30. Lille, 2015.

[26] Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. An energy and gpu-computation efficient backbone network for real-time object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 1–9, 2019.

[27] Kunpeng Li, Ziyan Wu, Kuan-Chuan Peng, Jan Ernst, and Yun Fu. Tell me where to look: Guided attention inference network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9215–9223, 2018.

[28] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[29] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[30] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[31] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[32] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 212–220, 2017.

[33] Patricio Loncomilla and Javier Ruiz-del Solar. Yolospoc: recognition of multiple object instances by using yolo-based proposals and deep spoc-based descriptors. In *RoboCup 2019: Robot World Cup XXIII 23*, pages 154–165. Springer, 2019.

[34] Patricio Loncomilla, Javier Ruiz-del Solar, and Luz Martínez. Object recognition using local invariant features for robotic applications: A survey. *Pattern Recognition*, 60:499–514, 2016.

[35] Qi-Chao Mao, Hong-Mei Sun, Yan-Bo Liu, and Rui-Sheng Jia. Mini-yolov3: real-time object detector for embedded applications. *IEEE Access*, 7:133529–133538, 2019.

[36] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[37] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/k2). In *Dokl. Akad. Nauk. SSSR*, volume 269, page 543, 1983.

[38] Rafael Padilla, Wesley L Passos, Thadeu LB Dias, Sergio L Netto, and Eduardo AB Da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3):279, 2021.

[39] David M W Powers. Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[40] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[41] Filip Radenović, Giorgos Tolias, and Ondřej Chum. Fine-tuning cnn image retrieval with no human annotation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(7):1655–1668, 2019.

[42] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[43] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[44] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.

[45] Synced Review. Look Again, YOLO baidus rt-detr detection transformer achieves sota results on real-time object detection, 2023. URL: `https://goo.su/2v0Ozf6` [Último acceso: 14/04/2024].

[46] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 658–666, 2019.

[47] Roboflow. YOLOv9 PyTorch TXT, 2024. URL: `https://roboflow.com/formats/yolov9-pytorch-txt` [Último acceso: 14/04/2024].

[48] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[49] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[50] Yutaka Sasaki et al. The truth of the f-measure. *Teach tutor mater*, 1(5):1–5, 2007.

[51] K. Shanmugam. *Fundamentals of Computer Graphics and Multimedia*. I.K. International Publishing House Pvt. Limited, 2019.

[52] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[53] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, pages 1–14, 2015.

[54] Fotis Sofoulis. Deployment of temporal gnns for mining large social networks over spark on kubernetes. Master's thesis, University of Patras, Patras, Greece, 09 2023. Thesis (M.Sc.)–Multidisciplinary M.Sc. Program "Data Driven Computing and Decision Making".

[55] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[56] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *Advances in neural information processing systems*, 30, 2017.

[57] Alaa Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 17(1):168–192, 2020.

[58] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9627–9636, 2019.

[59] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104:154–171, 2013.

[60] C. Y. Wang, H. Y. Mark Liao, and I-Hau Yeh. Designing network design strategies through gradient path analysis. *Journal of Information Science and Engineering*, 39(4):975–995, 7 2023. Invited paper.

[61] C. Y. Wang, H. Y. Mark Liao, and I-Hau Yeh. You only learn one representation: Unified network for multiple tasks. *Journal of Information Science and Engineering*, 39(3):691–709, 5 2023.

[62] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 13029–13038, 2021.

[63] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7464–7475, 2023.

[64] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5265–5274, 2018.

[65] X. S. Wei, Q. Cui, L. Yang, et al. Rpc: A large-scale and fine-grained retail product checkout dataset. *Sci. China Inf. Sci.*, 65(197101):1–14, 7 2022.

[66] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*, pages 499–515. Springer, 2016.

[67] Zhaohui Zheng, Ping Wang, Dongwei Ren, Wei Liu, Rongguang Ye, Qinghua Hu, and Wangmeng Zuo. Enhancing geometric factors in model learning and inference for object detection and instance segmentation. *IEEE transactions on cybernetics*, 52(8):8574–8586, 2021.

[68] Dingfu Zhou, Jin Fang, Xibin Song, Chenye Guan, Junbo Yin, Yuchao Dai, and Ruigang Yang. Iou loss for 2d/3d object detection. In *2019 international conference on 3D vision (3DV)*, pages 85–94. IEEE, 2019.

Anexo

Los resultados visuales obtenidos en los escenarios S2, S3, S4, M1 y M4 por el modelo JPNet Stage 1, se presentan en las figuras A.1, A.2, A.3, A.4 y A.5.

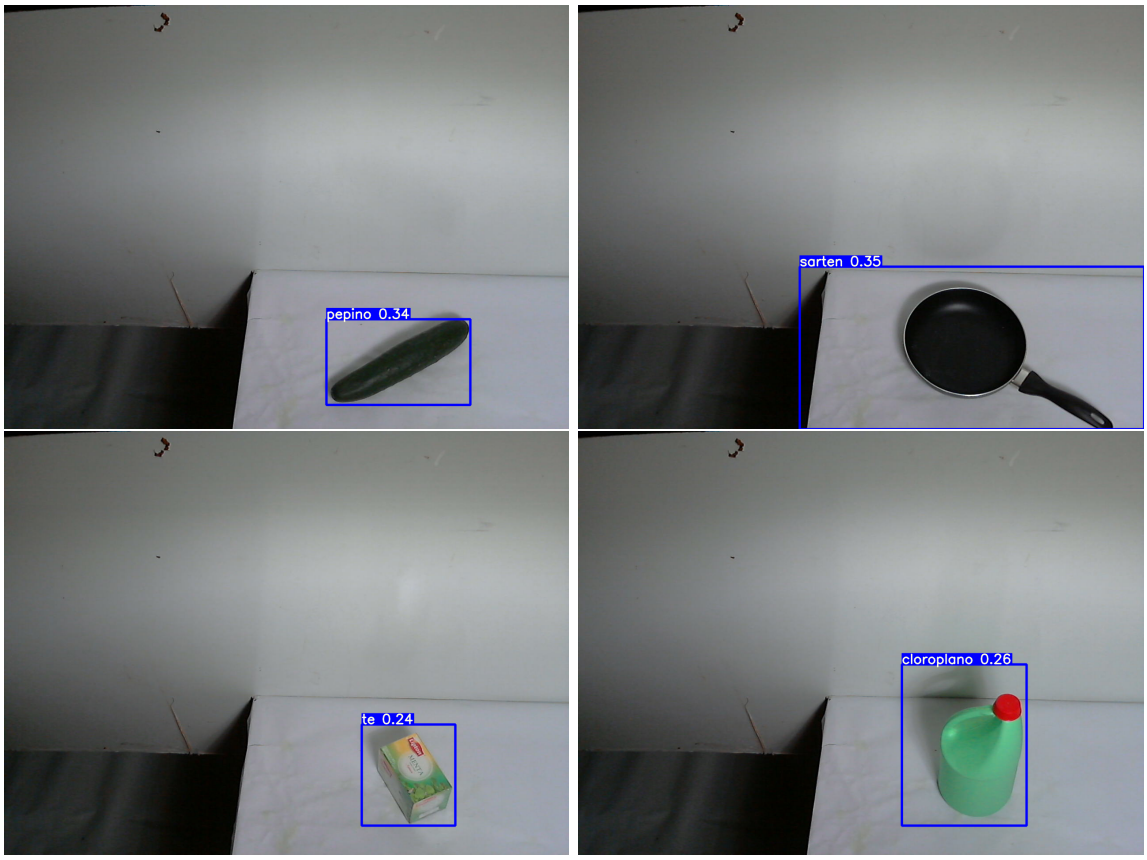


Figura A.1: Resultados obtenidos del sistema propuesto en el escenario S2 por el modelo JPNet Stage 1.

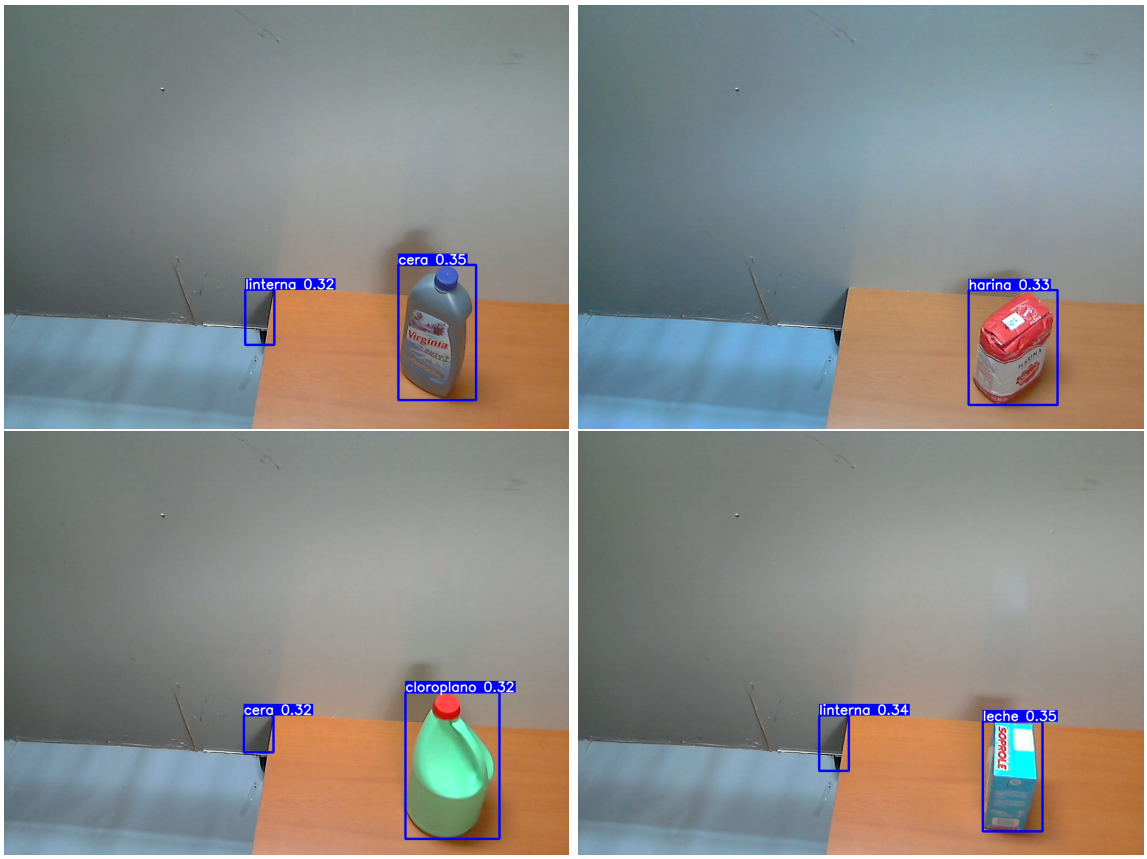


Figura A.2: Resultados obtenidos del sistema propuesto en el escenario S3 por el modelo JPNet Stage 1.

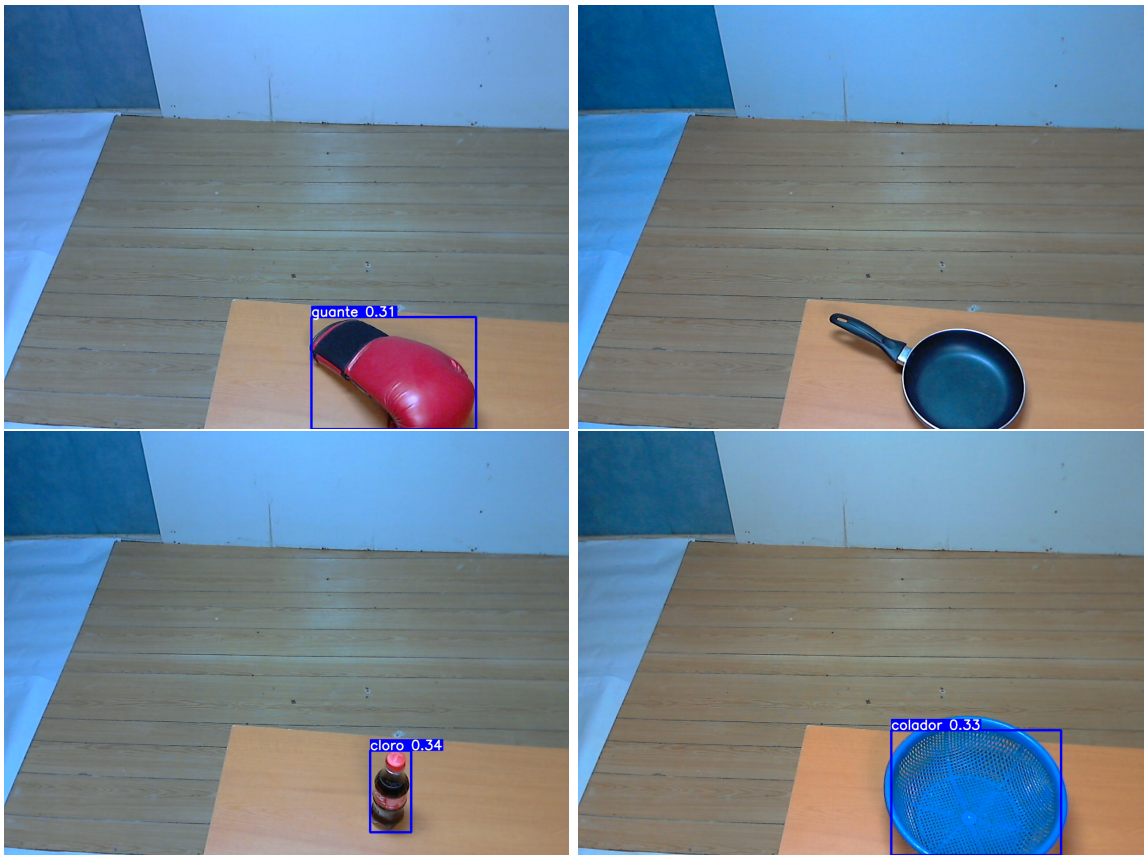


Figura A.3: Resultados obtenidos del sistema propuesto en el escenario S4 por el modelo JPNet Stage 1.

Figura A.4: Resultados obtenidos del sistema propuesto en el escenario M1 por el modelo JPNet Stage 1.

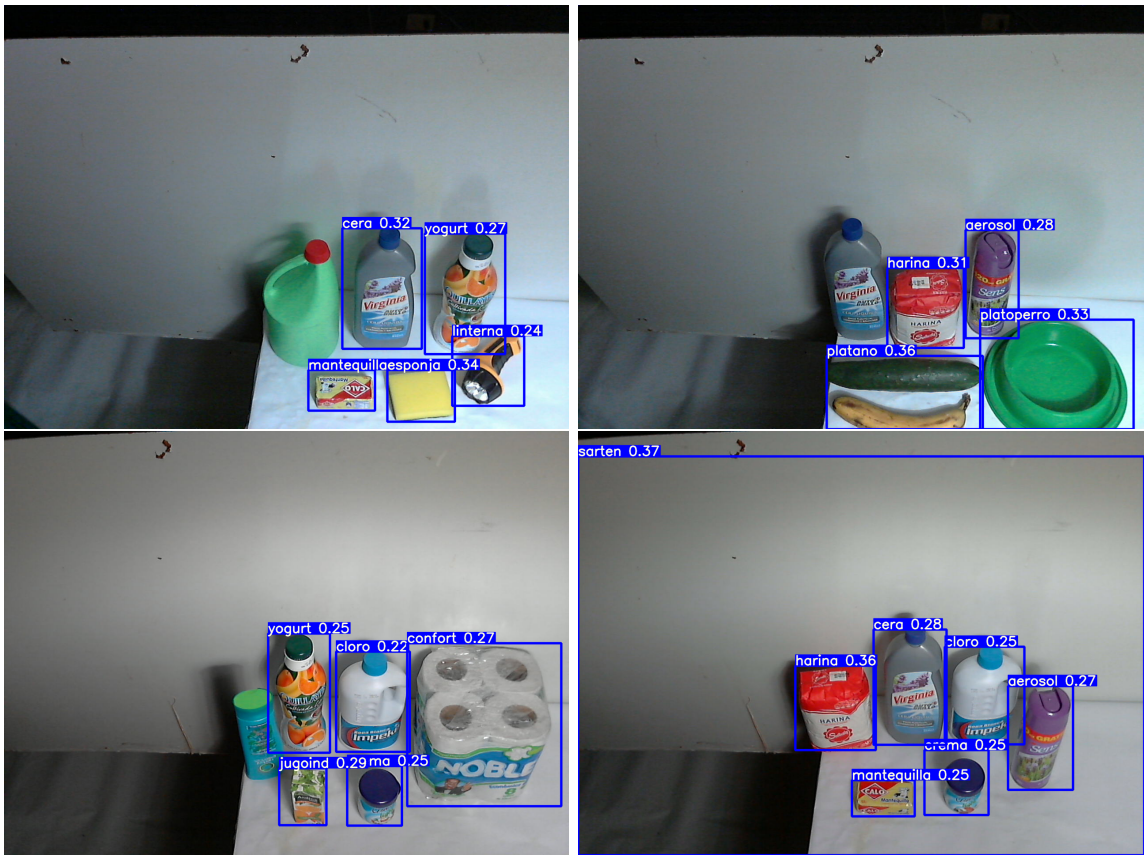


Figura A.5: Resultados obtenidos del sistema propuesto en el escenario M4 por el modelo JPNet Stage 1.