Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN INGENIERÍA DE CONTROL, AUTOMATIZACIÓN Y ROBÓTICA

TRABAJO FIN DE MÁSTER

DETECCIÓN DE OBJETOS BASADA EN MACHINE LEARNING PARA REALIDAD AUMENTADA

Estudiante: Calleja Corcuera, Jon Ander

Director/Directora: Zulueta Guerrero, Ekaitz

Curso: 2022-2023

Fecha: Bibao, a 20 de septiembre de 2023

Resumen

En las últimas décadas, la Realidad Aumentada (RA) ha irrumpido de manera espectacular en diversos ámbitos, transformando la manera en que interactuamos con la información y el entorno que nos rodea. Para que esta tecnología funcione de manera efectiva, es crucial que el sistema pueda identificar y rastrear objetos en tiempo real. Han sido muchas las investigaciones realizadas entorno a dicha detección de objetos en RA, centrándose en sus inicios principalmente en técnicas que utilizan marcadores o seguimiento sin marcadores. Sin embargo, con el avance de las capacidades de procesamiento y la disponibilidad de grandes conjuntos de datos, las técnicas de *Machine Learning* (ML) han emergido como una solución más potente y versátil. Este trabajo se centra principalmente en ello, en explorar y evaluar diversos algoritmos de ML para la detección de objetos en aplicaciones de RA, identificando aquel que mejores resultados ofrezca en términos de precisión y eficiencia.

Abstract

Over the past decades, Augmented Reality (AR) has dramatically burst into various fields, transforming the way we interact with the information and environment around us. For this technology to work effectively, it is crucial that the system can identify and track objects in real time. Much research has been done on such object detection in AR, initially focusing mainly on techniques using markers or markerless tracking. However, with the advancement of processing capabilities and the availability of large datasets, Machine Learning (ML) techniques have emerged as a more powerful and versatile solution. The main focus of this work is to explore and evaluate various ML algorithms for object detection in AR applications, identifying the one that offers the best results in terms of accuracy and efficiency.

Laburpena

Azken hamarkadetan, Errealitate Areagotua modu ikusgarrian sartu da hainbat esparrutan, inguratzen gaituen informazioarekin eta ingurunearekin elkarreragiteko dugun modua eraldatuz. Teknologia honek modu eraginkorrean funtziona dezan, funtsezkoa da sistemak objektuak denbora errealean identifikatu eta arakatu ahal izatea. Asko izan dira RAko objektuen detekzio horren inguruan egindako ikerketak, batez ere markatzaileak erabiltzen dituzten tekniketan edo markatzailerik gabeko jarraipenean oinarrituta. Hala ere, prozesatzeko gaitasunen aurrerapenarekin eta datu-multzo handien eskuragarritasunarekin,*Machine Learning* (ML) teknikak soluzio indartsuago eta moldakorrago gisa agertu dira. Lan hau horretan oinarritzen da batez ere, RA-aplikazioetan objektuak detektatzeko ML-algoritmo batzuk arakatzean eta ebaluatzean, eta doitasun eta efizientzia aldetik emaitzarik onenak ematen dituena identifikatzean.

Palabras clave:

Deep Learning, Realidad Aumentada, detección de objetos, redes neuronales convolucionales

Índice

Lista de tablas

Lista de ilustraciones

Acrónimos

ADAM Estimación del momento adaptativo, del inglés*Adaptative Moment Estimation*

ANN Red neuronal artificial, del inglés *Artificial Neural Network*

CNN Red neuronal convolucional, del inglés *Convolutional Neural Network*

COCO del inglés *Common Objects in Context*

CVAT *Computer Vision Anotation Tool*

DL Deep Learning

FPN red de piramide de caracteristicas, del inglés *Feature Pyramid Network*

GAN Red neuronal generativa adversaria, *Generative Adversarial Network*

HMD Casco de realidad virtual, del inglés *Human Mounted Display*

IA Inteligencia Artifical

IoU Intersección sobre unión, del inglés *Intersection over Union*

JSON del inglés *JavaScript Object Notation*

MAE Error medio absoluto, del inglés *Mean Absolute Error*

mAP del inglés mean Average Precision

MSE Error cuadrático medio, del inglés *Mean Squared Error*

NLP Natural Language Processing

RA Realidad Aumentada

R-CNN Red neuronal convolucional basada en regiones, del inglés *Region-based Convolutional Neural Network*

ReLU Unidad lineal rectificada, del inglés *Rectified Linear Unit*

RMSE Raíz del error cuadrático medio, del inglés *Root Mean Square Error*

RMSProp Propagación de la raíz cuadrática media, del inglés *Root Mean Square Propagation*

RNN Red neuronal recurrente, del inglés *Recurrent Neural Network*

RoI Región de Interés, del inglés *Region of Interest*

RPN Red de Propuesta de Regiones, del inglés *Region Proposal Network*

SGD Gradiente de descenso estocástico, del inglés *Stochastic Gradient Descent*

SSD *Single-shot-multi-box detector*

XML Lenguaje de Marcado Extensible, del inglés *Extensible Markup Language*

YOLO del inglés *You Only Look Once*

1 Introducción y contexto

La Inteligencia Artificial (IA) ha experimentado un notorio avance en los últimos tiempos, convirtiéndose en una tecnología de gran relevancia para abordar una amplia gama de problemas prácticos. Esta evolución se debe, en gran medida, a las altas capacidades de aprendizaje, razonamiento y adaptación de los sistemas inteligentes, lo cual da como resultado que los métodos de IA estén logrando niveles de rendimiento sin precedentes en la resolución de desafíos computacionales complejos [4]. En consecuencia, la IA se ha posicionado como una herramienta de gran valor en múltiples áreas de aplicación, entre las cuales destacan el procesamiento de imágenes, la visión artificial, el diagnóstico médico o la ingeniería aeroespacial.

Dentro de estas áreas de aplicación, ha emergido con fuerza en los últimos años una nueva tecnología disruptiva: la Realidad Aumentada (RA). Esta tecnología se puede definir de manera sencilla como un conjunto de técnicas que permiten la aplicación de elementos virtuales sobre una representación del mundo físico real. De esta manera, se consigue una versión ampliada del mundo físico superpuesta con contenido digital, la cual permite obtener más información del entorno, más allá de la que se puede alcanzar mediante los cinco sentidos que posee el ser humano. Esta combinación de la realidad física y elementos virtuales en tiempo real ofrece experiencias inmersivas y enriquecedoras en diversos campos como el entretenimiento, la medicina o la educación.

En esta área de aplicación, uno de los aspectos clave para lograr una experiencia convincente y fluida es la detección precisa de objetos en el entorno real. La capacidad de identificar y rastrear objetos en tiempo real permite superponer información digital de manera precisa y coherente, lo que mejora la sensación de presencia y realismo de la realidad aumentada. Es en este aspecto precisamente donde se centra la realización de este TFM, en desarrollar algoritmos que permitan realizar tareas de detección de objetos para luego integrarlos en dispositivos de RA. Para ello, los algoritmos y modelos de IA han demostrado ya en distintos ámbitos ser altamente efectivos tanto en la detección como en la clasificación de objetos en imágenes y vídeos. Por lo tanto, se tomarán éstos como punto de partida para el presente trabajo, ya que proporcionan una base sólida para la futura implementación de sistemas de detección de objetos en tiempo real para la Realidad Aumentada.

Este trabajo ha sido desarrollado realizando prácticas extracurriculares en Tecnalia, centro de investigación y desarrollo tecnológico referente en Europa; en concreto, en una de sus sedes del Parque Tecnológico de Álava, dentro del Core de Visual. Para

llevar a cabo el proyecto, se ha utilizado una CPU Intel(R) Core(TM) i5-8365U para generar código y obtener el dataset, así como una GPU NVIDIA GeForce RTX 2080 Ti para el entrenamiento de las distintas arquitecturas para la red neuronal. En cuanto a los programas empleados, todo el código y simulaciones han sido desarrollados con Python 3.10.7, dentro del entorno de Visual Studio Code.

Con todo ello, mediante la realización de este TFM, se pretende realizar un análisis de las distintas técnicas utilizadas para tareas de detección de objetos. De esta manera, se explorarán diferentes enfoques basados en Machine Learning con el objetivo de comparar sus resultados, seleccionando aquella que mejores prestaciones ofrezca para su futura integración en un dispositivo de RA.

2 Alcance y objetivos

La llegada de la digitalización y la Industria 4.0 ha transformado la forma en que se llevan a cabo las operaciones industriales. La automatización, la recopilación de datos en tiempo real y la conectividad en red han dado lugar a fábricas más eficientes y procesos de producción más inteligentes. En este contexto, la Realidad Aumentada (RA) ha emergido como una herramienta muy útil para ayudar en el desarrollo de la industria.

Esta tecnología revolucionaria combina el mundo físico con elementos digitales, proporcionando una experiencia inmersiva que está transformando la manera en la que el ser humano interactúa con su entorno y, en particular, está teniendo un impacto significativo en el mundo industrial. Esto simplifica la formación y la ejecución de tareas, reduciendo los errores y los tiempos de inactividad. Además, la RA habilita el mantenimiento predictivo al permitir que los técnicos accedan a manuales interactivos y guías de reparación, lo que agiliza la resolución de problemas y prolonga la vida útil de los equipos.

Entre alguna de las principales ventajas introducidas por esta tecnología, destaca la mejora en la eficiencia de las operaciones, ya que es capaz de proporcionar información relevante en tiempo a los operarios en su campo de visión. Esto simplifica la formación y la ejecución de tareas, reduciendo los errores y los tiempos de inactividad. Además, la RA habilita el mantenimiento predictivo al permitir que los técnicos accedan a manuales interactivos y guías de reparación, lo que agiliza la resolución de problemas y prolonga la vida útil de los equipos. Asimismo, el poder visualizar esos datos en tiempo real permite a los operadores tomar decisiones con mayor rigor y seguridad, optimizando así la producción. En pocas palabras, la Realidad Aumentada se está convirtiendo en un componente esencial de la Industria 4.0 y la digitalización industrial al aumentar la productividad, mejorar la seguridad, acelerar los procesos de toma decisiones y reducir costos.

Sin embargo, para poder sacar el mayor provecho posible de esta tecnología, en primer lugar existe la necesidad de contar con los dispositivos de hardware adecuados. Son muchas las empresas y marcas que trabajan y realizan constantes estudios en este aspecto, fabricando gafas inteligentes, tables o smartphones cada vez más potentes. No obstante, existen también otros aspectos que se deben tener en cuenta, los cuales pueden no ser tan lógicos como el anterior. En este caso, el principal reto que se debe afrontar es el de conseguir unos algoritmos de software lo más eficientes posibles. Estos algoritmos son requeridos por todo sistema de RA, ya que

deben ser capaces de identificar con precisión el entorno real que lo rodea y todos los objetos que lo componen para tener un rendimiento óptimo. La mayoría de estas tecnologías deben reconocer un mapa espacial en 3D de los componentes a su alrededor escaneando el mundo real. Es decir, dicho de otra manera, cualquier sistema que se base en esta tecnología debe ser capaz de realizar una detección de objetos del mundo real lo más precisa, rápida y eficaz posible en todo momento.

Es precisamente en este aspecto en el que se centra este trabajo. Como objetivo principal, se pretende explorar diferentes enfoques basados en el aprendizaje automático (principalmente, desarrollo de redes neuronales convolucionales) para comparar y seleccionar la opción más interesante para integrar en un dispositivo de RA. Asimismo, para poder llegar a tal punto, se pretende completar los siguientes objetivos y tareas:

- El primer objetivo reside en generar un dataset lo más completo posible para realizar las simulaciones oportunas
- Tras ello, se pretende familiarizarse con distintos algoritmos basados en Machine Learning
- El siguiente objetivo consiste en desarrollar distintos algoritmos para detección de objetos
- Por último, se deberán manejar los resultados obtenidos para extraer las conclusiones correspondientes

3 Antecedentes bibliográficos o estado del arte

Para poder conocer el ámbito en el que se lleva a cabo el trabajo, se ha optado por dividir el estado del arte en dos secciones. Por un lado, en primer lugar se realizará una pequeña introducción a la Inteligencia Artificial y, de manera más concreta, del Aprendizaje Automático (ML, del inglés *Machine Learning*). Por otro lado, la segunda sección se centra en la Realidad Aumentada y los beneficios que aporta esta tecnología en el mundo industrial.

3.1 Introducción a la Inteligencia Artificial y el *Machine Learning*

La Inteligencia Artificial (IA) puede ser definida de diversas maneras según el enfoque o el contexto en el que se esté hablando. Muchas de esas definiciones hacen referencia a la capacidad de una máquina para igualar el comportamiento humano, mientras que otras dan mayor importancia a la racionalidad y al decidir de manera correcta según el conocimiento de la máquina. Del mismo modo, existen autores que valoran en un alto grado el razonamiento de las máquinas, existiendo otros que se centran en mayor medida en la conducta de las mismas. Centrándose en aquellas definiciones que entienden las máquinas como sistemas que emulan el comportamiento humano, aparecen las siguientes definiciones:

- "El estudio de programas de computadora que pueden mejorar su rendimiento en tareas que, si se hicieran por un ser humano, requerirían inteligencia" [24]
- "Una máquina es llamada 'inteligente' si logra que un interrogador humano no pueda determinar cuál es la máquina y cuál es el humano basándose únicamente en las respuestas a preguntas" [45]
- "... máquinas que pueden simular cualqier inteligencia humana" [2]

En cuanto a aquellas definiciones que destacan la capacidad de los sistemas de actuar de manera correcta, destacan las definiciones que se muestran a continuación:

- "... es el campo de la informática que se enfoca en desarrollar sistemas y algoritmos capaces de permitir a las máquinas comprender, razonar, aprender

y tomar decisiones autónomas, a menudo con el objetivo de lograr resultados óptimos o cercanos a óptimos en tareas específicas" [44]

- "Un conjunto de algoritmos y técnicas que permiten a las computadoras aprender y tomar decisiones de manera autónoma" [21]

Asimismo, existen muchos autores que se focalizan en la capacidad de los sistemas de aprender a través de datos. En este aspecto, es destacable la siguiente definición:

- "La capacidad de una red neuronal profunda para aprender representaciones jerárquicas de datos" [17]

En definitiva, la IA es un campo de la informática y la ciencia en constante crecimiento, el cual se centra en desarrollar sistemas que puedan realizar tareas de manera autónoma y con un buen rendimiento. Dentro de ella, como se ha comentado anteriormente, se engloba una gran variedad de técnicas y subcampos, entre los cuales destaca el Aprendizaje Automático (ML, del inglés *Machine Learning*). Según Arthur Samuel [43], el ML es el campo de estudio que le da a las computadoras la capacidad de aprender sin ser programadas explícitamente. Asimismo, Tom Mitchell [3] indicó que un programa de computadora, el cual arranca con un índice de efectividad concreto, va aprendiendo a partir de los datos y acontecimientos que le llegan. Se trata de ir mejorando dicho rendimiento, a través de la mayor cantidad de datos posible, de tal manera que el programa acabe siendo autónomo. En pocas palabras, se entiende el Aprendizaje Automático como un proceso de aprendizaje, donde el programador proporciona una serie de reglas que el algoritmo ha de ir adaptando y, creando otras nuevas, para tratar de mejorar el rendimiento del modelo generado a través de la experiencia.

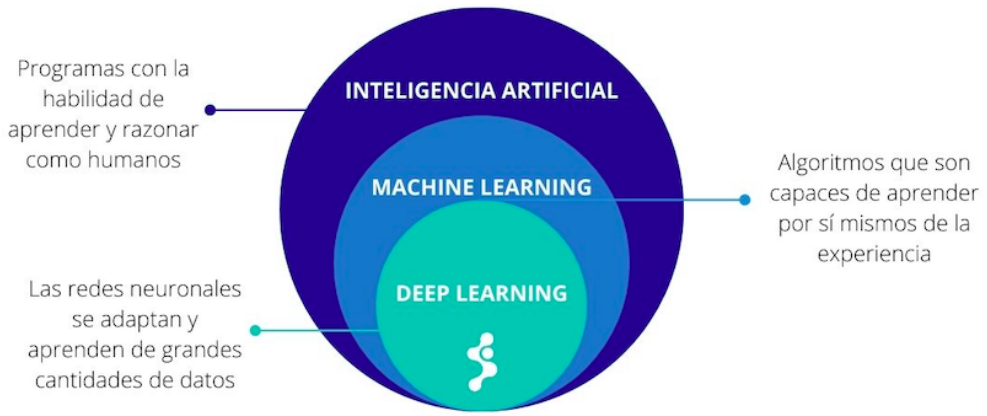


Figura 3.1: Jerarquía e interacción entre Inteligencia Artifical, *Machine Learning* y *Deep Learning* [1]

[1]Fuente: `https://iasolver.es/que-es-la-inteligencia-artificial/`

A su vez, como se puede observar en la Figura 3.1, dentro del ML, destaca también un subconjunto de técnicas conocidas como Aprendizaje Profundo (DL, del inglés *Deep Learning*). Este enfoque, inspirado en la arquitectura y funcionamiento del cerebro humano, propone modelar abstracciones de alto nivel de los datos empleando para ello arquitecturas compuestas por un elevado número de capas de transformaciones que pueden ser tanto lineales como no lineales [43]. Al igual que en el funcionamiento del cerebro humano, este tipo de redes, al recibir nueva información, la compara con los conocimientos previamente adquiridos e intenta darle sentido a este nuevo estímulo. De esta manera, el DL consigue descubrir estructuras complejas en grandes conjuntos de datos utilizando el algoritmo de retropropagación (del inglés *backpropagation*) para indicar cómo una máquina debería cambiar sus parámetros internos usados para el cálculo de la representación en cada capa desde la representación de la capa anterior.

La gran aportación de las técnicas de DL se encuentra en la posibilidad de entrenar arquitecturas funcionales profundas (modelos complejos con gran cantidad de capas), donde las técnicas clásicas fallan debido a problemas de bajo gradiente en la adaptación de los pesos de las capas más internas. Además, estas técnicas son pioneras en su escalabilidad respecto a la cantidad de datos de entrada. Como se observa en la Figura 3.2, cuantos más datos se manejen, mejor comportamiento se obtiene, lo cual permite explotar al máximo los beneficios de otros revolucionarios conceptos como el *Big Data* o la Industria 4.0.

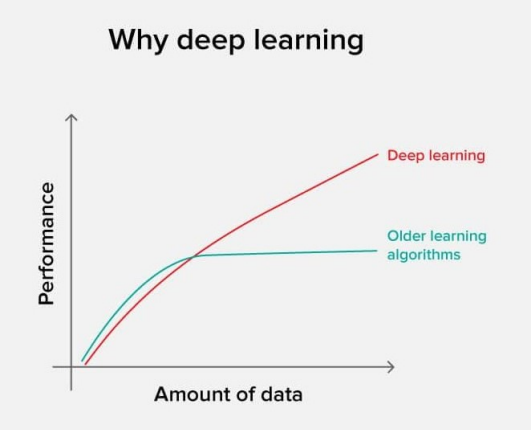


Figura 3.2: Evolución en el rendimiento de los sistemas al utilizar técnicas basadas en *Deep Learning* [2]

Es ésta una de las principales razones por las que en los últimos tiempos, la técnicas de DL han revolucionado el mundo industrial, aportando resultados muy por encima a los obtenidos con otras técnicas del campo de la IA. Entre las múltiples aplicaciones en las que se están desarrollando algoritmos de DL, destaca la visión por computador, el reconocimiento de caras o los traductores inteligentes [53].

[2]Fuente: Artículo web escrito por Yulia Gavrilova, disponnible en `https://serokell.io/blog/deep-learning-and-neural-network-guide`

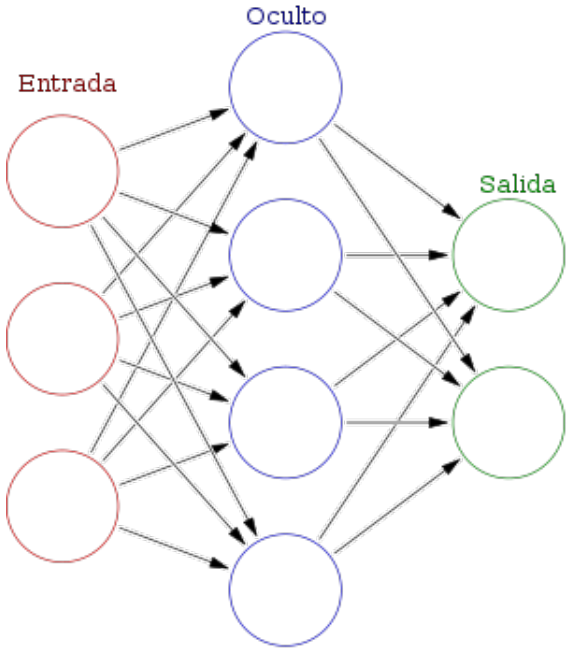


Figura 3.3: Diagrama de una red neuronal superficial, que contiene la capa de entrada, una capa oculta y la de salida [3]

Dentro del Aprendizaje Profundo existen varios aspectos a destacar, los cuales son fundamentales para comprender y aplicar esta tecnología de manera efectiva. Entre ellos sobresale el concepto de arquitectura de las redes neuronales. Una red neuronal artificial (ANN, del ingles *Artificial Neural Network* consiste en una capa de entrada de neuronas, una o más capas ocultas y una capa final de salida [49]. Cuando la red presenta una única capa oculta, como es el caso de la Figura 3.3, se habla de ANN superficiales, mientras que, cuando hay un mayor número de capas ocultas, les denomina ANN profundas, siendo este segundo caso el perteneciente a técnicas de DL.

Las redes neuronales generalmente tienen una arquitectura compuesta por neuronas, las cuales se encargan de llevar a cabo el cálculo de las funciones matemáticas dentro de la red y hacer fluir los datos a lo largo de la misma. Cada neurona de estas neuronas está compuesta por los siguientes componentes:

- Variables de entrada (debe existir al menos una). Estas variables se representan comúnmente mediante el vector $\boldsymbol{x} = [x_1, x_2, x_3, ...x_N]$
- Un peso sináptico asignado a cada una de las variable de entrada, el cual determina la importancia de cada conexión y se representa mediante el vector $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3, ...\omega_N]$

[3] Fuente: `https://es.wikipedia.org/wiki/Red_neuronal_artificial`

- Un parámetro llamado sesgo, b. Este parámetro indica la facilidad que tiene dicha neurona de activarse y permite conocer cómo de fácil de detectar es esa característica

- Una función de propagación que actúa sobre los pesos sinópticos y el sesgo antes de acceder a la función de activación correspondiente. Estas funciones pueden ser tanto lineales como no lineales y la más utilizada se muestra a continuación:

$$y = \sum_{i=1}^{N} w_i x_i + b \tag{3.1}$$

- Una función de activación encargada de determinar si la salida de la neurona se activa en función del valor resultante de la función de propagación, $z = f_{act}(y)$. Más adelante en el documento se detallan las funciones de activación más utilizadas.

Con todo ello, se muestra en la Figura 3.4 un posible esquema de la estructura interna de una neurona.

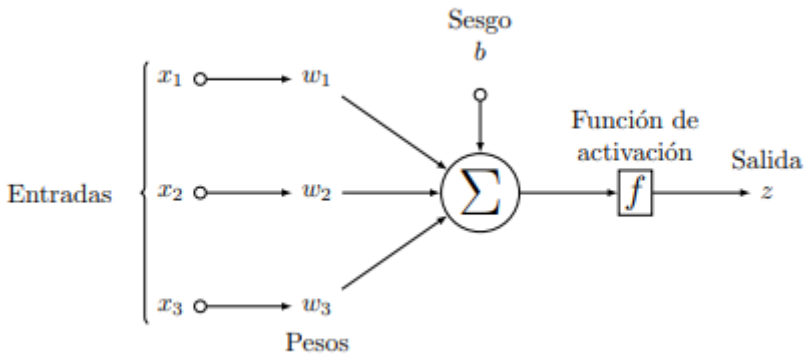


Figura 3.4: Representación gráfica de una neurona artificial

Por lo tanto, la salida de una neurona cuya entrada es un vector $\boldsymbol{x} = [x_1, x_2, x_3, ...x_N]$ se expresa de la siguiente manera:

$$z = f_{act}\left(\sum_{i=1}^{N} w_i x_i + b\right) \tag{3.2}$$

3.1.1 Parámetros e hiper-parámetros

Tanto en los modelos de ML como de DL existen dos tipos de parámetros. Por un lado, los propiamente denominados parámetros, los cuales de deben inicializar y actualizar durante el proceso de aprendizaje de la red neuronal. Por otro lado, los hiper-parámetros, que se inicializan previamente al inicio del entrenamiento del modelo [43].

Dentro del grupo denominado parámetros, se encuentran básicamente los pesos sinápticos y el sesgo previamente explicados. Mientras que los pesos son números reales que indican la pendiente de la recta, el sesgo representa la desviación de las predicciones con respecto del valor de salida deseado.

A su vez, según Yang y Shami [52], el segundo grupo, los conocidos como hiper-parámetros, se divide en dos categorías: los relacionados con la construcción de un modelo de DL, llamados hiper-parámetros de diseño de modelo; y los hiper-parámetros de optimización, que, como su propio nombre indica, se encargan de la optimización y el proceso de entrenamiento del modelo.

Dentro del grupo de los hiper-parámetros de diseño de modelo, destacan, entre otros, el número de capas ocultas, el número de neuronas por capa, la función de coste, la función de activación y el tipo de optimizador. Como se ha comentado anteriormente y se observa en la 3.3, las redes neuronales presentan como mínimo tres capas, donde dos de ellas pertenecen a la de entrada y de salida, pudiendo existir tantas capas ocultas como se desee. Precisamente es este hiper-parámetro, el número de capas ocultas de la red, junto al número de neuronas por capa, el que determina la profundidad de una red neuronal. Cuanto mayores sean dichos números, más compleja será la arquitectura de la red y datos de mayor tamaño podrá manejar.

Asimismo, a la hora de definir la arquitectura de una red neuronal, otro hiper-parámetro de diseño de modelo a tener muy en cuenta es el tipo de función de coste a emplear, ya que existen diferentes en función del tipo de problema al que haya que hacer frente. Entre las más utilizadas están la entropía cruzada binaria (del inglés *binary cross-entropy*) para los problemas de clasificación o la entropía cruzada multicategoría (del inglés *multi-class cross-entropy*) para los problemas de clasificación multivariable. Asimismo, para los problemas de regresión existen también el *L1 loss* o error medio absoluto (MAE, del inglés *Mean Absolute Error*), el *L2 loss* o error medio cuadrático (MSE, del inglés *Mean Squared Error*) y la raíz del error cuadrático medio (RMSE, del inglés *Root Mean Squared Error*). Las expresiones de estas tres últimas funciones vienen representadas en las ecuaciones 3.3, 3.4 y 3.5 respectivamente.

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m}|h(x^{(i)}) - y^{(i)},| \tag{3.3}$$

donde m es el número de muestras, $x^{(i)}$ es la muestra i del conjunto de datos, $h(x^{(i)})$ es la predicción para la muestra i, y $y^{(i)}$ es la etiqueta de la muestra i.

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2, \tag{3.4}$$

donde m es el número de muestras, $y^{(i)}$ es la etiqueta de la muestra i, y $\hat{y}^{(i)}$ es la etiqueta predicha para la muestra i.

$$\text{RMSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}, \tag{3.5}$$

donde m es el número de muestras, $x^{(i)}$ es la muestra i del conjunto de datos, $h(x^{(i)})$ es la predicc

Otro aspecto a tener en cuenta a la hora del diseño de la red neuronal es la función de activación escogida, la cual se emplea para la propagación de la salida de una neurona hacia delante. Entre dichas funciones, destacan la función lineal (ecuación 3.6), la sigmoidal (ecuación 3.7), la tangente hiperbólica (ecuación 3.8), la softmax (ecuación 3.9) o la unidad lineal rectificada (ReLU, del inglés *Rectified Linear Unit*) (ecuación 3.10), cuyas expresiones se muestran a continuación.

$$f(x) = x \tag{3.6}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.7}$$

$$f(x) = \tanh(x) \tag{3.8}$$

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad \text{para } i = 1, \ldots, n \tag{3.9}$$

$$f(x) = \begin{cases} x, & \text{si } x > 0 \\ 0, & \text{si } x \leq 0 \end{cases} \tag{3.10}$$

Por último, dentro del grupo de los hiper-parámetros de diseño de modelo, se debe seleccionar el tipo de optimizador a utilizar. La optimización consiste en minimizar o maximizar la función de coste correspondiente alterando el valor de la variable x. Entre los optimizadores más empleados se encuentran el gradiente de descenso estocástico (SGD, del inglés *Stochastic Gradient Descent*), la estimación de momento adaptativo (ADAM, del inglés *Adaptive Moment Estimation*), y la propagación de la raíz cuadrática media (RMSProp, del inglés *Root Mean Square Propagation*).

Dentro de los hiper-parámetros de optimización, el más importante es la tasa de aprendizaje (del inglés *learning rate*), el cual determina el tamaño de paso de cada

iteración para permitir que la función de coste converja. Si la tasa de aprendizaje es muy grande, el proceso de aprendizaje se acelera, pero puede provocar que el gradiente oscile sobre un mínimo local o que no llegue a converger; mientras que una tasa de aprendizaje muy pequeña provoca un incremento del tiempo de aprendizaje. Para prevenir este problema conocido como sobreajuste (del inglés *overfitting*, se utiliza una técnica conocida como *dropout*, la cual se encarga de eliminar en algunas iteraciones del entrenamiento ciertas neuronas y conexiones entre ellas. Dentro de este grupo existe también el hiper-parámetro *mini-batch size* que representa el número de muestras procesadas previas a la actualización del modo, así como el número de épocas el cual indica el número de veces que se itera sobre el conjunto de datos. En ocasiones ocurre que el error de validación no cambia durante varias épocas consecutivas, no generando beneficio. Esto se puede evitar determinando el hiperparámetro *early stopping*, el cual permite terminar antes la fase de entrenamiento si esto ocurre.

3.1.2 Tipos y características de las ANN

Este apartado tiene objetivo conocer algunas de las redes neuronales más utilizadas dentro de las técnicas de DL. Estas redes se pueden dividir en varías categorías según su estructura y función, lo que les permite abordar una amplia gama de tareas de procesamiento de datos.

- El perceptrón multicapa (MLP, del inglés *Multilayer Perceptron* es una estructura que consta de al menos tres capas: la capa de entrada, una o varias capas ocultas y la capa de salida. Como se puede observar en la Figura 3.5a, cada neurona de la red está conectada a todas las neuronas de la capa siguiente, lo que facilita la propagación hacia adelante de la información. Se pueden utilizar las funciones de activicación sigmoidal o ReLU anteriormente explicadas para introducir no linealidad en el modelo. Este tipo de arquitecutra se suele utilizar en tareas de clasificación y regresión.

- Las redes neuronales convolucionales (CNN, del inglés *Convolutional Neural Network* son un tipo de ANN compuestas como mínimo por una capa convolucional. Al utilizar capas tanto de convolución como de *pooling* (Figura 3.5b), consiguen reducir la dimensionalidad de la red, disminuyendo así el número de parámetros necesarios. Están específicamente diseñadas para procesar datos con estructura espacial como imágenes, por lo que son especialmente adecuadas para tareas de visión por computadora como reconocimiento de imágenes, segmentación o detección de objetos.

- Las redes neuronales recurrentes (RNN, del inglés *Recurrent Neural Network* presentan una estructura diferente al resto (Figura 3.5c), ya que tienen un estado interno creado con los datos de entrada empleados por la red, el cual

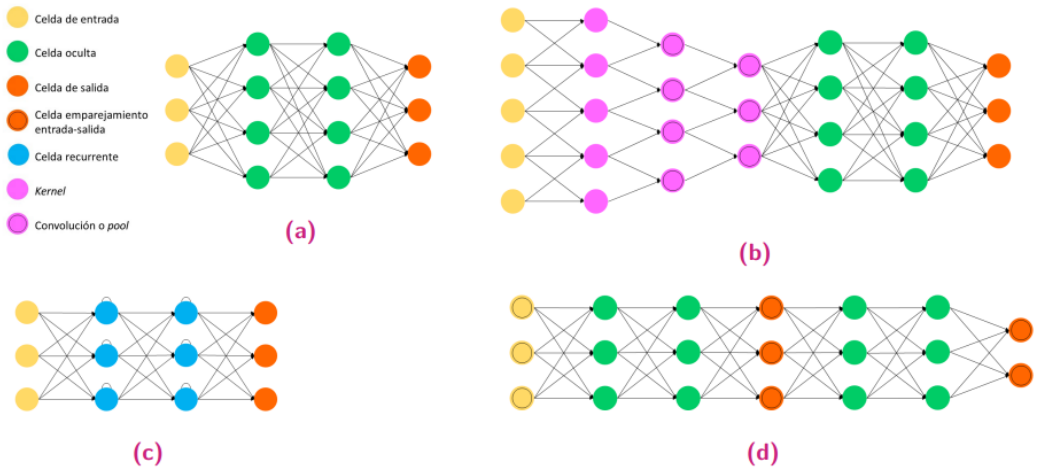


Figura 3.5: Diagrama de las arquitecturas de los principales tipos de redes neuronales: a) MLP, b) CNN, c) RNN y d) GAN

se va modificando en cada iteración para los nuevos datos entrantes. De esta manera, proporciona como salida una combinación entre dicho estado interno y las nuevas entradas. Este tipo de arquitectura las hace ideales para trabajar con datos secuenciales, siendo muy utilizadas para tareas de procesamiento de lenguaje natural (NLP, del inglés *Natural Language Processing*), generación de texto y análisis de series.

- Las redes neuronales generativas adversarias (GAN, del inglés *Generative Adversarial Networks*, como se puede observar en la Figura 3.5d, son una estructura dual compuesta por dos modelos: un modelo generativo y un modelo discriminativo. Estos dos modelos conviven y, mientras el generador produce muestras de datos sintéticas a partir de una distribución de ruido, el discriminador evalúa si una muestra es real o generada por el generador. A través de un proceso de entrenamiento, el generador mejora su capacidad para engañar al discriminador, y este último mejora su capacidad para distinguir lo real de lo falso. Esta estructura es utilizada para generar imágenes realistas, arte generativo y diversas aplicaciones de mejora de imágenes y vídeos.

Entre las estructuras mencionadas, la más eficaz para tareas de detección de objetos son las CNN y serán las utilizadas en el desarrollo de este proyecto. Como se ha comentado, este tipo de redes, además de la capa de entrada y salida, cuentan con diversas capas de convolución y *pooling*.

La operación de convolución implica la detección y el aprendizaje de patrones locales en ventanas bidimensionales dentro de una imagen. Una vez que se ha identificado un patrón específico en una imagen, es posible reconocer esa característica en cualquier parte de la misma. En su etapa inicial, una capa convolucional tiene la capacidad de aprender patrones simples como bordes, colores y líneas. Posteriormente, capas convolucionales subsiguientes utilizan estos patrones como base para

aprender jerarquías de patrones más complejos. A través de este proceso, al emplear múltiples capas de convolución, la red neuronal puede comprender patrones altamente sofisticados. La Figura 3.6 muestra un ejemplo de convolución y el resultado obtenido al aplicar el filtro correspondiente.

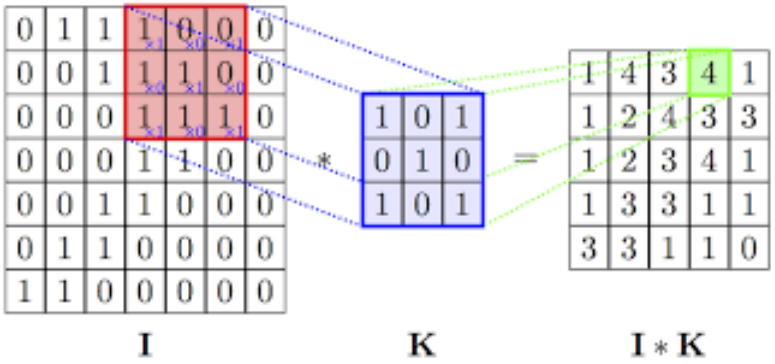


Figura 3.6: Ejemplo de operación de convolución [4]

En ciertas redes neuronales, es muy común utilizar capas de pooling después de una serie de capas convolucionales. En términos generales, la operación de *pooling* simplifica la información recopilada por la capa convolucional y crea una versión más compacta de la misma. Para lograrlo, esta capa divide la capa convolucional en pequeñas regiones de igual tamaño, lo que reduce el número de conexiones hacia las capas siguientes. A diferencia de las capas convolucionales, las capas de *pooling* no realizan aprendizaje por sí mismas, sino que reducen la cantidad de parámetros que deben aprenderse en las capas subsiguientes. Como se ilustra en la Figura 3.7, existen varios tipos de *pooling*, siendo los más comunes el *max-pooling*, que selecciona el valor máximo de la región, y el *average-pooling*, que calcula el valor promedio de la región.

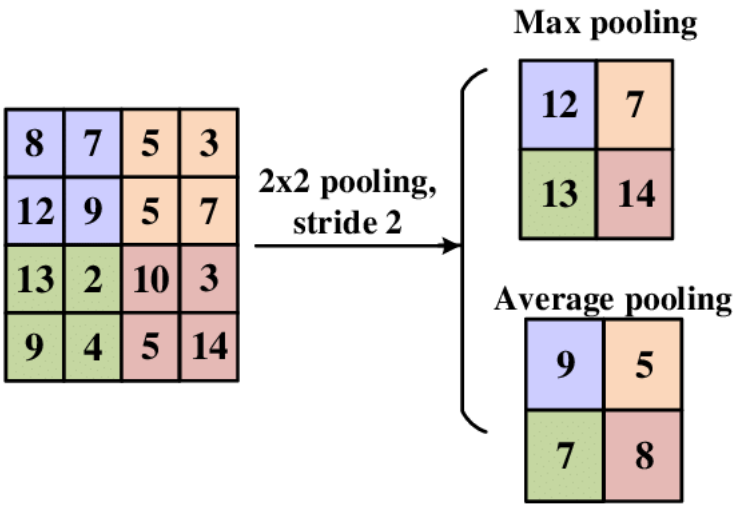


Figura 3.7: Ejemplo de distintas operaciones de *pooling* [5]

[4] Fuente: https://www.diegocalvo.es/red-neuronal-convolucional/
[5] Fuente: `https://www.modeldifferently.com/2021/10/image_classification/`

3.2 Realidad Aumentada como tecnología actual

La Realidad Aumentada (RA) es una tecnología innovadora que ha transformado la forma de percibir e interactuar con el mundo. Dicha tecnología se puede definir de manera sencilla como un conjunto de técnicas que permiten la aplicación de elementos virtuales sobre una representación del mundo físico real. De esta manera, a través de dispositivos tecnológicos, como smartphones, gafas inteligentes o tablets, se consigue una versión ampliada del mundo físico superpuesta con contenido digital (gráficos, información, objetos 3D...), lo cual permite obtener información del entorno, más allá de la que se puede alcanzar mediante los cinco sentidos físicos que posee el ser humano [1].

Es importante no confundir esta tecnología con la Realidad Virtual. Mientras que esta última crea un entorno completamente ficticio y proporciona un alto nivel de inmersión al sumergirse completamente en un mundo virtual, la RA enrique la experiencia del mundo real al superponer elementos virtuales sobre el mismo. Dicho de otra manera, la Realidad Virtual lleva al usuario a un escenario completamente nuevo, mientras que la RA combina el mundo físico real con información virtual. Ambas tecnologías tienen un gran potencial, pero la Realidad Virtual se utiliza principalmente en aplicaciones de entretenimiento, mientras que la RA tiene un rango mayor de aplicaciones, desde el entretenimiento hasta la educación.

Tal es el impacto generado por esta tecnología que ha tenido su efecto en el mundo industrial también. En el contexto industrial, la RA ha demostrado ser una herramienta con un potencial extraordinario y es por ello por lo que, en los últimos tiempos, son unos cuantos los ámbitos industriales en los que se está introduciendo como herramienta de apoyo [2]. Su capacidad para combinar el mundo físico con datos digitales en tiempo real ha revolucionado la toma de decisiones y la eficiencia operativa en entornos industriales.

Entre sus principales ventajas dentro del mundo industrial destacan el acceso a información crítica de manera inmediata, la mejora la capacitación de los trabajadores mediante simulaciones interactivas, la asistencia técnica remota al superponer instrucciones precisas y la reducción de los tiempos de mantenimiento. De esta manera, la RA ha demostrado su capacidad para aumentar la productividad, reducir los errores y mejorar la seguridad en el lugar de trabajo. Estos avances han transformado la forma en que las empresas abordan los desafíos y oportunidades, abriendo nuevas posibilidades para la mejora de procesos, la toma de decisiones basada en datos y la optimización de la eficiencia.

3.2.1 Historia de la RA

Para entender como se ha llegado a la situación actual, es necesario conocer el comienzo de esta tecnología y cuál ha sido la evolución que ha tenido hasta llegar al día de hoy.

Aunque se ha afirmado ampliamente que la RA es una innovación reciente, la realidad es que aplicaciones similares ya existían desde hace un sigo, en concreto, desde el año 1901. Fue en este año cuando Frank L. Baum, escritor estadounidense, desarrolló un dispositivo llamado *Character Maker*, el cual podría ser considerado como el prototipo de lo que hoy en día se conoce como Realidad Aumentada. Este dispositivo se basaba en un gran visor electrónico que era capaz de superponer información sobre las personas enfocadas a través de él [33].

Poco más de medio siglo después, sobre la década de los 60, hubo dos nuevas aportaciones. Por una parte, el científico estadounidense Ivan Sutherland desarrolló un dispositivo conocido como *Sword of Damocles*, el cual permitía a los usuarios ver objetos virtuales superpuestos en su entorno [41]. Como se observa en la Figura 3.8, este dispositivo era bastante aparatoso lo que lo hacía útil únicamente en ciertos entornos. Por otra parte, al mismo tiempo, el famoso cineasta e inventor Morton Hellig creó un aparato al que llamó *Sensorama*. Este dispositivo simulaba una experiencia inmersiva de realidad virtual en 3D mediante el uso de efectos sonoros como el viento, las vibraciones del asiento o el sonido envolvente. Este invento, el cual se ilustra en la Figura 3.9, permitía a los usuarios pasear por San Francisco viendo imágenes del lugar, con elementos adicionales que hacían la experiencia lo más realista posible [1].

Figura 3.8: Dispositivo conocido como *Sword of Damocles* desarrollado por el científico estadounidense Ivan Sutherland [6]

[6]Fuente: `https://www.dsource.in/course/virtual-reality-introduction/evolution-vr/sword-damocles-head-mounted-display`

Figura 3.9: *Sensorama*, aparato diseñado por el inventor Morton Hellig [7]

Sin embargo, no fue hasta la década de los 90 que se introdujo por primera ver el término de *«Realidad Aumentada»*. En esta época, el ingeniero Boeing Tom Caudell diseñó unos dispositivos que servían para describir un sistema de entrenamiento de alambrado de cables. Sin embargo, son muchos los autores que atribuyen el hito de acuñar el término a Louis Rosenberg. En el año 1992, este tecnólogo estadounidense creó el primer sistema real de RA llamado *Virtual Fixtures*. Este sistema, el cual se puede observar en la Figura 3.10, proyectaba un brazo robótico al usuario como guía para realizar tareas específicas [41].

Figura 3.10: Primer sistema real de RA, *Virtual Fixtures*, creado por el tecnólogo Louis Rosenberg [8]

[7] Fuente: `https://www.xataka.com/historia-tecnologica/el-primer-simulador-vr-de-la-historia-tenia-form`
[8] Fuente: `https://spectrum.ieee.org/history-of-augmented-reality`

El siguiente gran paso en el desarrollo de esta tecnología fue dado por la Universidad de Columbia (Nueva York, EEUU). En esta universidad, un pequeño grupo de científicos desarolló un sistema llamado KARMA, el cual contaba con un casco de realidad virtual (HMD, del inglés *Head Mounted Display*). Este dispositivo proyectaba una imagen tridimensional que guiaba al usuario mediante el uso de una impresora, siendo uno de sus principales objetivo el de eliminar los manuales de usuario.

Asimismo, a principios de este siglo, la aparición de ARToolkit, una biblioteca de código abierto para la creación de aplicaciones de RA, supuso un gran avance en la tecnología, desencadenando en una explosión de desarrollo en este campo. Tal fue dicha explosión que, pasado unos años, empresas punteras internacionalmente comenzaron a invertir mucho dinero en este campo. Entre otras muchas empresas, Google anunció unas de las primeras gafas de RA llamadas *Google Glass*, al mismo tiempo que Microsoft presentaba las gafas *HoloLens*. Ambos dispositivos generaron un gran interés y demostraron el potencial de esta tecnología.

Uno de los acontecimientos que supuso el descubrimiento de la RA para el gran público fue el lanzamiento del *Pokemon GO* en 2016. Este juego utilizaba la ubicación y la cámara del smartphone, como se observa en la Figura 3.11, para superponer personajes de Pokemon en el mundo real, demostrando el atractivo de esta tecnología para un entorno más lúdico.

Figura 3.11: Juego de entretenimiento *Pokemon GO* [9]

A medida que han ido pasando los años, los avances tecnológicos han sido cada vez mayores, permitiendo así aplicaciones de RA cada vez más precisas y realistas como las que existen hoy en la actualidad. Esta tecnología ha conseguido expandirse en una gran variedad de sectores a día de hoy, desde la medicina hasta la educación y el entretenimiento. Su adopción se ha acelerado gracias a las constantes mejoras en dispositivos y algoritmos de detección de objetos. Además, la RA se ha convertido en una herramienta valiosa para la formación, la asistencia técnica, la visualización de datos y la toma de decisiones basada en datos en entornos industriales y empresariales.

[9] Fuente: `https://www.3djuegosguias.com/simulacion/pokemon-go-dia-comunidad-clasico-enero-2022`

3.2.2 Detección de objetos en aplicaciones de RA

El proceso de localizar y clasificar un objeto en una serie de imágenes o en tiempo real mediante diferentes técnicas se conoce como detección de objetos. Este fenómeno, desde hace unos cuantos años y especialmente en los últimos tiempos, ha sido uno de los retos principales en distintos ámbitos de la industria [44].

Además, con la llegada del mundo digitalizado y la Industria 4.0, la calidad de la información ha mejorado de manera notable, dando lugar a una enorme generación de datos y una gran diversidad de objetos. Esto ha provocado que la detección de objetos sea aún más complicada. Es por ello por lo que una gran variedad de tecnologías se ha involucrado para intentar ofrecer soluciones novedosas, investigando acerca de diferentes maneras que permitan realizar dicha detección.

La Realidad Aumentada es un ejemplo destacado de este tipo de tecnologías, la cual se ha convertido en una de las tendencias tecnológicas más populares de la era actual [1]. Una correcta detección de los objetos es vital para poder vivir una experiencia satisfactoria en estos entornos, ya que todo sistema de RA debe identificar con precisión el entorno real que lo rodea y todos los objetos que lo componen para tener un rendimiento óptimo. La mayoría de estas tecnologías deben reconocer un mapa espacial en 3D de los componentes a su alrededor escaneando el mundo real. Es decir, dicho de otra manera, cualquier sistema que se base en esta tecnología debe ser capaz de realizar una detección de objetos del mundo real lo más precisa, rápida y eficaz posible en todo momento.

Los primeros pasos que se dieron en este ámbito utilizaban principalmente técnicas basadas en marcadores para identificar el contenido del mundo real en los sistemas de RA [31]. Estos marcadores generalmente son patrones visuales específicos y predefinidos como los que se pueden observan en la Figura 3.12, (códigos de barras, códigos QR, patrones geométricos...). En esta técnica, se asigna cada uno de estos marcadores a objetos o entornos del mundo real para poder ser detectados por una cámara o sensor de imágenes. Al ser conocida su posición, éstos son tomados como puntos de referencia y, cada vez que la cámara capta uno de ellos, el sistema de RA es capaz de calcular su posición y orientación con precisión. La principal ventaja de esta técnica es su precisión y estabilidad en la detección. Sin embargo, su principal limitación es la necesidad de tener marcadores físicos presentes en el entorno, lo que puede ser restrictivo en aplicaciones donde se busca una experiencia de RA más natural y sin restricciones. Además, existen situaciones concretas en donde está técnica suele ser imprecisa, especialmente cuando hay una distancia considerable entre la cámara y el objeto real, o un obstáculo entre ellos. Asimismo, es imprescindible que los marcadores no reflejen la luz, así como que el blanco y negro tengan un fuerte contraste [11].

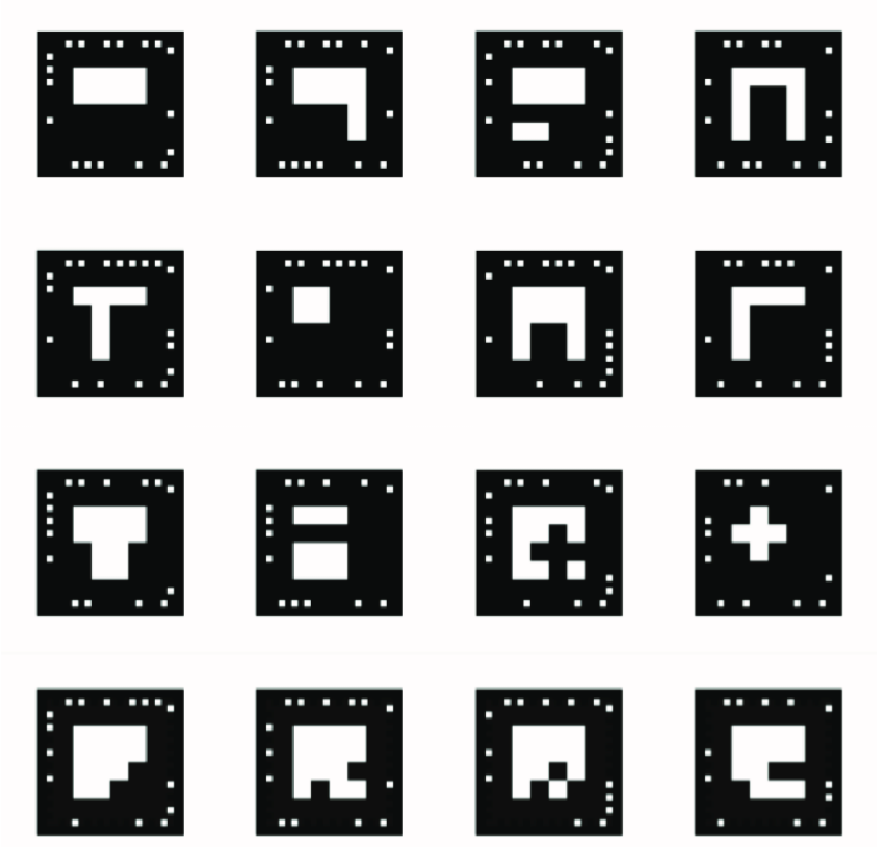


Figura 3.12: Ejemplos de marcadores utilizados en aplicaciones de RA [10]

Para hacer frente a estos inconvenientes, surge una nueva técnica: el seguimiento sin marcadores o en inglés *markerless tracking*, la cual se basa en la identificación y seguimiento de características naturales de la escena, como puntos de interés visuales o características distintivas, sin la necesidad de marcadores predefinidos. Estos sistemas de seguimiento sin marcadores utilizan algoritmos de visión por computadora para reconocer y rastrear estas características en tiempo real, lo que les permite determinar la posición y orientación de la cámara y, por lo tanto, la posición de objetos virtuales en relación con la escena del mundo real. La ventaja principal que ofrece el seguimiento sin marcadores es su capacidad para detectar objetos en entornos no preparados, lo que brinda una mayor flexibilidad y naturalidad en las aplicaciones de RA en comparación con la técnica basada en marcadores. No obstante, suelen ser más propensos a errores y menos precisos. Además, este tipo de técnicas necesita reconocer una superficie plana texturizada para aumentar el contenido digital de forma eficaz [14].

Estas limitaciones que ofrecen las técnicas basadas en marcadores y en el seguimiento sin marcadores pueden ser solucionadas utilizando técnicas basadas en *Deep Learning*, las cuales han supuesto una gran revolución en el mundo de la RA, proporcionando

[10] Fuente: `https://www.researchgate.net/figure/Figura-3-Algunos-marcadores-de-realidad-aumentada-utilizados_fig1_334107261`

detecciones más rápidas y precisas. Como en los otros casos, mediante estas técnicas se pretende reconocer y localizar la mayor cantidad de objetos posibles dentro de un fotograma específico. Este tipo de técnicas permite identificar los objetos físicos existentes en una imagen o vídeo, así como su ubicación (localización de objetos), además de ser capaz de reconocer la categoría a la que pertenecen (clasificación de objetos). Por lo tanto, usando DL, el algoritmo es capaz no solo de localizar el objeto, sino también de clasificarlo dentro de un grupo con características parecidas [20].

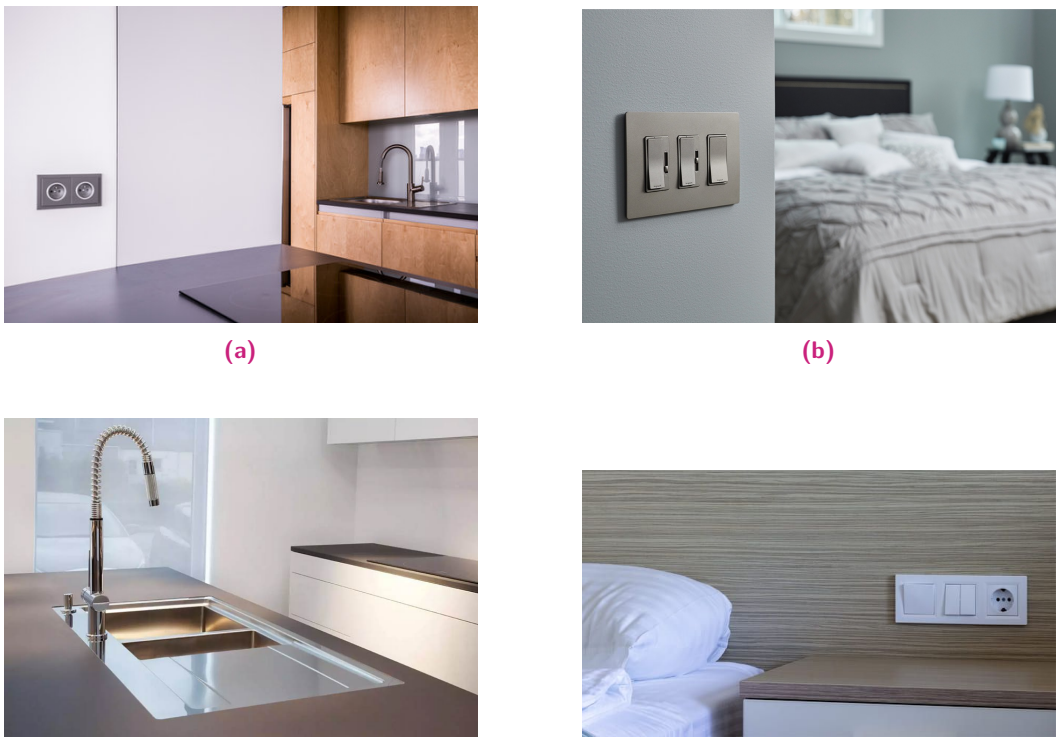
Son muchos los estudios realizados sobre la posibilidad de utilizar este tipo de técnicas para la detección de objetos en distintas aplicaciones [12, 22]. Sin embargo, aunque en los últimos años han aparecido estudios relacionados con la RA [29, 33, 46, 48], apenas se ha investigado acerca de la viabilidad de utilizar algoritmos basados en DL para detección de objetos dentro de los dispositivos de RA. Es por ello que en los siguientes apartados se desarrolla el estudio realizado acerca de las distintas técnicas de DL, analizando los resultados que ofrece cada una de ellas, así como sus ventajas y desventajas.

4 Desarrollo de la solución

El trabajo, como se ha comentado en apartados anteriores, se centra principalmente en analizar distintas técnicas basadas en DL; en concreto, en distintas arquitecturas de CNN. A lo largo de los siguientes apartados se detallan los pasos seguidos para realizar dicho estudio, los cuales se resumen en la generación del *Dataset* correspondiente y la simulación con cada una de las arquitecturas.

4.1 Preparación del *Dataset*

El primer paso para poder realizar entrenamientos de los modelos y que éstos ofrezcan buenos resultados es contar con un *dataset* representativo y de calidad. Este aspecto es fundamental en cualquier técnica de DL que se quiera implementar y debe contar con un número de imágenes adecuado, lo más diversas posibles y equilibradas en cuanto a las clases existentes. En este caso, al querer utilizar estos algoritmos en aplicaciones de RA relacionados con la construcción, se ha optado por utilizar como reto inicial imágenes que contengan elementos que pueden aparecer en dichas tareas: enchufes e interruptores.



(a) (b) (c) (d)

Figura 4.1: Tipos de imágenes del *dataset*

Como se observa en la Figura 4.1, existen cuatro categorías diferentes dentro del *dataset*: imágenes que únicamente contienen enchufe, imágenes con interruptores, imágenes que no tienen ninguno de estos dos elementos e imágenes que contienen ambos elementos. Para generar dicho *dataset* se ha contado con la ayuda de compañeros de la empresa que han sacado fotos en sus respectivas casas. Con éstas y alguna imagen descargada de Internet, se ha generado el conjunto de datos que se tomará como punto de partida. En la Tabla 4.1 se puede observar el número de imágenes que componen cada una de las categorías.

	Imágenes de enchufes	**Imágenes de interruptores**	**Imágenes sin enchufes ni interruptores**	**Imágenes con enchufes e interruptores**
Número de imágenes	124	115	138	93

Tabla 4.1: Número de imágenes por cada categoría del *dataset*

Una vez definidas las imágenes que formarán el conjunto de datos, se debe realizar el etiquetado de cada una de ellas, en este caso, de manera manual mediante la aplicación CVAT (del inglés, *Computer Vision Anotation Tool*). Esta herramienta de código abierto, diseñada para facilitar la anotación de objetos y la generación de conjuntos de datos etiquetados, es utilizada generalmente en el campo de la visión por computadora y el procesamiento de imágenes. Por ello, son muchos los usuarios que lo utilizan en tareas de aprendizaje automático y entrenamiento de modelos de IA. En este caso, al tratarse de una tarea de detección de objetos, se debe indicar mediante la opción *polygon* donde se encuentran aquellos objetos que se desean detectar medainte el algoritmo (en este caso concreto, se deben indicar la posición de los enchufes e interruptores).

Tras completar el etiquetado de todas las imágenes, esta herramienta permite exportar estos datos etiquetados en formatos comunes como XML (del inglés, *Extensible Markup Language*), JSON (del inglés, *JavaScript Object Notation*), YOLO (del inglés, *You Only Look Once*), COCO (del inglés, *Common Objects in Context*), entre otros. Para este trabajo se ha optado por utilizar el formato COCO [27] al ser el generalmente utilizado en tareas similares en la empresa.

Para más adelante poder implementar el código con éxito, es crucial conocer la estructura que presenta el formato COCO. Este generalmente se organiza en cinco apartados clave en su archivo principal:

- **'licenses':** contiene información sobre las licencias bajo las cuales se distribuyen las imágenes y los datos anotados. Incluye características como el nombre de la licencia, el identificador de la licencia y la URL de la licencia.

- **'info':** se proporciona información general sobre el conjunto de datos, como el nombre del conjunto de datos, el año de creación, el descripción, el autor y la URL del conjunto de datos.

- **'categories':** se enumeran todas las categorías presentes en el *dataset*. Cada una de esas categorías se describe mediante un identificador numérico único, un nombre y, en ocasiones, una *supercategory*. En este caso, como muestra el siguiente trozo de código, únicamente existen dos categorías: enchufe e interruptor.

```
        "categories": [
        {
            "id": 1,
            "name": "enchufe",
            "supercategory": ""
        },
        {
            "id": 2,
            "name": "interruptor",
            "supercategory": ""
        }
    ]
```

- **'images':** contiene información detallada sobre cada imagen en el conjunto de datos. Esto incluye un identificador único para cada imagen, el nombre del archivo de imagen, las dimensiones (ancho y alto) y cualquier dato adicional relacionado con dicha imagen.

- **'annotations':** se almacenan las anotaciones que describen la ubicación y las propiedades de los objetos detectados en las imágenes. Cada anotación se asocia a una imagen mediante su identificador de imagen (*"image_id"*) y se describe con detalles como coordenadas de caja delimitadora (*bbox*), segmentaciones, área...

De estos apartados, la información más relevante viene recogida en los dos últimos, por lo que es vital entender correctamente cómo se organiza. Se utiliza la imagen de la Figura 4.2 como ejemplo para explicar estos dos apartados. Como se puede observar en dicha imagen, contiene un enchufes y un interruptor, por lo que hay un total de dos objetos a detectar.

En primer lugar, se va a analizar cómo se representa esta imagen dentro del apartado 'images'. Como se ha mencionado en el punto correspondiente, en este apartado se asigna un identificador único para cada imagen mediante el parámetro *"id"*. Para la imagen de la Figura 4.2 le corresponde el número 178, como se puede observar

Figura 4.2: Imagen ejemplo del dataset para entender los apartados 'images' y 'annotations'

en el código mostrado a continuación. Asimismo, este apartado define el ancho y la altura de la imagen mediante los parámetros *"width"* y *"height"* respectivamente, así como el nombre asociado a dicha imagen en el parámetro *"file_name"*

```
1    {
2            "id": 178,
3            "width": 4000,
4            "height": 3000,
5            "file_name": "train44.jpg",
6            "license": 0,
7            "flickr_url": "",
8            "coco_url": "",
9            "date_captured": 0
10       },
```

Mediante el identificador se pueden comprobar cuáles son las anotaciones asociadas a dicha imagen dentro del apartado 'annotations'. Se muestra a continuación el trozo de código de este apartado que corresponde a la imagen ejemplo.

```
1    {
2            "id": 292,
3            "image_id": 178,
4            "category_id": 2,
5            "segmentation": [
6                [
7                    1816.62,
```

```
                    573.17,
                    1822.27,
                    725.57,
                    1972.78,
                    729.33,
                    1976.55,
                    571.29
                ]
            ],
            "area": 24111,
            "bbox": [
                1816.62,
                571.29,
                159.93,
                158.04
            ],
            "iscrowd": 0,
            "attributes": {
                "occluded": false
            }
        },
        {
            "id": 293,
            "image_id": 178,
            "category_id": 1,
            "segmentation": [
                [
                    1825.3,
                    2318.92,
                    1827.91,
                    2419.52,
                    1937.66,
                    2416.91,
                    1937.66,
                    2317.61
                ]
            ],
            "area": 11118,
            "bbox": [
                1825.3,
                2317.61,
                112.36,
```

```
                101.91
            ],
            "iscrowd": 0,
            "attributes": {
                "occluded": false
            }
        },
```

En este código, se puede observar que la imagen con el número de identificación *"image_id"* 178 efectivamente tiene dos objetos a detectar, representados mediante el parámetro *"id"*, asociado al 292 y 293. Para cada uno de esos objetos detectados son importantes los parámetros *"image_id"* y *"bbox"*. El primero de ellos indica la categoría a la que pertenece el objeto; en este caso, 1 para los enchufes y 2 para los interruptores. Asimismo, el parámetro *"bbox"* indica las coordenadas de la caja delimitadora o *bounding box* que engloba al objeto. Como se puede observar, está compuesto por un vector de cuatro elementos como el que se muestra en la siguiente ecuación:

$$bbox = [x, y, w, h], \tag{4.1}$$

donde x e y son las coordenadas de la esquina superior izquierda de la caja, w el ancho y h el alto.

Llegados a este punto donde se conoce perfectamente la forma en la que se organizan las anotaciones, se deben procesar correctamente todas las imágenes. Además de realizar operaciones de normalización o aumento de datos que se explicarán más adelante, se ha optado por realizar un cambio de tamaño o *resize* para que las imágenes de entrada a la red tengan las mismas dimensiones. En este caso, se ha optado por utilizar imágenes de $650 * 650$, realizando dicha operación mediante la librería *albumentations*, la cual se encarga de reescalar también todas las cajas delimitadoras.

Por último, se debe decidir qué tipo de división se va a realizar del *dataset*. De todo el conjunto de datos, algunas imágenes irán destinadas a la fase de entrenamiento, otras a la de validación y otras a la fase de prueba. Para iniciar los primeros entrenamientos de los modelos se utilizan los valores mostrados en la Tabla 4.2.

train	***validation***	***test***
70 %	10 %	20 %

Tabla 4.2: División del conjunto de datos en entrenamiento, validación y prueba

4.2 Implementación de los modelos de detección de objetos

Esta sección tiene como objetivo analizar la arquitectura de cada uno de los modelos utilizados en este trabajo, así como mencionar aspectos importantes que se deben tener en cuenta a la hora de su implementación. A lo largo de los siguientes apartados se describe cada uno de los modelos desarrollados por separado, prestando especial interés a sus limitaciones y arquitecturas.

4.2.1 *Convolutional nueral network (CNN)*

Las redes neuronales convolucionales, como se comenta en el apartado 3.1.2, son los algoritmos de DL más sencillos y más utilizados para la detección de objetos. En este tipo de modelos, en primer lugar, las imágenes se divide en piezas separadas, tomando el algoritmo cada uno de estos fragmentos como entrada y, tras varias capas de convolución y *pooling*, genera las clases de objetos. El problema principal de este tipo de enfoques es que los objetos pueden cubrir diferentes proporciones de fotogramas. Por lo tanto, se requiere dividir la imagen en muchas regiones, lo que supone un tiempo de cálculo y un coste computacional muy elevado [14].

Para hacer frente a este problema, se necesita un modelo más rápido que reduzca el número de regiones obtenidas. Para ello, se generaron redes neuronales convolucionales basadas en regiones (R-CNNN, del inglés *Region-based convolutional neural network*) que trabajan sobre un número determinado de regiones. Este algoritmo extrae un grupo de regiones de interés (RoI, del inglés *Region of Interest* y comprueba si existe un objeto en esa región específica. Las *R-CNN*s parte de una red neuronal convolucional preentrenada y, a partir de ella, en función del número de clases que se deban detectar, debe reentrenarse la última capa. A continuación, tras remodelar las regiones para adaptarlas al tamaño de entrada de la red, se utilizan clasificadores típicos para determinar la categoría de cada objeto. Por último, se utilizan técnicas de regresión para asignar un cuadro delimitador a cada clase predicha [14]. La arquitectura descrita para redes *R-CNN*s se resume en la siguiente Figura 4.3

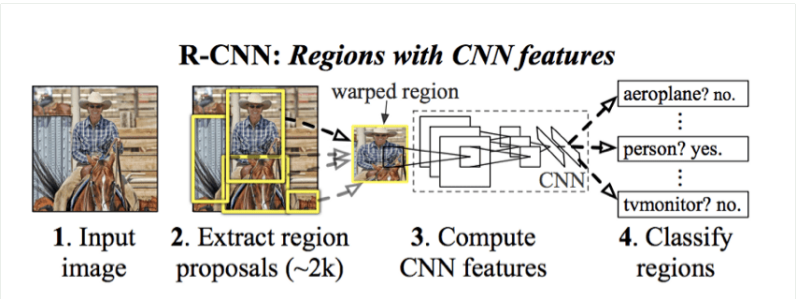


Figura 4.3: Resumen de las arquitecturas en *R-CNN*s [1]

[1]Fuente: `https://towardsdatascience.com/understanding-regions-with-cnn-features-r-cnn-ec69c15f8ea7`

Sin embargo, aunque las *R-CNN*s consiguen mejorar las prestaciones de las *CNN*s en cuanto a coste computacional, siguen siendo algo lentas en el cálculo. Para hacer frente a este inconveniente, modelos como el *Fast R-CNN* o el *Faster R-CNN* han sido desarrollados, siendo este último el primero con el que se realizan entrenamientos en este trabajo, utilizando para ello la clase proporcionada por PyTorch. La diferencia principal en cuanto a arquitectura entre estos dos modelos más rápidos es la inclusión de una red de propuesta de regiones (RPN, del inglés *Region Proposal Network*) en *Faster R-CNN*, lo que simplifica la generación de las RoIs y mejora significativamente la velocidad de detección de objetos.

4.2.1.1 *Faster R-CNN*

Antes de explicar las pruebas realizadas mediante los modelos de *Faster R-CNN*, es necesario conocer el tipo de arquitectura que presentan. Como se ha comentado en el apartado anterior, este modelo de detección de objetos se caracteriza por combinar las características de una RPN con las de una CNN [38]. Esta arquitectura está compuesta principalmente por los siguientes componentes, los cuales se pueden ver en la Figura 4.4:

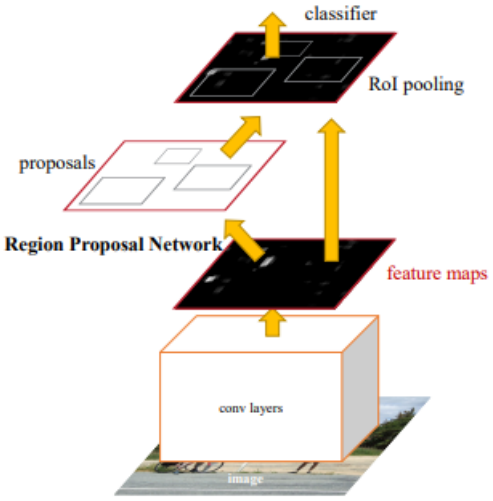


Figura 4.4: Arquitectura general de una *Faster R-CNN* [38]

- Un extractor de características CNN encargado de extraer las características más relevantes de la imagen de entrada. Generalmente se utiliza una arquitectura de CNN pre-entrenada, como VGG16, Resnet o Inception, la cual ha sido entrenada previamente en un conjunto de datos enorme para aprender características genéricas. La imagen de entrada se pasa a través de varias capas convolucionales y de agrupación para extraer características a diferentes niveles de resolución.

- Una RPN que opera en paralelo con el extractor de características, cuya tarea se basa en proponer regiones candidatas en la imagen que puedan contener

objetos de interés. Esta red genera una serie de *bounding boxes* con puntuaciones de confianza asociadas a ellas para indicar la probabilidad de que dicha región contenga objeto. Para generar esas regiones, suelen utilizan cajas de diferentes y tamaños relaciones de aspecto que se desplazan a lo largo de la cuadrícula de características generada por la CNN conocidas como *anchors*.

- Un módulo de detección. Una vez generadas las regiones propuestas, se aplica un proceso de filtrado para eliminar regiones duplicadas o regiones poco probables. Asimismo, se deben ajustar y refinar para mejorar la precisión de las ubicaciones de los objetos detectados. A continuación, dichas regiones se clasifican para determinar la clase de objetos presente en cada una de ellas, utilizando para ello capas completamente conectadas o *fully-connected layers*.

- Por último, como salida, este modelo genera una lista de cajas delimitadoras que representan los objetos detectados, junto con las etiquetas de clase correspondientes y las puntuaciones de confianza asociadas.

Para implementar este tipo de arquitectura se han utilizado dos de los modelos ofrecidos por Pytorch, dentro de su librería *torchvision*. Estos modelos, descritos en los siguientes puntos, se utilizan precismaente para instanciar modelos de *Faster R-CNN*, tanto con pesos preentrenados como sin ellos.

- *Faster R-CNN model with a ResNet-50-FPN backbone*, el cual se basa en el artículo [38] y se instancia mediante *torchvision.models.detection.fasterrcnn_resnet50_fpn*. Esta arquitectura de red combina la red ResNet-50 con la características de la Red de Pirámide de Características (FPN, del inglés *Feature Pyramid Network*). Por un lado, la ResNet-50 se caracteriza por ser una variante de la ResNet que utiliza 50 capas profundas y por su capacidad para aprender características profundas y ricas en detalles. Por otro lado, la FPN es una técnica que agrega características de múltiples escalas en la red, pudiendo combinar características de nivel bajo y alto de manera que se capturen objetos de diferentes tamaños en la imagen.

- *Faster R-CNN model with a MobileNetV3-Large FPN backbone*, el cual se instancia mediante *torchvision.models.detection.fasterrcnn_mobilenet_v3_large_fpn*. En este modelo, se utiliza el MobileNetV3-Large para extraer las características de la imagen de entrada y, al igual que en el modelo anterior, la FPN se encarga de detectar los objetos en una variedad de escalas. En este caso, la familia de redes neuronales MobileNetV3 es muy eficiente en términos computaciones y de recursos, por lo que este modelo es adecuado para dispositivos móviles y aplicaicones en tiempo real.

Ambos modelos han sido utilizados en los distintos entrenamientos realizados. Para ello, se han realizado distintas simulaciones donde se han modificado diversos

parámetros para ver su efecto en el rendimiento. El optimizador escogido para la mayoría de entrenamientos ha sido el de AdamW, descrito por el algoritmo 1, el cual actualiza el vector de gradiente y el gradiente cuadrático utilizando la media móvil exponencial. Los coeficientes β_1 y β_2 son los factores de olvido (*forgetting factors*) para los gradientes y los segundos momentos de los gradientes, respectivamente.

Algoritmo 1: Algoritmo AdamW

Entradas: $\gamma(lr)$, β_1, β_2(betas), θ_0(params), $f(\theta)$(objetivo), ϵ, λ(weight decay), ams-grad, maximize

Inicializar: $m_0 \leftarrow 0$ (primer momento), $v_0 \leftarrow 0$ (segundo momento)

for $t = 1$ **to** $\ldots$ **do**

  if *maximize* **then**

    $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$

  end

  else

    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

  end

  $\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\theta_{t-1}$;

  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$;

  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$;

  $m_{ct} \leftarrow m_t / (1-\beta_1^t)$;

  $v_{bt} \leftarrow v_t / (1-\beta_2^t)$;

  if *ams-grad* **then**

    $v_{bt}^{\max} \leftarrow \max(v_{bt}^{\max}, v_{bt})$;

    $\theta_t \leftarrow \theta_t - \gamma \frac{m_{ct}}{\sqrt{v_{bt}^{\max}} + \epsilon}$;

  end

  else

    $\theta_t \leftarrow \theta_t - \gamma \frac{m_{ct}}{\sqrt{v_{bt}} + \epsilon}$;

  end

end

Salida: θ_t

Asimiso, se han realizado pruebas modificando la función de coste, el valor de la tasa de aprendizaje, el número de épocas... También se ha comprobado el efecto generado por utilizar modelos preentrenados. Los resultados obtenidos en todos estos entrenamientos se desarrolla en el apartado 5 del presente documento.

4.2.2 *Single-shot-multi-box detector (SSD)*

El enfoque SSD se basa en una red convolucional *feedforward* que produce una colección de de cajas delimitadores de tamaño fijo y puntuaciones para la presencia de instancias de clases de objetos en esas cajas. Luego, se aplica un paso de supresión de no máximos para producir las detecciones finales [28]. Las primeras capas de la red están basadas en una arquitectura estándar utilizada para la clasificación de imágenes de alta calidad, pero se truncan antes de llegar a cualquier capa de clasificación. A esto se le denomina red base y, en este trabajo, se utilizará la red

VGG-16 como base, a pesar de poder utilizar muchas otras. En este punto, se agregan estructuras auxiliares a la red para producir detecciones, las cuales deben cumplir con las siguientes características clave:

- Mapas de características a múltiples escalas para la detección. Al agregar capas convolucionales al final de la red base, se consigue disminuir el tamaño de manera progresiva y esto, a diferencia de otros modelos, permiten hacer predicciones de detección en múltiples escalas [50].
- Predictores convolucionales para la detección. Cada capa de características agregada puede producir un conjunto fijo de predicciones de detección utilizando un conjunto de filtros convolucionales. Estos filtros se utilizan para predecir la presencia de una categoría específica o un desplazamiento de forma en relación con las coordenadas de una caja predeterminada. Esto se realiza en cada ubicación de la capa de características donde se aplica el filtro, generando así un valor de salida [26].
- Cajas predeterminadsa y relaciones de aspecto. Se asocia un conjunto de cajas predeterminadas con cada celda del mapa de características, las cuales cubren el mapa de características de manera convolucional, lo que significa que la posición de cada caja en relación con su celda correspondiente es fija. En cada celda, se predicen los desplazamientos relativos a las formas de las cajas predeterminadas en esa celda, así como las puntuaciones por clase que indican la presencia de una instancia de clase en esas cajas [50].

Mediante estas características, como el propio nombre del modelo indica, este tipo de red es capaz de detectar objetos de diferentes tamaños en una sola pasada de la red. En este trabajo, el modelo utilizado se basa en el artículo [50] y en Pytorch se puede instanciar mediante *torchvision.models.detection.ssd300_vgg16*. En este caso, como se puede observar en la Figura 4.5 que muestra la estructura del modelo, las imágenes de entrada deben ser de $300 * 300$. A esta imagen se le aplican diversas capas de convolución y *pooling* hasta llegar al resultado final.

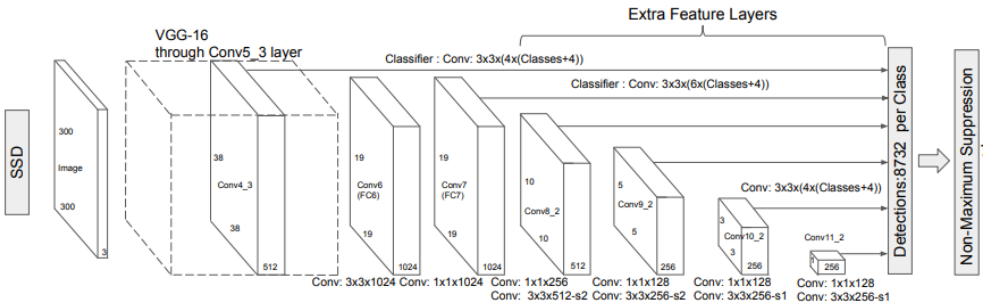


Figura 4.5: Arquitectura de red para un modelo SSD [50]

Al igual que para los modelos de *Faster R-CNN*, se han realizado distintos entrenamientos, variando el valor de los parámetros e hiper-parámetros correspondientes. En el siguiente apartado se detallan los resultados obtenidos para cada uno de estos entrenamientos.

Análisis de resultados

5

A lo largo del presente apartado, se realiza una recopilación de los resultados obtenidos tras los diversos entrenamientos de los modelos. Se muestran tablas que recogen la inicialización de los distintos hiper-parámetros, así como datos de las simulaciones realizadas: duración del entrenamiento, tasa de acierto, errores... Asimismo, se añade algún ejemplo de imágenes con sus *bounding boxes* predichas.

Para realizar los distintos entrenamientos que componen este trabajo, se han utilizado diversos valores de los hiper-parámetros, los cuales se recogen en la Tabla 5.1

Parámetro	**Valores**
Optimizador	AdamW; SGD
Tasa de aprendizaje (lr)	0.01; 0.001; 0.0001
Batch size	16; 32; 64
Nº de épocas	200; 500; 1000

Tabla 5.1: Valores de los hiper-parámetros utilizados en los distintos entrenamientos

5.1 Duración de los entrenamientos

Este primer apartado se centra principalmente en la duración de cada uno de los entrenamientos, realizando para ello distintas combinaciones de los hiper-parámetros presentes en la Tabla 5.1. Es importante destacar que no todas estas simulaciones han sido realizadas utilizando el mismo dispositivo. Como se ha comentado en el apartado 1 de este documento, parte de ellas han sido realizadas utilizando una GPU NVIDIA GeForce RTX 2080 Ti ofrecida por Tecnalia, la cual agilizaba mucho los entrenamientos. Sin embargo, al no haber podido realizar suficientes entrenamientos en el periodo de prácticas y no disponer de una GPU a nivel personal, muchas de ellas han sido entrenadas en una CPU. Para compensar esa diferencia, se ha calculado de manera aproximada la diferencia existente entre ambas máquinas a nivel de tiempo de entrenamiento, pudiendo estimar el tiempo que le llevaría a la GPU realizar el mismo entrenamiento. De esta manera, aunque el resultado quizás no sea tan preciso como se desearía, se pueden dar por válidos los tiempos.

Los resultados obtenidos para los modelos *Faster R-CNN* y SSD se muestran en las Tablas 5.2 y 5.3 respectivamente.

ID	*Backbone*	Optimizador	lr	*Batch size*	N.º Épocas	Duración Entrenamiento (min)
1	ResNet-50-FPN	AdamW	0.01	32	200	3,78
2	ResNet-50-FPN	AdamW	0.001	16	200	4,58
3	ResNet-50-FPN	AdamW	0.001	32	200	4,31
4	ResNet-50-FPN	AdamW	0.01	16	500	7,14
5	ResNet-50-FPN	AdamW	0.01	32	500	6,35
6	ResNet-50-FPN	AdamW	0.001	16	500	9,47
7	ResNet-50-FPN	AdamW	0.001	32	500	8,73
8	ResNet-50-FPN	AdamW	0.001	64	500	7,19
9	ResNet-50-FPN	AdamW	0.0001	32	500	10,25
10	ResNet-50-FPN	AdamW	0.01	32	1000	12,02
11	ResNet-50-FPN	AdamW	0.001	32	1000	15,27
12	ResNet-50-FPN	AdamW	0.001	64	1000	13,86
13	ResNet-50-FPN	AdamW	0.0001	32	1000	19,11
14	ResNet-50-FPN	SGD	0.001	32	500	9,03
15	ResNet-50-FPN	SGD	0.001	64	1000	8,17
16	MobileNetV3-Large FPN	AdamW	0.01	32	200	3,46
17	MobileNetV3-Large FPN	AdamW	0.001	16	200	4,51
18	MobileNetV3-Large FPN	AdamW	0.001	32	200	4,07
19	MobileNetV3-Large FPN	AdamW	0.01	16	500	6,81
20	MobileNetV3-Large FPN	AdamW	0.01	32	500	6,17
21	MobileNetV3-Large FPN	AdamW	0.001	16	500	9,16
22	MobileNetV3-Large FPN	AdamW	0.001	32	500	8,81
23	MobileNetV3-Large FPN	AdamW	0.001	64	500	6,97
24	MobileNetV3-Large FPN	AdamW	0.0001	32	500	10,13
25	MobileNetV3-Large FPN	AdamW	0.01	32	1000	11,54
26	MobileNetV3-Large FPN	AdamW	0.001	32	1000	14,69
27	MobileNetV3-Large FPN	AdamW	0.001	64	1000	13,52
28	MobileNetV3-Large FPN	AdamW	0.0001	32	1000	18,57
29	MobileNetV3-Large FPN	SGD	0.001	32	500	8,91
30	MobileNetV3-Large FPN	SGD	0.001	64	1000	7,32

Tabla 5.2: Identificador de cada modelo, combinación de los hiper-parámetros y duración del entrenamiento para los modelos *Faster R-CNN*

ID	*Backbone*	Optimizador	lr	*Batch size*	N.º Épocas	Duración Entrenamiento (min)
31	VGG16	AdamW	0.01	32	200	3,27
32	VGG16	AdamW	0.001	16	200	4,01
33	VGG16	AdamW	0.001	32	200	3,53
34	VGG16	AdamW	0.01	16	500	6,45
35	VGG16	AdamW	0.01	32	500	5,89
36	VGG16	AdamW	0.001	16	500	8,97
37	VGG16	AdamW	0.001	32	500	7,81
38	VGG16	AdamW	0.001	64	500	6,59
39	VGG16	AdamW	0.0001	32	500	9,43
40	VGG16	AdamW	0.01	32	1000	11,17
41	VGG16	AdamW	0.001	32	1000	13,98
42	VGG16	AdamW	0.001	64	1000	13,13
43	VGG16	AdamW	0.0001	32	1000	17,88
44	VGG16	SGD	0.001	32	500	8,23
45	VGG16	SGD	0.001	64	1000	6,93

Tabla 5.3: Identificador de cada modelo, combinación de los hiper-parámetros y duración del entrenamiento para los modelos SSD

Comparando los resultados obtenidos, se pueden extraer una serie de conclusiones. En primer lugar, aunque la diferencia no sea muy notoria, en general, los entrenamientos realizados mediante los modelos SSD son algo más rápidos. La explicación rápida que se encuentra a este resultado es la diferencia entre la arquitectura de ambos lados, ya que la arquitectura *Faster R-CNN* es más compleja al utilizar dos etapas para detectar los objetos, mientras que la arquitectura SSD es de una sola etapa. Asimismo, en cuanto al *backbone* utilizado en los modelos de *Faster R-CNN*, la arquitectura a MobileNetV3-Large FPN ha ofrecido tiempos de entrenamiento algo más bajos, aunque la diferencia no es notoria.

Por otro lado, la elección del valor para los distintos hiper-parámetros sí que ha resultado crítica. En cuanto al *learning rate* escogido, se puede afirmar con rotundidad, que una tasa de aprendizaje pequeña requiere mayor tiempo de entrenamiento, ya que se necesitan más pasos para alcanzar la convergencia. Del mismo modo, la elección del *batch size* también tiene su efecto en la duración. En este caso, se ha podido concluir que un tamaño de lote grande acelera los entrenamientos, ya que se consigue procesar más información a la vez y no se requiere tanto tiempo para procesar el conjunto de datos. Por último, el número de épocas es directamente proporcional a la duración del entrenamiento; cuantas más épocas se realicen, más tiempo tarda el entrenamiento. Estas dos últimas conclusiones eran aspectos esperados, ya que cuanto más se quiere afinar, mayor será el tiempo de entrenamiento.

5.2 Rendimiento y efectividad de los modelos

Una vez comprobados los tiempos de entrenamiento requeridos por cada uno de los modelos, es necesario analizar la efectividad de cada uno de esos modelos a la hora de realizar la fase de inferencia. Para ello, en este trabajo en concreto, se han utilizado las siguientes métricas:

- Precisión, o en inglés *precision*. Esta métrica, comprendida siempre entre 0 y 1, representa la proporción de detecciones positivas correctas con respecto al total de detecciones positivas realizadas por el modelo, como bien indica la siguiente ecuación. Un valor alto de esta métrica significa que el rendimiento es adecuado.

$$\text{Precision} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Positivos}} \tag{5.1}$$

- Recuperación, o en inglés *recall*. Esta métrica, también comprendida siempre entre 0 y 1, mide la proporción de detecciones positivas correctas con respecto al total de objetos positivos presentes en el *dataset*, como representa esta expresión. Cuanto más cercano sea su valor a 1, mejores resultados ofrecerá el modelo.

$$\text{Recall} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Negativos}} \tag{5.2}$$

- Intersección sobre unión (IoU, del inglés *Intersection over Union)* [39]. Esta métrica se encarga de evaluar la superposición en la *bounding box* predicha por el modelo y la real del objeto. Para ello calcula la división entre el área de intersección y el área de unión de las dos cajas, tal y como se describe en la Figura 5.1.

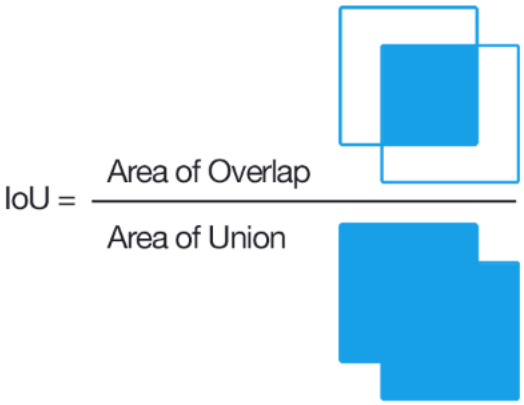


Figura 5.1: Interpretación gráfica y fórmula de la métrica *IoU* [1]

[1] Fuente: `https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`

- *mean Average Precision* (mAP). Esta métrica, muy utilizada en este tipo de tareas, calcula la media de los valores AP (del inglés, *Average Precision*) calculados para cada clase. Estos valores representan la precisión promedio según ciertos umbrales establecidos [35].

Para explicar las métricas de *precision* y *recall*, se ha hablado de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. Estos términos representan las cuatro opciones que pueden aparecer en la fase de *test*, como muestra la Figura 5.2. En este ejemplo de detección de semáforos, existe la posibilidad de efectivamente detectar un semáforo (*True Positive*), de detectar objeto pero que no sea semáforo (*False Positive*), de no detectar nada no habiendo objeto (*True Negative*) o de no detectar nada cuando sí existe semáforo (*False Negative*).

Figura 5.2: Posibles resultados que ofrece la fase de *test* [23]

A su vez, dentro de los falsos positivos, existen distintas variantes, representadas en la Figura 5.3. Una imagen puede ser declarada como falso positivo en tareas de detección de objetos debido a tres motivos. Por un lado, que haya detectado objeto, pero se le haya asignado una clase distinta. Por otro lado, que haya detectado el objeto pero la métrica Iou sea inferior a 1. Por último, que no exista intersección entre la *bounding box* predicha y la real.

[2]Fuente: `https://machinethink.net/blog/object-detection/`

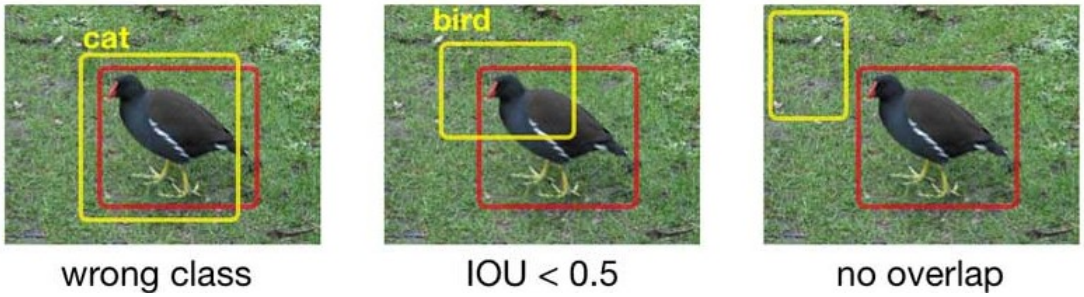


Figura 5.3: Tipos de falsos positivos [2]

Una vez asimilados estos conceptos, se representan en la Tabla 5.4 los resultados obtenidos en cada uno de los entrenamientos. Esta tabla muestra para cada uno de los modelos, representados por el mismo ID de las Tablas 5.2 y 5.3, el valor de cada métrica recién explicada. Dicha tabla, adjunta en la siguiente hoja, da lugar a una serie de interpretaciones, entre las cuales destacan:

- En cuanto al modelo utilizado, comparando los resultados obtenidos mediante *Faster R-CNN* y SSD, se observa que, de modo general, el rendimiento de estos primeros es algo mejor. Aunque en ciertas combinaciones de los hiper-parámetros, esta diferencia no se llega a apreciar, en la mayoría de ellos, ante las mismas combinaciones, el *Faster R-CNN* es ligeramente más eficaz.

- En cuanto a los *backbones* escogidos dentro de los modelos *Faster R-CNN*, la conclusión obtenida es similar. Quizás tenga un mejor rendimiento el ResNet-50-FPN, pero se ha llegado a resultados muy similares. Esto se puede deber a que este modelo tiene una mayor profundidad y capacidad de representación.

- El *learning-rate* se postula como un hiper-parámetro clave en el entrenamiento de estos modelos. Una elección adecuada puede acelerar la convergencia del modelo, pero siempre hay que andar con mucha atención. Equivocarse en su elección puede concluir en una solución poco efectiva o de un rendimiento menor al deseado. Por lo tanto, es importante conseguir una buena correlación entre éste y el resto de parámetros.

- Al igual que en el caso anterior, la elección del *batch size* es crucial para un buen rendimiento. Éste no debe ser ni muy elevado ni muy pequeño, buscando siempre una buena relación con el resto de valores.

- Como era esperado desde un primer momento, de modo general, cuanto mayor ha sido el número de épocas utilizado, mejores resultados en cuanto a eficiencia se han conseguido. Sin embargo, en ocasiones, con un número de épocas elevado, ha llegado a aparecer el problema de *overfitting*. Para solucionarlo se han empleado técnicas como el *dropout* o *data augmentation*, consiguiendo más o menos el efecto deseado, aunque no se han podido llevar a cabo tantas pruebas como se hubiera querido.

ID	*Precision*	*Recall*	IoU	mAP
1	0.35	0.47	0.41	0.38
2	0.47	0.42	0.43	0.45
3	0.41	0.49	0.45	0.43
4	0.39	0.32	0.37	0.41
5	0.58	0.52	0.53	0.55
6	0.71	0.65	0.67	0.63
7	0.65	0.72	0.70	0.71
8	0.52	0.46	0.51	0.53
9	0.73	0.75	0.73	0.74
10	0.44	0.45	0.45	0.43
11	0.73	0.69	0.70	0.68
12	0.64	0.57	0.61	0.63
13	0.75	0.72	0.73	0.70
14	0.66	0.61	0.63	0.65
15	0.63	0.61	0.61	0.61
16	0.32	0.31	0.29	0.32
17	0.46	0.41	0.41	0.42
18	0.4	0.47	0.42	0.47
19	0.39	0.35	0.33	0.37
20	0.52	0.55	0.51	0.53
21	0.67	0.63	0.64	0.65
22	0.65	0.67	0.67	0.65
23	0.55	0.50	0.51	0.57
24	0.70	0.68	0.67	0.61
25	0.45	0.49	0.46	0.47
26	0.69	0.71	0.70	0.72
27	0.57	0.65	0.59	0.53
28	0.71	0.68	0.70	0.75
29	0.65	0.57	0.59	0.61
30	0.56	0.53	0.55	0.53
31	0.27	0.35	0.29	0.35
32	0.39	0.37	0.39	0.45
33	0.42	0.45	0.43	0.38
34	0.37	0.41	0.40	0.45
35	0.47	0.52	0.51	0.55
36	0.63	0.67	0.64	0.61
37	0.65	0.59	0.62	0.61
38	0.51	0.53	0.49	0.47
39	0.67	0.63	0.66	0.65
40	0.43	0.51	0.45	0.39
41	0.69	0.73	0.72	0.65
42	0.52	0.55	0.53	0.55
43	0.63	0.67	0.64	0.61
44	0.60	0.56	0.57	0.58
45	0.54	0.55	0.53	0.50

Tabla 5.4: Identificador de cada modelo y métricas asociadas

5.3 Imágenes con sus *bounding boxes* reales y predichas

En este apartado se analizan las distintas imágenes obtenidas en la fase de inferencia para algunos de los modelos entrenados. Como se pueden observar en las figuras, hay variedad en dichos resultados, ya que los resultados obtenidos son dispersos. Por ejemplo, la Figura 5.4 muestra *bounding boxes* predichas en la fase de *test* muy alejadas de la realidad, ya sea porque la detección está fuera de lugar o porque la clase es equivocada. Sin embargo, en la Figura 5.5 se observa como dichas predicciones han mejorado, consiguiendo un rendimiento satisfactorio.

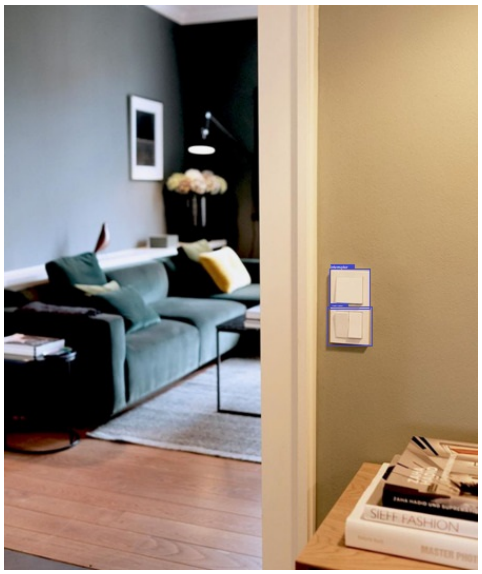
(a) (b)

(c) (d)

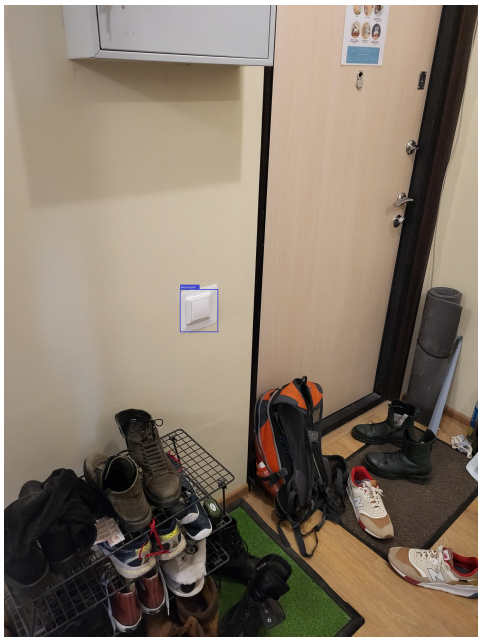
Figura 5.4: Ejemplos de imágenes con malos resultados

(a)

(b)

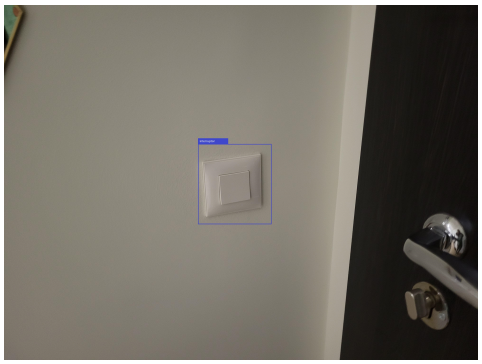


(c)



(d)

(e)



(f)

Figura 5.5: Ejemplos de imágenes con relativamente buenos resultados

Conclusiones y trabajos futuros

6

En este trabajo, se ha llevado a cabo una extensa evaluación de diversos modelos de redes neuronales convolucionales en el contexto de la detección de objetos. Si bien es cierto que quizás no se han podido trabajar con tantos modelos de DL como se pretendía en un instante inicial, la experimentación con distintas arquitecturas ha proporcionado una comprensión profunda de sus respectivas fortalezas y debilidades. Asimismo, se ha podido comprobar la importancia de parámetros como las funciones de activación, los optimizadores y las tasas de aprendizaje en la mejora del rendimiento de los modelos. Esto no hace más que resaltar la importancia de realizar una selección cuidadosa de parámetros e hiper-parámetros en todo proyecto de DL que se trabaje.

Otro aspecto que merece la pena destacar es que no existe una única solución para todos los problemas de visión por computadora. Cada modelo de DL tiene una serie de características que le permitirá adaptarse mejor a según que tareas, por lo que, que una arquitectura funcione bien en una tarea concreta, no quiere decir que vaya a ofrecer soluciones satisfactorias en cualquier otra. En función de la variable a la que se dé mayor importancia en cada aplicación (coste computacional, tiempo de cálculo, rapidez, precisión...) será más recomendable uno u otro modelo. Por ejemplo, en tareas en la que el coste computacional no sea un problema y se quiera una gran precisión, se pueden utilizar *R-CNN*, mientras que en aquellas tareas donde se necesite mayor rapidez, una arquitectura *Faster R-CNN* puede ser mejor opción. Asimismo, el utilizar modelos ya preentrenados reduce el tiempo de entrenamiento significativamente, ofreciendo también resultados por lo general más eficientes. Esta variedad puede tener sus desventajas, pero también su parte positiva, ya que, con pequeñas modificaciones en las arquitecturas de los modelos, se puede llegar a ser efectivo en aplicaciones muy variadas.

Como conclusión más específica del proyecto, se puede afirmar que se han logrado resultados relativamente precisos al utilizar ciertos modelos, superando las prestaciones que ofrecen los marcadores o el seguimiento sin marcadores. Entre estos modelos se deben destacar aquellos que mejores resultados han ofrecido y se podrían implementar en una aplicación real. Es importante mencionar que, a la hora de implementar, se deberá tener en cuenta qué tipo de características se desea que sean relevantes: velocidad, rendimiento, eficiencia... El objetivo será encontrar un buen equilibrio entre todas ellas.

En resumen, este trabajo ha servido para conocer de una manera más profunda el funcionamiento de los diversos modelos de CNNs y su aplicación en tareas como la detección de objetos. El poder aplicar estas técnicas de DL en aplicaciones de RA concretas debe superar significativamente a otras técnicas tradicionales utilizadas hasta hace no mucho, como el uso de marcadores o el seguimiento sin marcadores.

6.1 Líneas de trabajo futuras

Respecto a futuras líneas de investigación y desarrollo relacionadas con este proyecto, se abren varios enfoques realmente interesantes. Si bien se han logrado avances en la detección de objetos utilizando diferentes arquitecturas de CNNs, se puede explorar aún más la aplicación de otros tipos de modelos que no han podido ser entrenados debido a falta de tiempo, como los sistemas YOLO, o simplemente realizar más pruebas modificando el valor inicial de los parámetros e hiper-parámetros. La implementación de sistemas YOLO podría permitir una detección más rápida y precisa de objetos en entornos dinámicos, lo que podría ser esencial para aplicaciones de RA y robótica.

Además, se podría considerar la expansión de esta investigación hacia la estimación de la pose de objetos 6D [19]. De esta manera se lograría no solo detectar los objetos en una imagen, sino también comprender su posición y orientación en el espacio tridimensional, lo que enriquería en gran medida la experiencia en aplicaciones de RA. Asimismo, una estimación precisa de la pose es fundamental para otro tipo de aplicaciones como la manipulación robótica y la navegación autónoma.

Por último, como enfoque más práctico y destinado a poner en marcha el algoritmo seleccionado, se podría implementar dicho algoritmo en dispositivos de RA. Esto permitiría la integración dEL sistema en aplicaciones de RA para mejorar la experiencia del usuario al interactuar con el entorno físico a través de los dispositivos correspondientes. Esta implementación podría requerir la optimización de los modelos y la consideración de las restricciones de recursos de los dispositivos de RA.

Referencias bibliográficas

[1]Iñigo Fernández del Amo, John Ahmet Erkoyuncu, Rajkumar Roy, Riccardo Palmarini y Demetrius Onoufriou. „A systematic review of Augmented Reality content-related techniques for knowledge transfer in maintenance applications“. En: *Computers in Industry* 103 (dic. de 2018), págs. 47-71 (vid. págs. 15, 16, 19).

[2]Nantheera Anantrasirichai y David Bull. „Artificial intelligence in the creative industries: a review“. En: *Artificial Intelligence Review* 55 (1 ene. de 2022), págs. 589-656 (vid. págs. 5, 15).

[3]Itamar Arel, Derek Rose y Thomas Karnowski. „Deep machine learning-A new frontier in artificial intelligence research“. En: *IEEE Computational Intelligence Magazine* 5 (4 nov. de 2010), págs. 13-18 (vid. pág. 6).

[4]Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser et al. „Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI“. En: (oct. de 2019) (vid. pág. 1).

[5]Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser et al. „Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI“. En: *Information Fusion* 58 (jun. de 2020), págs. 82-115.

[6]Ronald T Azuma. *A Survey of Augmented Reality*. 1997, págs. 355-385.

[7]Oliver Bimber. „Whats Real About Augmented Reality?“ En: ().

[8]Cristiano Castelfranchi. „Alan Turing's "Computing Machinery and Intelligence"“. En: *Topoi* 32 (2 oct. de 2013), págs. 293-299.

[9]IEEE Romania Section. CAS/CS Joint Chapter, IEEE Power Electronics Society. Romania Chapter, Institute of Electrical y Electronics Engineers. *2019 6th International Symposium on Electrical and Electronics Engineering (ISEEE)*.

[10]B Jack Copeland. *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life: Plus The Secrets of Enigma*.

[11]Ajaya Kumar Dash, Santosh Kumar Behera, Debi Prosad Dogra y Partha Pratim Roy. „Designing of marker-based augmented reality learning environment for kids using convolutional neural network architecture“. En: *Displays* 55 (dic. de 2018), págs. 46-54 (vid. pág. 19).

[12]Shabnam Sadeghi Esfahlani, Alireza Sanaei, Mohammad Ghorabian y Hassan Shirvani. „The Deep Convolutional Neural Network Role in the Autonomous Navigation of Mobile Robots (SROBO)“. En: *Remote Sensing* 14 (14 jul. de 2022) (vid. pág. 21).

[13]Nihon Robotto Gakkai, IEEE Robotics, Automation Society et al. *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008 IROS 2008 ; 22-26 Sept., 2008, Acropolis Convention Center, Nice, France*.

[14]Yalda Ghasemi, Heejin Jeong, Sung Ho Choi, Kyeong Beom Park y Jae Yeol Lee. *Deep learning-based object detection in augmented reality: A systematic review*. Ago. de 2022 (vid. págs. 20, 29).

[15]Yalda Ghasemi, Heejin Jeong, Sung Ho Choi, Kyeong Beom Park y Jae Yeol Lee. *Deep learning-based object detection in augmented reality: A systematic review*. Ago. de 2022.

[16]Ronny Alejandro Dieguez Guach, Jairo Jesús Gomez Tejeda y Manuel Ramón Pérez Abreu. „Características clínico-epidemiológicas de la COVID-19“. En: *Revista Habanera de Ciencias Médicas* 19.2 (2020), págs. 1-15.

[17]Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun. „Deep residual learning for image recognition“. En: vol. 2016-December. IEEE Computer Society, dic. de 2016, págs. 770-778 (vid. pág. 6).

[18]Aaron Hertzmann. *Machine Learning for Computer Graphics: A Manifesto and Tutorial*. 2003.

[19]Tomáš Hodaň, Jiří Matas y Stěpán Obdržálek. *On Evaluation of 6D Object Pose Estimation* (vid. pág. 46).

[20]Sabera Hoque, Md Yasir Arafat, Shuxiang Xu, Ananda Maiti y Yuchen Wei. „A Comprehensive Review on 3D Object Detection and 6D Pose Estimation with Deep Learning“. En: *IEEE Access* 9 (2021), págs. 143746-143770 (vid. pág. 21).

[21]Mitsuru Igami. „Artificial intelligence as structural estimation: Deep Blue, Bonanza, and AlphaGo“. En: *Econometrics Journal* 23 (3 2020), S1-S24 (vid. pág. 6).

[22]Emil L. Jacobsen y Jochen Teizer. „Deep Learning in Construction: Review of Applications and Potential Avenues“. En: *Journal of Computing in Civil Engineering* 36 (2 mar. de 2022) (vid. pág. 21).

[23]Tiagrajah v. Janahiraman y Mohamed Shahrul Mohamed. „Traffic Light Detection Using Tensorflow Object Detection Framework“. En: *IEEE* (2019) (vid. pág. 39).

[24]N.Rochester J.McCarthy M.L.Minsky y C.E.Shannon. „A proposal for the Dartmouth summer research project on Artificial Intelligence“. En: (1995) (vid. pág. 5).

[25]Max Kuhn y Kjell Johnson. *Applied Predictive Modeling*.

[26]Jeong Seon Lim, Marcella Astrid, Hyun Jin Yoon y Seung Ik Lee. „Small Object Detection using Context and Attention“. En: Institute of Electrical y Electronics Engineers Inc., abr. de 2021, págs. 181-186 (vid. pág. 33).

[27]Tsung-Yi Lin, Michael Maire, Serge Belongie et al. „Microsoft COCO: Common Objects in Context“. En: (mayo de 2014) (vid. pág. 24).

[28]Chang Liu, Samad M.E. Sepasgozar, Sara Shirowzhan y Gelareh Mohammadi. „Applications of object detection in modular construction based on a comparative evaluation of deep learning algorithms“. En: *Construction Innovation* 22 (1 ene. de 2022), págs. 141-159 (vid. pág. 32).

[29]Yanli Liu, Xingming Zou, Songhua Xu et al. „Real-Time Shadow Detection from Live Outdoor Videos for Augmented Reality“. En: *IEEE Transactions on Visualization and Computer Graphics* 28 (7 jul. de 2022), págs. 2748-2763 (vid. pág. 21).

[30]Luis Muñoz-Saavedra, Lourdes Miró-Amarante y Manuel Domínguez-Morales. „Augmented and virtual reality evolution and future tendency“. En: *Applied Sciences (Switzerland)* 10 (1 ene. de 2020).

[31]Hanhoon Park y Jong Ii Park. „Invisible marker-based augmented reality“. En: *International Journal of Human-Computer Interaction* 26 (9 2010), págs. 829-848 (vid. pág. 19).

[32]International Association for Pattern Recognition, Zhongguo ke xue yuan y Chinese Association of Automation. *2018 24th International Conference on Pattern Recognition (ICPR)*.

[33]Dawid Połap, Karolina Kęsik, Kamil Książek y Marcin Woźniak. „Obstacle detection as a safety alert in augmented reality models by the use of deep learning techniques“. En: *Sensors (Switzerland)* 17 (12 dic. de 2017) (vid. págs. 16, 21).

[34]*PyTorch Lightning Documentation Release 1.1.5 William Falcon et al*. 2021.

[35]Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhadi. „You Only Look Once: Unified, Real-Time Object Detection“. En: (jun. de 2015) (vid. pág. 39).

[36]Joseph Redmon y Ali Farhadi. „YOLO9000: Better, Faster, Stronger“. En: (dic. de 2016).

[37]Joseph Redmon y Ali Farhadi. „YOLOv3: An Incremental Improvement“. En: (abr. de 2018).

[38]Shaoqing Ren, Kaiming He, Ross Girshick y Jian Sun. „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks“. En: (jun. de 2015) (vid. págs. 30, 31).

[39]Hamid Rezatofighi, Nathan Tsoi, Junyoung Gwak et al. *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression* (vid. pág. 38).

[40]Louis Rosenberg. *HOW A PARACHUTE ACCIDENT HELPED JUMP-START AUGMENTED REALITY IN 1992, HARDWARE FOR THE FIRST INTERACTIVE AR SYSTEM LITERALLY FELL FROM THE SKIES BY LOUIS ROSENBERG*.

[41]Dieter Schmalstieg y Tobias Höllerer. *Augmented Reality: Principles and Practice* (vid. págs. 16, 17).

[42]K. Shalini, Abhishek Kumar Srivastava, Surendra Allam y Dilip Lilaramani. „Comparative analysis on Deep Convolution Neural Network models using Pytorch and OpenCV DNN frameworks for identifying optimum fruit detection solution on RISC-V architecture“. En: Institute of Electrical y Electronics Engineers Inc., 2021, págs. 738-743.

[43]Eli Stevens, Luca Antiga y Thomas Viehmann. *Deep Learning with PyTorch* (vid. págs. 6, 7, 9).

[44]Kang Tong, Yiquan Wu y Fei Zhou. „Recent advances in small object detection based on deep learning: A review“. En: *Image and Vision Computing* 97 (mayo de 2020) (vid. págs. 6, 19).

[45]A M Turing. *M I N D A QUARTERLY REVIEW OF PSYCHOLOGY AND PHILOSOPHY I.-COMPUTING MACHINERY AND INTELLIGENCE*. 1950 (vid. pág. 5).

[46] Dong Wang, Xiaoling Wang, Bingyu Ren et al. „Vision-Based Productivity Analysis of Cable Crane Transportation Using Augmented Reality–Based Synthetic Image“. En: *Journal of Computing in Civil Engineering* 36 (1 ene. de 2022) (vid. pág. 21).

[47] Pei Wang. „On Defining Artificial Intelligence“. En: *Journal of Artificial General Intelligence* 10 (2 ene. de 2019), págs. 1-37.

[48] Shaohan Wang, Sakib Ashraf Zargar y Fuh Gwo Yuan. „Augmented reality for enhanced visual inspection through knowledge-based deep learning“. En: *Structural Health Monitoring* 20 (1 ene. de 2021), págs. 426-442 (vid. pág. 21).

[49] Sun-Chong Wang. *Interdisciplinary Computing in Java Programming*. Springer US, 2003 (vid. pág. 8).

[50] Dumitru Erhan Christian Szegedy Scott Reed Cheng-Yang Fu Alexander C. Berg Wei Liu Dragomir Anguelov. „SSD: Single Shot MultiBox Detector“. En: () (vid. pág. 33).

[51] Bo Xiao, Hairong Xiao, Jingwen Wang y Yuan Chen. „Vision-based method for tracking workers by integrating deep learning instance segmentation in off-site construction“. En: *Automation in Construction* 136 (abr. de 2022).

[52] Li Yang y Abdallah Shami. „On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice“. En: (jul. de 2020) (vid. pág. 10).

[53] Yoshua Bengio Yann LeCun Léon Bottou y Patrick Haffner. „Gradient-Based Learning Applied to Document Recognition“. En: (nov. de 1998) (vid. pág. 7).

Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN INGENIERÍA DE CONTROL, AUTOMATIZACIÓN Y ROBÓTICA

TRABAJO FIN DE MÁSTER

DETECCIÓN DE OBJETOS BASADA EN MACHINE LEARNING PARA REALIDAD AUMENTADA

ANEXO A: ESQUEMAS Y PROGRAMAS FUENTE

Estudiante: Calleja Corcuera, Jon Ander

Director/Directora: Zulueta Guerrero, Ekaitz

Curso: 2022-2023

Fecha: Bibao, a 20 de septiembre de 2023

Esquemas y programas fuente

A

A.1 Archivo del modelo

```
    import timm as timm
import pytorch_lightning as pl
import torch
import torchvision
from torchvision import ops
from torchvision.models.detection.faster_rcnn import
   FastRCNNPredictor

from torch import nn
from timm.optim import create_optimizer_v2
from torch.optim import lr_scheduler
from torchmetrics import AveragePrecision
from torchmetrics.detection import mean_ap


class FasterRCNN(pl.LightningModule):

    def __init__(self,num_classes,params,training):
        super().__init__()

        # Load an object detection model pre-trained on
           COCO
        self.model = torchvision.models.detection.
           fasterrcnn_resnet50_fpn(pretrained=True)

        # Get number of input features for the classifier
        in_features = self.model.roi_heads.box_predictor.
           cls_score.in_features
        # Replace the pre-trained head with a new one
        self.model.roi_heads.box_predictor =
           FastRCNNPredictor(in_features, num_classes)

        self.save_hyperparameters()
        self.opt=params['train']['optimizer']
```

```
30          self.lr=params['train']['learning_rate']
31          self.batch=params['train']['batch']
32
33          # List of categories with their corresponding
                indexes
34          self.category_list = params['categories']
35          self.category_list.sort()
36          self.pred=[]
37
38          self.training=training
39
40
41      def forward(self, x, targets=None):
42
43          if self.training:
44              # En el caso de entrenamiento y validaci n,
                    se le pasa al modelo las im genes y las
                    targets
45              output = self.model(x, targets)
46              return output
47          else:
48              # En el caso de evaluaci n o prueba, solo se
                     pasan las im genes al modelo
49              return self.model(x)
50
51
52      def configure_optimizers(self):
53          optimizer=create_optimizer_v2(self.model,opt=self
                .opt,lr=self.lr)
54          # optimizer = params['train']['optimizer']
55          # Learning rate reduction
56          scheduler=lr_scheduler.ReduceLROnPlateau(
                optimizer=optimizer,mode='min',factor=0.5,
                patience=100)
57          return [optimizer],[{"scheduler": scheduler, "
                monitor": "train_loss"}]
58
59      '''def classification_loss(self, output):
60          # Se obtienen la perdida de clasificacion entre
                las etiquetas reales y las predichas
61          loss_classification = output.losses['
                loss_classifier']
```

```
62
63          labels = targets['labels']
64
65          # Calcular la p rdida de clasificaci n
                utilizando la funci n de p rdida de
                entrop a cruzada
66          loss_classification = nn.functional.cross_entropy
                (logits, labels)
67          loss_classification=nn.CrossEntropyLoss(logits,
                labels)
68
69          return loss_classification
70
71
72      def bbox__iou_loss(self, output): # , targets (no se
            si hara falta a adir)
73          # Obtener perdida de regresion entre las
                predicciones de las bounding boxes y las
                bounding boxes reales
74          box_loss = output['loss_box_reg']
75
76          pred_boxes = output['boxes']
77          target_boxes = targets['bounding_boxes']
78
79          # Calcular la intersecci n sobre la uni n (IoU)
                entre las bounding boxes
80          iou = ops.box_iou(pred_boxes, target_boxes)
81
82          # Calcular la p rdida utilizando 1 - IoU como
                medida de p rdida
83          loss_iou = 1 - iou
84
85          return box_loss
86
87
88      def total_loss(self, output, classification_weight
            =1.0, bbox_iou_weight=1.0): # , targets
89          # Calcular las p rdidas de clasificaci n y
                bounding box IoU
90          classification_loss = self.classification_loss(
                output) # , targets (no se si hara falta
                a adir)
```

```
91          bbox_iou_loss = self.bbox_iou_loss(output) # ,
                targets (no se si hara falta a adir)
92
93          # Calcular la p rdida total como la media
                ponderada de las dos p rdidas
94          total_loss = (classification_weight *
                classification_loss) + (bbox_iou_weight *
                bbox_iou_loss)
95          # Calcular la p rdida total como la suma de las
                dos p rdidas entre dos
96          # total_loss = (classification_loss +
                bbox_iou_loss)/2
97
98          return total_loss
99
100     def training_step(self,batch,batch_idx):
101
102         images, targets = batch
103
104         # Calcular la salida del modelo y la p rdida
                total
105         output = self(images)
106         total_loss = self.total_loss(output) #, targets
107
108         # Log the loss
109         self.log('train_loss', total_loss, on_step=True,
                on_epoch=True, prog_bar=True)
110
111         # Calcular la precisi n utilizando la Mean
                Average Precision (MAP)
112         logits = output['loss_classifier']
113         labels = targets['category_id']
114         preds = logits.argmax(dim=1)
115         self.mean_ap(output, targets)
116         self.log('train_precision', self.mean_ap,
                on_epoch=True, prog_bar=True)
117
118         # Devuelve tanto la p rdida como la precisi n
119         return {'loss': total_loss, 'mean_avg_precission
                ': self.ap}
120
121
```

```
122     def training_epoch_end(self, outputs):
123         avg_train_loss=torch.tensor([x['loss'] for x in
                outputs]).mean()
124         avg_train_map=torch.tensor([x['
                mean_avg_precission'] for x in outputs]).mean
                ()
125
126         self.log("train_loss_mean",avg_train_loss)
127         self.log("train_map_mean",avg_train_map)
128
129     def validation_step(self, batch, batch_idx):
130         valid_results=self.training_step(batch=batch,
                batch_idx=batch_idx)
131         return valid_results
132
133     def validation_epoch_end(self, outputs):
134         avg_val_loss=torch.tensor([x['loss'] for x in
                outputs]).mean()
135         avg_val_map=torch.tensor([x['mean_avg_precission
                '] for x in outputs]).mean()
136
137         self.log("val_loss_mean",avg_val_loss)
138         self.log("val_acc_mean",avg_val_map)
139
140
141     def predict_step(self, batch, batch_idx,
            dataloader_idx = None): # MODIFICAR
142         # x,y=batch
143
144         # logits=self.forward(x)
145         #pred_idx = torch.argmax(logits, -1)
146         #self.pred.extend(pred_idx.cpu().numpy())
147
148         images, _ = batch
149
150         # Pasar las im genes por el modelo para obtener
                las predicciones
151         output = self(images)
152
153         # Obtener las predicciones de las bounding boxes
                y las etiquetas
154         box_predictions = output['boxes']
```

```
155         label_predictions = output['labels']
156 
157         return box_predictions, label_predictions'''
158 
159     def training_step(self, batch, batch_idx):
160 
161         self.training = True
162         # FasterRCNN takes both images and targets for
                training and returns the loss
163         images = batch[0]
164         targets = batch[1]
165         loss_dict = self.forward(images, targets)
166         train_loss = sum(loss for loss in loss_dict.
                values())
167 
168         self.log("train_loss", train_loss, on_epoch=True,
                 on_step=False, batch_size=self.batch)
169 
170         return train_loss
171 
172 
173     def validation_step(self, batch, batch_idx):
174 
175         self.training = True
176         # This step is the same as the training step
177         self.model.train() # Necessary train model for
                the model to return the loss (val_loss in this
                 case)
178         #print("Batch length: ", len(batch))
179         images = batch[0]
180         targets = batch[1]
181         #print("Targets length: ", len(targets))
182         #print(targets)
183 
184         loss_dict = self.forward(images, targets)
185         val_loss = sum(loss for loss in loss_dict.values
                ())
186 
187         self.log("val_loss", val_loss, on_epoch=True,
                prog_bar=True, batch_size=self.batch)
188 
189 
```

```
190     def predict_step(self, batch, batch_idx,
            dataloader_idx = None): # MODIFICAR
191
192         images, _ = batch
193
194          # Verificar si images es una lista y convertirla
                en tensor si es necesario
195         if isinstance(images, list):
196             images = torch.stack(images, dim=0)
197
198         # Pasar las im genes por el modelo para obtener
               las predicciones
199         output = self(images)
200
201         # Obtener las predicciones de las bounding boxes
               y las etiquetas
202         box_predictions = output[0] # El orden de la
               lista que devuelve es: boxes, labels, scores
203         label_predictions = output[1]
204
205         return box_predictions, label_predictions
```

A.2 Archivo para generar *Dataset*

```
1     import os
2 import torch
3 import pytorch_lightning as pl
4
5 from torch.utils.data import Dataset,DataLoader
6 from utils.io import read_json
7 import json
8 import numpy as np
9 from PIL import Image
10 #from pycocotools.coco import COCO IGUALsoloEnElModel
11 from torchvision.transforms import functional as F
12
13
14 class dataset_from_list(Dataset):
15     def __init__(self,image_list,dataset_info,
           category_list,json_dir,input_size,transform=None):
16         self.image_list=image_list
```

```
17          # En el de instance segmentation 2+:
18          # self.images_dict = images_dict
19          # self.list_images = list(self.images_dict.keys()
                )
20          # List of categories with their corresponding
                indexes
21          self.category_list = category_list
22          self.n_categories = len(category_list)
23
24          self.dataset_info = dataset_info
25          self.json_dir = json_dir
26          # self.data_dir=data_dir #Puede que haya que
                a adir arriba entre json_dir e input_size
27          self.input_size = input_size
28          self.transform = transform
29
30
31      def __getitem__(self, idx):
32
33          # image_path=self.image_list[idx] # Sacar nombre
                de la imagen
34          # input_path = os.path.join(self.data_dir,
                image_path)
35          # category_id =
36          # image_data = self.image_data[idx]
37
38          image_file = self.image_list[idx]
39          image_name = os.path.splitext(image_file)[0]  #
                Obtiene el nombre del archivo sin la
                extensi n
40
41          json_file = image_name + ".json"
42          json_path = os.path.join(self.json_dir, json_file
                )
43
44          with open(json_path, 'r') as f:
45              image_data = json.load(f)
46
47          # Obtener la informaci n de la imagen y las
                anotaciones del diccionario
48          image_path = image_data['image_path']
49          category_id = image_data['category_id']
```

```
50          bounding_boxes = image_data['bboxes']
51
52          # Cargar la imagen utilizando PIL u otra
                librer a de tu elecci n
53          image = Image.open(image_path).convert('RGB')
54
55          image = np.array(image) # Albumentations necesita
                del tipo de datos numpy
56
57          # En caso de que existan transformaciones, se
                transforma la imagen
58          if self.transform != None:
59              transformed = self.transform(image=image,
                    bboxes=bounding_boxes, category_id=
                    image_data['category_id']) # Creo que
                    sobra: class_labels=target["labels"]
60              image = transformed["image"]
61              bounding_boxes = transformed["bboxes"]
62              #target["labels"] = transformed["class_labels
                    "]
63
64          image = F.convert_image_dtype(image)
65          # target["labels"] = torch.as_tensor(target["
                labels"], dtype=torch.int64)
66          bounding_boxes = torch.as_tensor(bounding_boxes,
                dtype=torch.float32)
67          # target["boxes"] = torch.as_tensor(target["boxes
                "], dtype=torch.float32)
68          # target["masks"] = torch.permute(target["masks
                "],(2,0,1))
69
70          # Convertir las listas en tensores de PyTorch
71          category_ids = torch.tensor(category_id, dtype=
                torch.long)
72          # bboxes = torch.tensor(bounding_boxes, dtype=
                torch.float32)
73
74          if bounding_boxes.numel() > 0:
75              # Obtener las coordenadas individuales de las
                    bounding boxes
76              x1 = bounding_boxes[:, 0]
77              y1 = bounding_boxes[:, 1]
```

```
78              w = bounding_boxes[:, 2]
79              h = bounding_boxes[:, 3]
80
81              # Calcular las coordenadas de la esquina
                    inferior derecha (x2, y2)
82              x2 = x1 + w
83              y2 = y1 + h
84
85              # Crear el nuevo formato de bounding boxes [
                    x1, y1, x2, y2]
86              new_bounding_boxes = torch.stack([x1, y1, x2,
                     y2], dim=1)
87
88          else:
89              new_bounding_boxes = torch.empty((0, 4),
                    dtype=torch.long)
90              # category_ids = torch.empty((0, 0), dtype=
                    torch.long)
91
92
93
94          # Construir el target que contiene las
                category_ids y bounding boxes
95          target = {
96              'boxes': new_bounding_boxes,
97              'labels': category_ids
98          }
99
100         return image, target
101
102     def __len__(self):
103         # return len(self.image_list)
104         return len(self.image_list)
105
106 def collate_fn(batch):
107         '''
108         Since each image may have a different number of
                objects, we need a collate function (to be
                passed to the DataLoader).
109
110         :param batch: an iterable of N sets from
                __getitem__()
```

```
111         :return: a tensor of images, lists of varying-
                size tensors of bounding boxes, labels, and
                difficulties
112         '''
113
114         images = list()
115         targets=list()
116
117         for i, t in batch:
118             #if len(batch) != 3:
119                 #print('Invalid batch:', batch)
120             images.append(i)
121             targets.append(t)
122         images = torch.stack(images, dim=0)
123
124         return images,targets
125
126 class PL_DataModule(pl.LightningDataModule):
127     def __init__(self,data_dir,split_dir,json_image,
            params,dataset_info,input_size,train_transform,
            test_transform):
128         super().__init__()
129         self.batch_size=params['train']['batch']
130         self.data_dir=data_dir
131         self.split_dir=split_dir
132         self.json_image=json_image
133         self.categories=params['categories']
134         self.dataset_json=dataset_info
135         self.input_size=input_size
136         self.train_transform= train_transform
137         self.test_transform = test_transform
138
139     def setup(self, stage= None):
140         # From json file to torch custom dataset
141         train_split = read_json(os.path.join(self.
                split_dir, 'train_images.json'))['Images']
142         validation_split = read_json(os.path.join(self.
                split_dir, 'validation_images.json'))['Images'
                ]
143         test_split = read_json(os.path.join(self.
                split_dir, 'test_images.json'))['Images']
144         category_list = self.categories
```

```
145         category_list.sort()
146         dataset_info = read_json(self.dataset_json)
147 
148         # Split dataset into training, validation and
               test sets         self.train_set=
               dataset_from_list(train_split,dataset_info,
               category_list,self.json_image,self.input_size,
               transform=self.train_transform) #Aqui antes se
                a adia self.train_transform
149         self.val_set=dataset_from_list(validation_split,
               dataset_info,category_list,self.json_image,
               self.input_size,transform=self.test_transform)
150         self.test_set=dataset_from_list(test_split,
               dataset_info,category_list,self.json_image,
               self.input_size,transform=self.test_transform)
151 
152         '''# Verificar si self.train_set contiene datos
153         if len(self.train_set) > 0:
154             print("self.train_set contiene datos")
155             print(len(self.train_set))
156         else:
157             print("self.train_set est  vac o")
158         # Verificar si self.val_set contiene datos
159         if len(self.val_set) > 0:
160             print("self.val_set contiene datos")
161             print(len(self.val_set))
162         else:
163             print("self.val_set est  vac o")
164         # Verificar si self.test_set contiene datos
165         if len(self.test_set) > 0:
166             print("self.test_set contiene datos")
167             print(len(self.test_set))
168         else:
169             print("self.test_set est  vac o")'''
170 
171     def train_dataloader(self):
172         train_loader=DataLoader(self.train_set,
               batch_size=self.batch_size,shuffle=True,
               num_workers=12, collate_fn=collate_fn,
               pin_memory=True)
173         return train_loader
174 
```

```
175     def val_dataloader(self):
176         val_loader=DataLoader(self.val_set, batch_size=
                self.batch_size,shuffle=False,num_workers=12,
                collate_fn=collate_fn, pin_memory=True)
177         return val_loader
178 
179     def test_dataloader(self):
180         test_loader=DataLoader(self.test_set, batch_size=
                self.batch_size,shuffle=False,num_workers=12,
                collate_fn=collate_fn, pin_memory=True)
181         return test_loader
```

A.3 Archivo de entrenamiento

```
1  import argparse
2  import os
3  import pytorch_lightning as pl
4  import torch
5  import pandas as pd
6  import numpy as np
7  import albumentations as A
8  import cv2
9  
10 from datamodule import PL_DataModule
11 from model import FasterRCNN
12 
13 from utils.io import load_params
14 from utils.devices import config_device
15 
16 from torch import nn
17 
18 from torchvision import transforms
19 from torchvision.transforms.functional import
       InterpolationMode
20 from albumentations.pytorch.transforms import ToTensorV2
21 
22 from pytorch_lightning.callbacks import ModelCheckpoint,
       EarlyStopping,BaseFinetuning
23 from pytorch_lightning.loggers import CSVLogger
24 from pytorch_lightning import seed_everything
25 
```

```
26 class ModelFinetuning (BaseFinetuning): #Callback to
      unfreeze the model in the specified epoch
27     def __init__(self, unfreeze_at_epoch=10):
28         super().__init__()
29         self._unfreeze_at_epoch = unfreeze_at_epoch
30
31     def freeze_before_training(self, pl_module):
32         # the layers are frozen when generating the
              model so nothing is done here
33         pass
34
35     def finetune_function(self, pl_module, current_epoch,
          optimizer, optimizer_idx):
36         # When 'current_epoch' is equal to '
              unfreeze_at_epoch', the network is unfrozen.
37         if current_epoch == self._unfreeze_at_epoch:
38             self.unfreeze_and_add_param_group(
39                 modules=pl_module.model,
40                 optimizer=optimizer,
41                 train_bn=True,
42              )
43
44 def transformations (params):
45     input_size=params['model']['input_size']
46     mean=params['data']['mean']
47     std=params['data']['std']
48
49     #train set transformations
50     train_transform = A.Compose([
51         A.Resize(height=input_size, width=input_size), #
              interpolation=cv2.INTER_LINEAR
52         # A.VerticalFlip(p=0.05),
53         # A.ShiftScaleRotate(p=0.1),
54         # A.RandomBrightnessContrast(p=0.3),
55         A.Normalize(mean,std,max_pixel_value=255.0),
56         ToTensorV2(),
57         # A.Lambda(image=lambda x: np.array(x)),
58         ],
59         bbox_params=A.BboxParams(format='coco',
              label_fields=['category_id'])
60     )
61
```

```
62      #Preprocessing steps applied to validation and test
            set
63      test_transform = A.Compose([
64          A.Resize(height=input_size, width=input_size,
                interpolation=cv2.INTER_LINEAR),
65          A.Normalize(mean,std,max_pixel_value=255.0),
66          ToTensorV2(),
67          # A.Lambda(image=lambda x: np.array(x)),
68          # A.HorizontalFlip(p=0.1),
69          # A.VerticalFlip(p=0.2),
70          # A.ShiftScaleRotate(p=0.5),
71          # A.RandomBrightnessContrast(p=0.3),
72          ],
73          bbox_params=A.BboxParams(format='coco',
                label_fields=['category_id'])
74      )
75      return train_transform, test_transform
76
77  def get_callbacks_list(params):
78      callbacks_list=[]
79
80      # Callback used to save the best weights
81      # Save intermediate weights (only when the weights
            that offer the best validation loss are wanted)
82      if (params['train']['restore_best_weights']):
83          checkpointer =ModelCheckpoint(
84              monitor="val_loss",
85              mode='min',
86              dirpath=args.output,
87              filename='weights',
88              verbose=False,
89              )
90          callbacks_list.append(checkpointer)
91
92      # Early stop (stop the training if the validation
            loss does not improve)
93      if params['train']['early_stopping']:
94          early_stop = EarlyStopping(
95              monitor='val_loss',
96              min_delta=params["train"]["
                    early_stopping_params"]["min_delta"],
```

```
97              patience=params["train"]["
                    early_stopping_params"]["patience"],
98              mode='min',
99              )
100         callbacks_list.append(early_stop)
101
102     if params['train']['epochs'] > 0:
103         fine_tune=ModelFinetuning(
104             unfreeze_at_epoch=params['train']['
                    finetune_epochs']
105         )
106         callbacks_list.append(fine_tune)
107
108     return callbacks_list
109
110 def get_trainer(params, device,accelerator,logger,
    callbacks_list):
111     trainer=pl.Trainer(max_epochs=params['train']['
        finetune_epochs']+params['train']['epochs'],
112                        accelerator=accelerator, #
                           specify the accelerator to
                           use
113                        devices=device, # Select the
                           devices to use
114                        logger=logger,
115                        strategy='ddp',
116                        callbacks=callbacks_list,
117                        deterministic=True, # To ensure
                           reproducibility
118                        log_every_n_steps=1,
119                        )
120     return trainer
121
122 if __name__ == '__main__':
123     parser = argparse.ArgumentParser(description='train')
124     parser.add_argument('--images', help='path␣of␣the␣
        images', default='/home/jonander/ObjectDetection/
        object_detection_project/dataset/images')
125     parser.add_argument('--dataset-json', help='path␣of␣
        the␣dataset␣info␣json', default='/home/jonander/
        ObjectDetection/object_detection_project/dataset/
        dataset_info.json')
```

```
126     parser.add_argument('--split-dir', help='train/val/
            test␣jsons', default='/home/jonander/
            ObjectDetection/object_detection_project/dataset/
            train_val_test')
127     parser.add_argument('--json-image', help='jsons␣per␣
            image', default='/home/jonander/ObjectDetection/
            object_detection_project/dataset/json_per_image')
128     parser.add_argument('--output', help='path␣of␣the␣
            output', default='/home/jonander/ObjectDetection/
            object_detection_project/outputs/train/')
129
130     args = parser.parse_args()
131
132     params = load_params()
133     device,accelerator=config_device(params['device']) #
            Obtain device for training
134     input_size=params['model']['input_size']
135
136     os.makedirs(args.output, exist_ok=True)
137
138     """Set the random seed so all of our experiments can
            be reproduced"""
139     # sets seeds for numpy, torch, python.random and
            PYTHONHASHSEED
140     seed_everything(42, workers=True)
141
142     #GENERATE DATAMODULE
143     train_transform,test_transform=transformations(params
            ) # Data augmentation
144     data=PL_DataModule(args.images,args.split_dir,args.
            json_image,params,args.dataset_json,input_size,
            train_transform,test_transform)
145
146     # CREATE THE MODEL
147     num_classes=len(params['categories'])+1
148
149     # Define the object detection model
150     model = FasterRCNN(num_classes, params, training=True
            )
151     # CALLBACKS
152     callbacks_list= get_callbacks_list(params)
153     # END CALLBACKS
```

```
154
155     #LOGGER
156     # Save the training process metrics in a csv
157     logger = CSVLogger(args.output,version=0)
158
159     #TRAINER
160     trainer = get_trainer(params,device,accelerator,
            logger,callbacks_list)
161
162     # RUN TRAINING
163     trainer.fit(model=model, datamodule=data)
164
165     # If there is no callback to save the best weights,
            save checkpoint of the state of your last training
             epoch
166     if not(params['train']['restore_best_weights']):
167         trainer.save_checkpoint(args.output+"weights.ckpt
                ")
168     ckpt= torch.load(args.output+'weights.ckpt') #Load
            the saved checkpoint
169
170     model.load_state_dict(ckpt['state_dict'])
171     torch.save(model.state_dict(),args.output+'model.pt')
            #Save the model only with the weights
172     # END RUN TRAINING
173
174     # MODIFY LOGGING CSV STRUCTURE
175     # read the csv file
176     df = pd.read_csv(args.output+'lightning_logs/
            version_0/metrics.csv')
177
178     # Fill NaN with 0
179     df.fillna(0,inplace=True)
180
181     #Erase the step column
182     df.drop('step',inplace=True, axis=1)
183     #Erase first row
184     df = df.iloc[1:]
185     #Erase last row
186     df=df[:-1]
187
188     #Group together values of each epoch
```

```
189     df= df.groupby(['epoch'], sort=False).sum()
190
191     # writing into the file
192     df.to_csv(args.output+'lightning_logs/version_0/
            metrics.csv')
```

A.4 Archivo de *test*

```
1     import argparse
2 import os
3 import torch
4 import pandas as pd
5 import numpy as np
6 import cv2
7 import json
8
9 from utils.io import read_image,save_image,load_params
10 from utils.devices import config_device
11 from train import transformations,get_trainer
12 from datamodule import PL_DataModule
13 from model import FasterRCNN
14
15 from torch import nn
16 from pytorch_lightning import seed_everything
17
18 from sklearn.metrics import accuracy_score
19
20 if __name__ == '__main__':
21     parser = argparse.ArgumentParser(description='test')
22     parser.add_argument('--images', help='path␣of␣the␣
            images', default='/home/jonander/ObjectDetection/
            object_detection_project/dataset/images')
23     parser.add_argument('--dataset-json', help='path␣of␣
            the␣json␣with␣the␣dataset␣info', default='/home/
            jonander/ObjectDetection/object_detection_project/
            dataset/dataset_info.json')
24     parser.add_argument('--split-dir', help='train/val/
            test␣json', default='/home/jonander/
            ObjectDetection/object_detection_project/
            train_val_test')
```

```
25      parser.add_argument('--json-image', help='jsons␣per␣
           image', default='/home/jonander/ObjectDetection/
           object_detection_project/dataset/json_per_image')
26      parser.add_argument('--model', help='path␣of␣model',
           default='/home/jonander/ObjectDetection/
           object_detection_project/outputs/train/model.pt')
27      parser.add_argument('--output', help='path␣of␣the␣
           output', default='/home/jonander/ObjectDetection/
           object_detection_project/outputs/test')
28      parser.add_argument('--metrics', help='path␣of␣the␣
           metrics', default='/home/jonander/ObjectDetection/
           object_detection_project/metrics.json')
29
30      args = parser.parse_args()
31      dataset_json='/home/jonander/ObjectDetection/
           object_detection_project/dataset/dataset_info.json
           '
32      json_image='/home/jonander/ObjectDetection/
           object_detection_project/dataset/json_per_image'
33      params = load_params()
34
35      device,accelerator=config_device(params['device']) #
           Obtain device for training
36
37      os.makedirs(args.output, exist_ok=True)
38
39      """Set the random seed so all of our experiments can
           be reproduced"""
40      # sets seeds for numpy, torch, python.random and
           PYTHONHASHSEED
41      seed_everything(42, workers=True)
42
43      # List of categories with their corresponding indexes
44      category_list = params['categories']
45      input_size = params['model']['input_size']
46      threshold = params['test']['threshold']
47      category_list.sort()
48
49      # GENERATE DATAMODULE
50      train_transform,test_transform=transformations(params
           ) # Data augmentation
```

```
51      data=PL_DataModule(args.images,args.split_dir,
            json_image,params,dataset_json,input_size,
            train_transform,test_transform)
52      data.setup()
53
54      # Load the information of the test images to a python
            dictionary
55      test_split =  data.test_set.image_list
56      # dataset_info = data.test_set.dataset_info
57      # split_dir = data.test_set.split_dir
58
59      dataset_dict = {'input': [], 'filename': [], '
            category_index': [], 'bbox': []}
60
61      for i, filename in enumerate(test_split):
62          json_filename = f'test{i}.json'
63          json_path = os.path.join(json_image,
                json_filename)
64
65          # Cargar el contenido del archivo JSON
66          with open(json_path, 'r') as file:
67              json_data = json.load(file)
68
69          input_path = json_data['image_path']
70          input_name = json_data['image_filename']
71          category = json_data['category_id']
72          bounding_box = json_data['bboxes']
73
74          dataset_dict['input'].append(input_path)
75          dataset_dict['filename'].append(input_name)
76          dataset_dict['category_index'].append(category)
77          dataset_dict['bbox'].append(bounding_box)
78
79      # CREATE THE MODEL
80      num_classes=len(params['categories'])+1
81      # model = Classifier(num_classes,params,freeze=True,
            loss_fn=loss_fn)
82      training=False
83      model = FasterRCNN(num_classes, params, training)
84
85      # Load the weights (new fine-tuned model)
86      model.load_state_dict(torch.load(args.model))
```

```
87
88      # TRAINER
89      trainer = get_trainer(params,device,accelerator,
            logger=False,callbacks_list=None)
90
91      # OBTAIN PREDICTIONS
92      y_test_pred = trainer.predict(model, dataloaders=data
            .test_dataloader())
93      # y_test_pred = model.pred
94
95      # Postprocess the output, calculate the metrics and
            save the results
96      results_dict = {'gt': [], 'gt_index': [], 'pred': [],
            'pred_index': [], 'bbox': [], 'pred_bbox': []}
97
98      for i in dataset_dict['category_index']:
99          # Add category_index and category to the results
                dictionary
100         results_dict['gt_index'].append(i)
101         names = []
102         for j in i:
103             if j == 1:
104                 names.append('enchufe')
105             elif j == 2:
106                 names.append('interruptor')
107         results_dict['gt'].append(names)
108
109     for i in dataset_dict['bbox']:
110         # Add bbox to the results dictionary
111         results_dict['bbox'].append(i)
112
113     for i in range(len(y_test_pred)):
114
115         for j in range(y_test_pred[i][1]['scores']):
116         # Add predictions only if the score is more than
                the threshold
117             if y_test_pred[i][1]['scores'][j]>threshold:
118                 # Add category predictions to results
                        dictionary
119                 predicted_categories = y_test_pred[i][1][
                        'labels'][j]
```

```
120                 results_dict['pred_index'].append(
                        predicted_categories)
121                 names = []
122                 for j in predicted_categories:
123                     if j == 1:
124                         names.append('enchufe')
125                     elif j == 2:
126                         names.append('interruptor')
127                 results_dict['pred'].append(names)
128 
129                 # Add bbox predictions to the results
                        dictionary
130                 predicted_boxes = y_test_pred[i][1]['
                        boxes'][j]
131                 results_dict['pred_bbox'].append(
                        predicted_boxes)
132             else:
133                 continue
134 
135 
136     # Para guardar la accuracy, hay que guardar la
            correspondiente a la Classification y a la
            Regression
137     '''# Calculate the accuracy and save it in a json
138     accuracy = accuracy_score(results_dict['gt_index'],
            results_dict['pred_index'])
139     # accuracy = accuracy_score(results_dict['gt_index'],
             [idx[0] for idx in results_dict['pred_index']])
            '''
140     accuracy = 0
141     metrics_dict = {'Accuracy': accuracy}
142     with open(args.metrics, 'w+') as metrics_file:
143         json.dump(metrics_dict, metrics_file)
144 
145     # Create the csv with the results
146     os.makedirs(args.output, exist_ok=True)
147     csv_file_path = os.path.join(args.output, 'results.
            csv')
148     df_complete = list()
149     for i in range(len(y_test_pred)): #PORQUE SOLO
            PREDICE 8-9
150         df_row = list()
```

```
151         df_row.append(dataset_dict['input'][i])
152         df_row.append(results_dict['gt'][i])
153         df_row.append(results_dict['pred'][i])
154         df_row.append(results_dict['bbox'][i])
155         df_row.append(results_dict['pred_bbox'][i])
156         df_complete.append(df_row)
157     test_results_df = pd.DataFrame(df_complete, columns=[
        'image', 'Category', 'PredictedCategory', 'BBox',
        'PredictedBBox'])
158     test_results_df.to_csv(csv_file_path,index=False)
159
160     # Save the output images (image with the predicted
        label and bounding box)
161     dir_output_images = os.path.join(args.output, '
        output_images')
162     os.makedirs(dir_output_images, exist_ok=True)
163
164     for i in range(len(y_test_pred)):
165         if results_dict['gt'][i] != results_dict['pred'][
            i]:
166             original_image = read_image(dataset_dict['
                input'][i])
167             # image_resized = cv2.resize(original_image,
                (int(round(original_image.shape[1]/5)),
                int(round(original_image.shape[0]/5)))) #
                Resize a 130x130
168
169             # Convierte la lista en un tensor NumPy
170             pred_bbox_tensor = np.array(results_dict['
                pred_bbox'][i])
171             # Obt n el n mero de cajas delimitadoras
172             num_bboxes = pred_bbox_tensor.shape[0]
173
174             # Recorre las cajas delimitadoras y
                categor as
175             for j in range(num_bboxes):
176                 # Obt n las coordenadas de la caja
                    delimitadora como una lista de listas
177                 bbox = pred_bbox_tensor[j].tolist()
178                 x1, y1, x2, y2 = bbox
179
```

```
180                 # Dibuja el rect ngulo rojo en la imagen
                        original
181                 cv2.rectangle(original_image, (int(x1),
                        int(y1)), (int(x2), int(y2)), (0, 0,
                        255), 2)
182
183                 # Agrega el texto de la categor a en la
                        esquina superior izquierda del
                        rect ngulo
184                 category = results_dict['pred'][i][j]
185                 cv2.putText(original_image, category, (
                        int(x1), int(y1)-10), cv2.
                        FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255)
                        , 2)
186
187             image_name = os.path.basename(dataset_dict['
                    filename'][i])
188             #image_dir = os.path.join(dir_output_images)
                    # , results_dict['gt'][i]
189             #os.makedirs(image_dir, exist_ok=True)
190             save_image(os.path.join(dir_output_images,
                    image_name), original_image)
```