Comparación de técnicas de detección de objetos en imágenes
Comparison of object detection techniques in images

Trabajo de Fin de Máster
Curso 2022–2023

Autor
José Luis Martín Pérez

Director
Gonzalo Pajares Martinsanz

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Comparación de técnicas de detección de objetos en imágenes
Comparison of object detection techniques in images

Trabajo de Fin de Máster en Ingeniería Informática
Departamento de Ingeniería del Software e Inteligencia Artificial

Autor
José Luis Martín Pérez

Director
Gonzalo Pajares Martinsanz

Convocatoria: *Septiembre 2023*
Calificación: *9.0*

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

5 de septiembre de 2023

Autorización de difusión

El abajo firmante, matriculado en el Máster en Ingeniería en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Comparación de técnicas de detección de objetos en imágenes", realizado durante el curso académico 2022-2023 bajo la dirección de Gonzalo Pajares Martinsanz en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

José Luis Martín Pérez

25 de septiembre de 2023

Dedicatoria

Quiero dedicar este trabajo a mi familia y amigos, quienes me han apoyado incondicionalmente a lo largo de estos años. Y a mi pareja, quien se ha convertido en mi fuente de inspiración. Sin vosotros a mi lado, no habría podido lograr esto. Gracias por todo.

Agradecimientos

Quiero agradecerle a Gonzalo Pajares Martínsanz que me haya permitido realizar este trabajo con él. Por ayudarme en la realización de la memoria y permitirme ampliar mi conocimiento en el campo del Aprendizaje Profundo.

Resumen

Comparación de técnicas de detección de objetos en imágenes

El presente trabajo se sitúa en el campo del *Aprendizaje Profundo* (AP), donde los modelos de Redes Neuronales Artificiales han experimentado una notable evolución y se han convertido en elementos fundamentales para abordar desafíos de alta complejidad, tales como el reconocimiento de imágenes o el procesamiento del lenguaje natural.

Este proyecto surge del interés en comprender en profundidad las técnicas de detección de objetos mediante AP. La detección de objetos es un método o técnica de visión artificial que permite reconocer y localizar objetos en imágenes o videos mediante Redes Neuronales Artificiales.

Este trabajo se inicia con la selección y la posterior adaptación de un conjunto de datos que se utiliza para realizar el entrenamiento y la validación de los modelos. Para ello se ha realizado la implementación del detector de objetos de una etapa, *You Only Look Once* (YOLO), y de dos etapas, *Regiones con CNN* (R-CNN). Cada detector se ha entrenado y validado, utilizando como columnas vertebrales las arquitecturas de Redes Neuronales Artificiales AlexNet, MobileNetV2 y ResNet-50, con el fin de obtener el modelo con mejores resultados de cada detector de objetos. Finalmente, se trató de mejorar el modelo de mayor rendimiento de cada detector de objetos ajustando los hiperparámetros de estos.

Palabras clave

Inteligencia Artificial, Aprendizaje Automático, Aprendizaje Profundo, Redes Neuronales Artificiales, Redes Neuronales Convolucionales, Detección de Objetos, Matlab.

Abstract

Comparison of object detection techniques in images

This work is situated in the field of *Deep Learning* (DL), where *Artificial Neural Network* (ANN) models have evolved significantly and have become essential elements in addressing highly complex challenges, such as image recognition and natural language processing.

This project arises from the interest in gaining a deep understanding of object detection techniques through DL. Object detection is a method or technique in computer vision that enables the recognition and localization of objects in images or videos using ANNs.

This study begins with the selection and subsequent adaptation of a dataset, which is used for training and validating the models. For this purpose, have been implemented a single-stage and two-stage object detectors, YOLO and R-CNN respectively. Each object detector has been trained and validated, using the backbone architectures of the Artificial Neural Networks AlexNet, MobileNetV2, and ResNet-50, with the aim of obtaining the best-performing model for each object detector. Finally, an attempt to improve the results of the top-performing model of each object detector was made by fine-tuning the hyperparameters.

Keywords

Artificial Intelligence, Machine Learning, Deep Learning, Artificial Neural Networks, Convolutional Neural Networks, Object Detection, Matlab.

Índice

1. Introducción **1**

1.1. Contexto 1

1.2. Objetivos 3

1.3. Motivación 3

1.4. Plan de trabajo 4

1.5. Organización de la memoria 5

1. Introduction **7**

1.1. Context 7

1.2. Objectives 8

1.3. Motivation 9

1.4. Work Plan 10

1.5. Report Organization 11

2. Fundamentos teóricos **13**

2.1. Conceptos básicos en las Redes Neuronales Convolucionales 13

2.1.1. ¿Qué es una red neuronal artificial? 13

2.1.2. Arquitectura de una red neuronal artificial multicapa 15

2.1.3. ¿Cómo aprende una red neuronal artificial? 16

2.2. Redes neuronales convolucionales (RNC) 20
2.2.1. Arquitectura de una red neuronal convolucional 20
2.3. Detectores de objetos 27
2.3.1. ¿Qué son los detectores de objetos? 27
2.4. Entrenamiento y validación 36
2.4.1. Hiperparámetros 37
2.4.2. Data augmentation 37
2.4.3. Validación 38
2.4.4. Transfer Learning 38

3. Diseño y desarrollo de la aplicación **41**
3.1. Recursos utilizados 41
3.1.1. Conjunto de datos 41
3.1.2. Visual Studio Code 45
3.1.3. Python 45
3.1.4. Matlab 46
3.1.5. ClickUp 46
3.1.6. Hardware 47
3.1.7. Librerías 47
3.2. Implementación de los detectores de objetos 48
3.2.1. Carga de los conjuntos de datos 48
3.2.2. Implementación del detector de objetos R-CNN 49
3.2.3. Implementación del detector de objetos YOLO 49

4. Análisis de resultados **51**
4.1. Resultados 51
4.1.1. R-CNN 52
4.1.2. YOLO 58

5. Conclusiones y Trabajo Futuro **61**

5. Conclusions and Future Work **63**

Bibliografía **65**

A. Código fuente **71**

B. Descarga e instalación del proyecto **75**

Índice de figuras

1.1. Comparación de la programación clásica y el aprendizaje automático 2

1.1. Comparison of classical programming and machine learning 8

2.1. Componentes de un perceptrón. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 14

2.2. Arquitectura del modelo *Multilayer Perceptron* (MLP). Fuente: Aprendizaje Profundo (Pajares et al., 2021) 16

2.3. Gradiente descendente. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 17

2.4. Convolución bidimensional. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 22

2.5. Ejemplo convolución bidimensional para N=2, i1=i2=5, k1=k2=3, s1=s2=1, p1=p2=0. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 22

2.6. Ejemplo de agrupamiento máximo con relleno con ceros. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 23

2.7. Invarianza del agrupamiento máximo. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 23

2.8. Ejemplo invarianza aprendida. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 24

2.9. Ejemplo de sobreajuste. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 25

2.10. Dropout. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 25

2.11. Esquema general de los detectores de objetos. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 27

2.12. Índice *Intersección sobre la unión* (IoU). Fuente: Aprendizaje Profundo (Pajares et al., 2021) 28

2.13. Asignación de *ground truth bounding boxes* a *anchor boxes*. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 31

2.14. Índice IoU. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 32

2.15. Arquitectura del detector R-CNN. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 33

2.16. Arquitectura de la *Red Neuronal Convolucional* (RNC) del detector YOLO. Fuente: Aprendizaje Profundo (Pajares et al., 2021) 35

2.17. Esquema de la transferencia de aprendizaje. Fuente: MathWorks (2023j). . . 38

3.1. Número de instancias por categoría. Fuente: *Microsoft Common Objects in Context* (MS COCO) (Lin et al., 2015) 42

3.2. Número de instancias por imagen. Fuente: MS COCO (Lin et al., 2015) . . 42

3.3. Número de categorías frente a número de instancias por categoría. Fuente: MS COCO (Lin et al., 2015) 42

4.1. Entrenamiento del detector de objetos R-CNN utilizando la arquitectura AlexNet 53

4.2. Entrenamiento del detector de objetos R-CNN utilizando la arquitectura MobileNetV2 55

4.3. Entrenamiento del detector de objetos R-CNN utilizando la arquitectura ResNet-50 56

4.4. Entrenamiento del detector de objetos YOLO utilizando la arquitectura ResNet-50 59

A.1. Estructura del fichero que contiene las anotaciones del *dataset* MS COCO . 71

A.2. Estructura de una anotación del *dataset* MS COCO 72

A.3. Carga del modelo preentrenado con la arquitectura de RNC AlexNet para realizar la transferencia de aprendizaje en Matlab 72

A.4. Función que permite entrenar el detector de objetos R-CNN dadas unas opciones de entrenamiento 73

A.5. Proceso de validación del rendimiento del detector de objetos R-CNN 74

B.1. Generación de los conjuntos de datos de entrenamiento, validación y test . . 75

B.2. Documentos en formato *Comma-Separated Values* (CSV) que contienen las *ground truth tables* 76

B.3. Carga de los conjuntos de datos 76

B.4. Añadir ruta del directorio donde se encuentran las imágenes de los conjuntos de datos 76

B.5. Cambio de la arquitectura de RNC utilizada como columna vertebral por el detector de objetos 76

B.6. Parámetros de entrenamiento modificables 77

Índice de tablas

4.1. Arquitecturas de RNC por detector de objetos 51

4.2. Hiperparámetros utilizados en el entrenamiento de los modelos del detector de objetos R-CNN 52

4.3. Métricas obtenidas durante el entrenamiento del detector de objetos R-CNN utilizando la arquitectura AlexNet 53

4.4. Precisión media obtenida por clase durante la evaluación del detector de objetos R-CNN utilizando la arquitectura AlexNet 54

4.5. Métricas obtenidas durante el entrenamiento del detector de objetos R-CNN utilizando la arquitectura MobileNetV2 55

4.6. Precisión media obtenida por clase durante la evaluación del detector de objetos R-CNN utilizando la arquitectura MobileNetV2 55

4.7. Métricas obtenidas durante el entrenamiento del detector de objetos R-CNN utilizando la arquitectura ResNet-50 57

4.8. Precisión media obtenida por clase durante la evaluación del detector de objetos R-CNN utilizando la arquitectura ResNet-50 57

4.9. *mean Average Precision* (mAP) por optimizador del detector de objetos R-CNN utilizando la arquitectura ResNet-50 58

4.10. mAP por tamaño del lote del detector de objetos R-CNN utilizando la arquitectura ResNet-50 58

4.11. Hiperparámetros utilizados en el entrenamiento de los modelos del detector de objetos YOLO 59

4.12. Métricas obtenidas durante el entrenamiento del detector de objetos YOLO utilizando la arquitectura ResNet-50 60

4.13. Precisión media obtenida por clase durante la evaluación del detector de objetos YOLO utilizando la arquitectura ResNet-50 60

Capítulo 1

Introducción

"A veces la persona que nadie imagina capaz de nada es la que hace cosas que nadie imagina"
— Alan Turing

1.1. Contexto

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

La *Inteligencia Artificial* (IA) se materializó y el término fue acuñado en la década de 1950 (Chollet, 2017). Nace de la cuestión que se hacen los pioneros de la informática: «¿Pueden pensar las máquinas?». Alan Turing sugirió que, al igual que el ser humano, las máquinas podrían resolver problemas y tomar decisiones utilizando la información disponible y la razón. Según McCarthy (2004), la inteligencia artificial es «la ciencia y la ingeniería de la fabricación de máquinas inteligentes, especialmente programas informáticos inteligentes que realizan tareas propias del pensamiento humano». Está relacionada con la tarea similar de usar computadoras para entender la inteligencia humana, pero la IA no tiene que limitarse a métodos que son biológicamente observables. La IA es un campo de la informática que abarca el *Aprendizaje Automático* (AA) o *Machine Learning* (ML) y el AP, entre otros, si bien existen otras ramas que no implican aprendizaje propiamente dicho. Los primeros programas de ajedrez solo eran una serie de reglas. Durante un largo periodo de tiempo se creyó que si se creaba un conjunto de reglas suficientemente grande podrían replicar la inteligencia del ser humano. A esto se le conoce como IA simbólica y fue ampliamente utilizada desde 1950 hasta finales de la década de 1980. Aunque la IA era una estrategia adecuada para resolver problemas lógicos, resultó ineficaz ante otros problemas como la clasificación de imágenes o el procesamiento del lenguaje natural. Es por esto por lo que surgió lo que se conoce como ML.

El ML surge de la cuestión de si una máquina puede aprender a realizar tareas bien de forma supervisada o no supervisada. En vez de escribir un conjunto de reglas que

determinen cómo se deben procesar los datos, se cuestionan si la máquina podría aprender dichas reglas observando los datos. La figura 1.1 muestra un esquema ilustrativo diferencial entre la programación clásica y el AA.

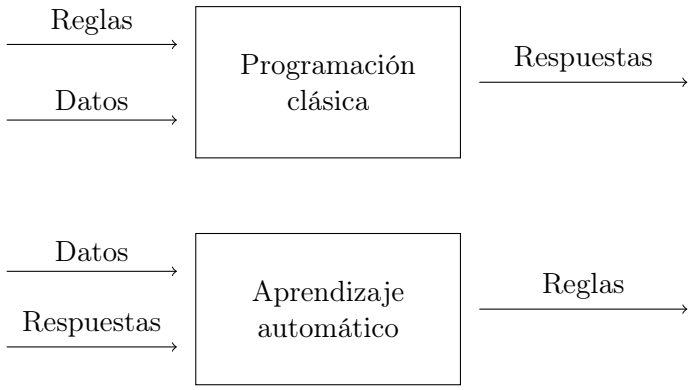

Figura 1.1: Comparación de la programación clásica y el aprendizaje automático

Los sistemas de AA además de programarse, se entrenan. A partir de un conjunto de datos etiquetados, el sistema aprenderá las reglas o mecanismos que asocian los datos con las etiquetas. Aunque hasta la década de 1990 el AA no se extendió con énfasis, se ha convertido en un campo muy popular de la IA debido al incremento de los recursos computacionales, en particular por el hecho de que las memorias de computación son cada vez más rápidas (Ley de Moore) y con más prestaciones, incluyendo su capacidad de almacenamiento. Otro hecho clave en el desarrollo de la IA ha sido el hecho de que cada vez se dispone de conjuntos de datos cada vez más grandes.

Dentro del AA se ubica el AP que, no adquirió relevancia hasta la década de 2010. En esta rama se han logrado los siguientes avances:

- Clasificación de imágenes, reconocimiento del lenguaje y transcripción de texto escrito a nivel casi humano. Destacan aquí los modelos generativos cada vez con mayor potencialidad.
- Mejora de la traducción automática, de la conversión de texto a voz y de la segmentación publicitaria.
- Mejora en los motores de búsqueda.

El AP se puede aplicar a multitud de problemas, progresando a pasos agigantados, y expandiéndose con profusión en procesos industriales, ámbito este donde está llamado a desempeñar un importante papel. En términos generales, en la actualidad, la IA «ayuda» en determinadas tareas (cada vez más), siendo una herramienta más de las que dispone el ser humano. Sin embargo, todo apunta a que acabará aplicándose a la mayoría de procesos que conforman y se encuentran implantados en nuestra sociedad y nuestra vida cotidiana, al igual que hoy en día ocurre con Internet.

1.2. Objetivos

El objetivo principal de este proyecto consiste en investigar y analizar el funcionamiento de los detectores de objetos en imágenes, bajo el paradigma del AP, utilizando un conjunto de imágenes etiquetadas convenientemente. Para ello se han elegido una serie de herramientas y tecnologías que se explicarán en detalle en el punto 3.1 de este documento. Como lenguaje de programación principal se ha decidido utilizar Matlab, ya que nos proporciona herramientas y un *Integrated Development Environment* (IDE).

Bajo este planteamiento, se plantean los siguientes objetivos específicos, que se han dividido en dos grupos bien diferenciados: investigación e implementación.

- Investigación
  - Exploración de conjuntos de datos y selección del más adecuado para el entrenamiento y validación de los modelos de detección de objetos.
  - Estudio previo de las herramientas utilizadas.
  - Estudio de los fundamentos del aprendizaje profundo para comprender el funcionamiento de los detectores de objetos.
  - Comprensión de la documentación de Matlab relacionada con el entrenamiento y validación de los detectores.
  - Análisis de ejemplos de Matlab para entender cómo se entrenan y validan los detectores de objetos.
- Implementación
  - Limpieza y adaptación del conjunto de datos para compatibilizarlo con la entrada de los detectores de objetos.
  - Adaptación de los ejemplos de Matlab para utilizar el conjunto de datos seleccionado.
  - Entrenar y validar modelos de detección de objetos utilizando el conjunto de datos seleccionado.
  - Evaluar el rendimiento de los mejores modelos en los detectores de objetos, comparando los resultados con las implementaciones base

1.3. Motivación

La motivación principal surge por el interés de conocer conceptualmente y en funcionamiento las técnicas de detección de objetos mediante AP.

La motivación previa se refuerza por el hecho de disponer de un IDE muy completo perfecto para realizar el entrenamiento y la validación de los detectores de objetos. Como lenguaje de programación auxiliar se ha utilizado Python, que ha permitido adaptar el conjunto de datos para compatibilizarlo con los detectores de objetos proporcionados por Matlab. Además, se ha utilizado *Visual Studio Code* (VSCode) para realizar la programación con Python, ya que proporciona un entorno completo y ligero. Una vez identificado

el objetivo principal y las herramientas y tecnologías a utilizar, conviene desgranarlo en objetivos más pequeños para finalizar el proyecto de forma satisfactoria.

Históricamente un elemento motivacional surge de los aspectos curriculares de formación previa. En efecto, debido al reducido número de asignaturas relacionadas con la IA vistas en el grado de Ingeniería Informática, apenas tuve la oportunidad de profundizar en la rama de la IA y más concretamente en el AP. En el grado, más concretamente en la asignatura «Fundamentos de Sistemas Inteligentes», se estudiaron algoritmos de búsqueda como A^* o *Backtracking*, también se introdujo el lenguaje de programación Lisp que se basa en la manipulación de listas, es decir, los programas y los datos se representan mediante listas y la herramienta *C Language Integrated Production System* (CLIPS) que permite la creación de sistemas expertos basados en reglas. Como se puede ver, los conceptos estudiados son básicos y no existían otras asignaturas que profundizaran en el AA y menos en el AP. Por este motivo el Trabajo de Fin de Grado consistió en una aplicación destinada a la clasificación de imágenes y así poder adquirir conocimientos especializados en una rama hasta ese momento desconocida para mí.

En el Trabajo de Fin de Grado creé una aplicación de escritorio orientada al sector de la salud que permitiera clasificar imágenes. Para la clasificación de imágenes desarrollé un modelo de RNC utilizando el lenguaje de programación Python y el *framework* Keras. Aprendí los conceptos básicos del AP y me sirvió como base para poder plantear el presente trabajo. Si bien, ahora se trata de dar un paso más allá para enfocarse en la detección de objetos en imágenes en vez de la clasificación de imágenes. La detección de objetos en imágenes es un tema de especial interés motivador, en particular para entender cómo los detectores son capaces de localizar en una imagen los objetos a detectar.

1.4. Plan de trabajo

Durante el desarrollo del proyecto se ha seguido un plan de trabajo con el objetivo de establecer las tareas, tiempos y alcance de su desarrollo y así poder conseguir los objetivos mencionados anteriormente. Para ello se ha utilizado una herramienta de gestión de proyectos llamada ClickUp (2023). Se trata de una herramienta de gestión de proyectos que proporciona la posibilidad de realizar multitud de actividades tales como gestión de tareas, seguimiento del tiempo o administrar proyectos entre otras cosas. Aunque en este caso el proyecto era un trabajo individual, tener un plan de trabajo claro ha facilitado el desarrollo y el cumplimiento de los objetivos. El plan de trabajo se ha dividido en los siguientes puntos:

- Reuniones iniciales con el tutor: al comienzo del curso académico se realizaron reuniones junto con el tutor para decidir el tema a tratar y las tecnologías a utilizar. Una vez decidido el tema, la comunicación con el tutor se decidió realizarla bien presencial u *on-line*, a conveniencia.
- Estudio de las tecnologías a utilizar: una vez decididas las tecnologías a utilizar se llevó a cabo el estudio y búsqueda de recursos para reutilizar material existente y agilizar el desarrollo del proyecto. Se comenzó con el estudio de las tecnologías y la lectura del libro titulado «Aprendizaje Profundo» (Pajares et al., 2021). Posterior-

mente se amplió el conocimiento con recursos online y se buscó documentación acerca de los detectores a estudiar.

- División de tareas: antes de comenzar con el desarrollo de la aplicación se decidió dividir el alcance en diferentes tareas estimando tiempo y fecha de cumplimiento para poder realizar un seguimiento del estado del proyecto en todo momento.
- Entrenamiento: una vez realizada toda la planificación se comenzó con el diseño y entrenamiento de los detectores seleccionados. Esta tarea se paralelizó con el estudio y profundización en la comprensión y funcionamiento de los detectores. Una vez los entrenamientos terminaron de ejecutarse, se procedió al estudio de los resultados y, tras esto, se ajustaron los hiperparámetros para obtener mejores resultados. Por último, se extrajeron las conclusiones a partir de los resultados obtenidos durante la evaluación de los modelos.
- Escritura de la memoria: durante los entrenamientos de los detectores, se fue elaborando la memoria en la que se plasman los conocimientos adquiridos durante la realización del proyecto y los resultados y conclusiones obtenidas a partir de las pruebas realizadas con los modelos de detección de objetos.
- Revisión final y cierre del proyecto: una vez terminada la aplicación y escrita la memoria, se procedió a la revisión y corrección de ambas. Antes de la entrega final y la finalización del proyecto, se procedió a realizar las modificaciones pertinentes tanto a la aplicación como a la memoria propiamente dicha atendiendo a las sugerencias proporcionadas por el tutor tras su corrección.

En definitiva, tener un plan de trabajo bien estructurado con los pasos a seguir y una organización sólida ha permitido conseguir concluirlo de forma satisfactoria. Gracias a la división de tareas y a la estimación del tiempo en cada una de ellas, se ha podido avanzar de manera eficiente y cumplir con los hitos establecidos en el tiempo límite marcado.

1.5. Organización de la memoria

La memoria se compone de los siguientes capítulos:

1. Introducción: se expone el tema a tratar en este trabajo junto a los motivos de realización y los objetivos de este.
2. Fundamentos teóricos: se definen los conceptos necesarios para comprender los modelos de detección de objetos bajo el paradigma del AP.
3. Diseño y desarrollo de la aplicación: se describen los recursos utilizados durante la realización del trabajo, así como la implementación de los modelos desarrollados.
4. Análisis de resultados: se analizan y comentan los resultados obtenidos tras el entrenamiento y la validación de los modelos de detección de objetos y se explica y determina el conjunto de hiperparámetros que obtiene el mejor resultado.

5. Conclusiones y trabajo futuro: se exponen las conclusiones extraídas de la realización del trabajo, así como las diferentes líneas de mejora propuestas para solventar los problemas detectados o mejorar la solución propuesta.

Chapter 1

Introduction

"Sometimes the person no one imagines capable of anything is the one who does things that no one can imagine."
— Alan Turing

1.1. Context

Artificial Intelligence (AI) materialized, and the term was coined in the 1950s (Chollet, 2017). It arose from the question posed by computer pioneers: «Can machines think?» Alan Turing suggested that, like humans, machines could solve problems and make decisions using available information and reasoning. According to McCarthy (2004), artificial intelligence is «the science and engineering of making intelligent machines, especially intelligent computer programs that perform tasks typical of human thinking». It is related to the similar task of using computers to understand human intelligence, but AI does not have to be limited to biologically observable methods. AI is a field of computer science that encompasses ML and deep learning, among others, although there are other branches that do not involve proper learning. Early chess programs were just a set of rules. For a long time, it was believed that if a large enough set of rules could be created, they could replicate human intelligence. This is known as symbolic AI and was widely used from the 1950s until the late 1980s. Although AI was a suitable strategy for solving logical problems, it proved ineffective for other tasks such as image classification or natural language processing. This is why ML emerged.

ML arises from the question of whether a machine can learn to perform tasks, whether supervised or unsupervised. Instead of writing a set of rules to determine how data should be processed, the question is whether the machine could learn these rules by observing the data. Figure 1.1 illustrates the key difference between classical programming and ML.

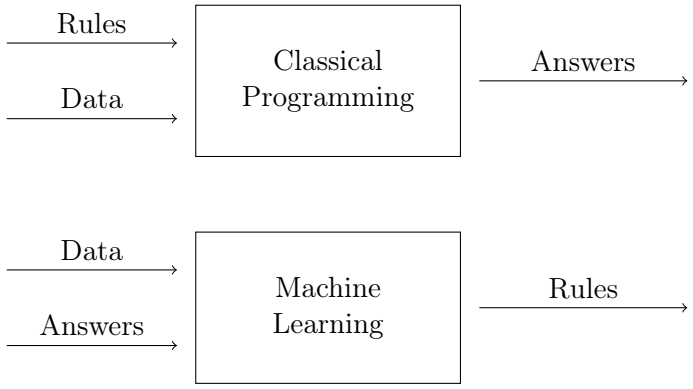

Figure 1.1: Comparison of classical programming and machine learning

ML systems are not just programmed; they are trained. Using a labeled dataset, the system learns the rules or mechanisms that associate the data with labels. While ML did not gain prominence until the 1990s, it has become a very popular field of AI due to increased computational resources, particularly the fact that computer memories are becoming faster (Moore's Law) and more powerful, including their storage capacity. Another key development in AI has been the availability of increasingly large datasets.

Within ML is DL, which did not gain relevance until the 2010s. In this field, the following advances have been made:

- Image classification, natural language recognition, and almost human-level text transcription. Notable are the generative models with increasing potential.
- Improvement in automatic translation, text-to-speech conversion, and advertising segmentation.
- Enhancement of search engines.

DL can be applied to a multitude of problems, progressing rapidly and proliferating in industrial processes, an area where it is poised to play a significant role. In general terms, currently, AI «assists» in certain tasks (increasingly), becoming one of the tools available to humans. However, everything points to it eventually being applied to most processes that shape and are integrated into our society and daily life, much like the Internet does today.

1.2. Objectives

The main objective of this project is to investigate and analyze the operation of object detectors in images, under the DL paradigm, using a set of appropriately labeled images. To achieve this, a series of tools and technologies have been chosen, which will be explained in detail in section 3.1 of this document. Matlab has been chosen as the primary programming language, as it provides us with tools and an IDE.

Under this approach, the following specific objectives have been set, divided into two distinct groups: research and implementation.

- Research
  - Exploration of datasets and selection of the most suitable one for training and validating object detection models.
  - Preliminary study of the tools used.
  - Study of the fundamentals of deep learning to understand how object detectors work.
  - Understanding Matlab documentation related to training and validating object detectors.
  - Analysis of Matlab examples to understand how object detectors are trained and validated.
- Implementation
  - Cleaning and adapting the dataset to make it compatible with the input of object detectors.
  - Adapting Matlab examples to use the selected dataset.
  - Training and validating object detection models using the selected dataset.
  - Evaluating the performance of the best models in object detection, comparing the results with baseline implementations.

1.3. Motivation

The main motivation arises from the interest in conceptually understanding and working with object detection techniques using deep learning.

The previous motivation is reinforced by the availability of a comprehensive IDE, which is perfect for training and validating object detectors. Python has been used as the auxiliary programming language, allowing the dataset to be adapted for use with Matlab's object detectors. Additionally, VSCode has been used for Python programming, providing a comprehensive and lightweight environment. Once the main objective and the tools and technologies to be used have been identified, it is advisable to break it down into smaller objectives to successfully complete the project.

Historically, motivational elements arise from the curricular aspects of previous education. Indeed, due to the limited number of AI-related subjects in the Computer Science degree, there was hardly any opportunity to delve into the field of AI, particularly in DL. In the degree program, specifically in the course «Foundations of Intelligent Systems», algorithms such as A^* or Backtracking were studied, and the Lisp programming language was introduced, which is based on list manipulation, where programs and data are represented using lists. The CLIPS tool, which allows the creation of rule-based expert systems, was also introduced. As can be seen, the concepts studied were basic, and there were no other courses that delved into ML or DL. For this reason, the Bachelor's Thesis focused

on image classification as an opportunity to acquire specialized knowledge in a previously unfamiliar field.

In the Bachelor's Thesis, a desktop application was created for the healthcare sector to classify images. For image classification, a *Convolutional Neural Network* (CNN) model was developed using Python and the Keras framework. Basic concepts of deep learning were learned, serving as a foundation for the current project. However, the current project aims to go a step further and focus on object detection in images instead of image classification. Object detection in images is a motivating topic, especially to understand how detectors can locate objects in an image.

1.4. Work Plan

During the project's development, a work plan was followed to establish tasks, timelines, and the scope of the project's development, enabling the achievement of the previously mentioned objectives. A project management tool called ClickUp was used for this purpose. ClickUp is a project management tool that offers a wide range of capabilities, such as task management, time tracking, and project administration, among other things. Although this project was an individual effort, having a well-structured work plan facilitated project development and objective fulfillment have a really positive impact in the project. The work plan was divided into the following points:

- Initial meetings with the advisor: At the beginning of the academic year, meetings were held with the advisor to decide on the topic to address and the technologies to use. Once the topic was chosen, communication with the advisor was conducted either in person or online, as needed.
- Study of technologies to use: After deciding on the technologies to use, the study and search for resources to reuse existing materials and expedite project development were carried out. The study of technologies began, with the reading of the book titled «Deep Learning» (Pajares et al., 2021). Subsequently, knowledge was expanded through online resources, and search for documentation regarding the detectors to be studied was carried out.
- Task division: Before starting developing, the scope was divided into different tasks, estimating time and completion dates to track the project's status at all time.
- Training: Once all the planning was done, the design and training of the selected detectors began. This task was parallelized with the study and deepening of the understanding and operation of the detectors. After the training runs were completed, the results were studied, and hyperparameters were adjusted to achieve better results. Finally, conclusions were drawn from the results obtained during model evaluation.
- Writing the report: During the training of the detectors, the report was gradually developed to document the knowledge acquired during the project and the results and conclusions obtained from testing the object detection models.
- Final review and project closure: After completing the developed application and writing the report, both were reviewed and corrected. Before the final delivery and

the project completion, the necessary modifications were made to both the application and the report based on the suggestions provided by the advisor after he review.

In summary, having a well-structured work plan with clear steps and solid organization enabled successful project completion. Thanks to task division and time estimation for each task, progress was made efficiently, and the established milestones were met within the set deadline.

1.5. Report Organization

The report consists of the following chapters:

1. Introduction: This chapter presents the topic of this work, along with the reasons for its execution and its objectives.
2. Theoretical Foundations: This section defines the necessary concepts to understand object detection models under the Deep Learning (DL) paradigm.
3. Application Design and Development: This chapter describes the resources used in the execution of the work, as well as the implementation of the developed models.
4. Results Analysis: In this section, the analysis and discussion of the results obtained after training and validating the object detection models were carried out, as well as the explication and the selection of the hyperparameters that achieve the best result.
5. Conclusions and Future Work: This chapter presents the conclusions drawn from the work and proposes various lines of improvement to address the detected problems or enhance the proposed solution.

Capítulo 2

Fundamentos teóricos

El capítulo se enfoca en explicar los conceptos necesarios para comprender los modelos de detección de objetos bajo el paradigma del AP, de tal manera que comprenden desde los conceptos básicos como «¿Qué es una neurona artificial?» hasta conceptos más complejos y abstractos como «¿Qué son los detectores de objetos en imágenes basados en Redes Neuronales Convolucionales (CNN, *Convolutional Neural Networks*)»?. El capítulo está estructurado de forma que en primer lugar se introducen los conceptos básicos relativos al concepto de *Redes Neuronales Artificiales* (RNA) en su planteamiento clásico, exponiendo los conceptos fundamentales válidos para las RNC. A continuación, se explican los conceptos relativos a las RNC en exclusiva, siguiendo la referencia Pajares et al. (2021).

2.1. Conceptos básicos en las Redes Neuronales Convolucionales

2.1.1. ¿Qué es una red neuronal artificial?

Una RNA es un modelo matemático que se inspira en las redes neuronales biológicas. Las RNAs están compuestas por neuronas artificiales, también conocidas como perceptrones, que se encuentran interconectadas para trasmitirse señales (Yu y Malan, 2020).

Un perceptrón es el modelo más básico de red neuronal artificial y está compuesto por una serie de elementos bien definidos que se puede observar en la figura 2.1.

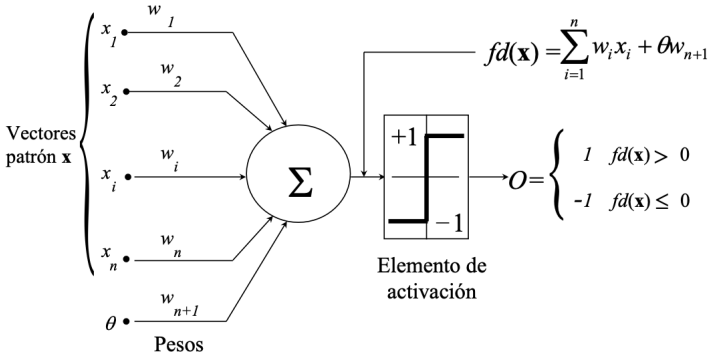

Figura 2.1: Componentes de un perceptrón. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

- Entradas: Un perceptrón recibe una serie de entradas que representan características y cada entrada se asocia con un peso.
- Pesos: Los pesos representan la importancia de cada entrada en la salida del perceptrón. Durante el proceso de entrenamiento los pesos son ajustados para que la RNA aprenda.
- Sesgo (*Bias*): A veces, cambiar el origen de la función de activación puede ser beneficioso para los modelos. Como cambiar los valores de los pesos solo modifica la inclinación de la curva, se introduce el sesgo para permitir este desplazamiento.
- Función potencial de entrada: El potencial de entrada es el resultado de la suma de las señales de entradas por sus correspondientes pesos siendo una de las entradas el *bias* θ como se puede ver en la ecuación 2.1.

$$fd(x) = \sum_{i=1}^{n}(w_i x_i) + \theta w_{n+1} \tag{2.1}$$

- Función de activación: La función de activación se encarga de determinar si el perceptrón debe activarse o no en función del potencial de entrada. Las funciones de activación más utilizadas son:
  - Escalón (*Step*): La función escalón o de paso es una función no lineal definida por la ecuación 2.2. La función asigna el valor 0 a todas las entradas que no alcancen el umbral establecido y el valor 1 a todas aquellas entradas que lo alcancen o superen.

$$f(x) = \begin{cases} 0 & \text{si } x < threshold \\ 1 & \text{si } x \geq threshold \end{cases} \tag{2.2}$$

  - Sigmoide (*Sigmoid*): La función sigmoide o sigmoidal es una función no lineal cuyo comportamiento viene definido por la ecuación 2.3. La salida es un número real perteneciente al intervalo [0, 1].

$$f(\alpha, x, c) = \frac{1}{1 + e^{-\alpha(x-c)}} \tag{2.3}$$

- Tangente Hiperbólica (Tanh, *Hyperbolic Tangent*): La función Tanh posee un comportamiento similar a la función sigmoide 2.3, pero el intervalo de valores que presenta en este caso es [-1, 1]. La función está definida por la ecuación 2.3.

$$tanh(x) = 2sigmoid(2x) - 1 \tag{2.4}$$

- Unidad Lineal Rectificada (ReLU, *Rectified Linear Unit*): La función ReLU es una función no lineal y eficiente debido a su simplicidad como se puede observar en la ecuación 2.5. El intervalo de valores producido es [0, 1] siendo el valor 0 utilizado para representar los valores negativos y 1 para los valores positivos.

$$f(x) = max(0, x) \tag{2.5}$$

- Exponencial normalizada (*Softmax*): La función *softmax* es una función no lineal que se utiliza en problemas de clasificación multiclase. A partir de un vector de valores reales obtiene la probabilidad por cada clase donde la suma de todas las probabilidades tiene que ser 1.

$$softmax(x)_i = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)} \tag{2.6}$$

- Salida: La salida es el resultado obtenido tras transformar las entradas.

2.1.2. Arquitectura de una red neuronal artificial multicapa

El perceptrón tiene limitaciones en la capacidad de aprendizaje ya que solo puede resolver problemas de clasificación linealmente separables. La separación lineal establece que cuando se tienen dos clases, existe un punto, línea, plano o hiperplano que puede dividir las características de entrada de manera que todos los puntos de una clase se encuentren en un lado del espacio, mientras que los de la segunda clase se encuentren en el otro lado. Por lo que el perceptrón no podrá aprender relaciones no lineales más complejas. Para poder resolver problemas no linealmente separables, Rumelhart et al. (1986) presentaron la «Regla Delta Generalizada» para aprendizaje por retropropagación. La regla permitió trabajar con múltiples capas y con funciones de activación no lineales como las vistas en el punto anterior 2.1.1 y dio lugar al Perceptrón Multicapa (MLP, *Multilayer Perceptron*).

La arquitectura del perceptrón multicapa, que se puede observar en la figura 2.2, representa un modelo totalmente conectado, es decir, cada perceptrón de una capa está conectado a todos los perceptrones de la capa siguiente.

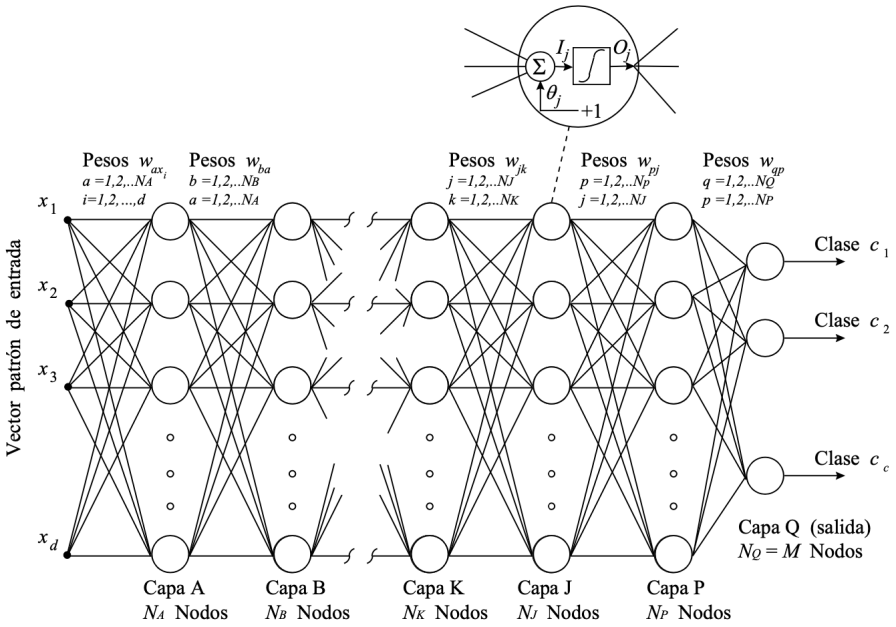

Figura 2.2: Arquitectura del modelo MLP. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

Los perceptrones ahora se agrupan en capas y cada tipo de capa tiene una función específica dentro de la red neuronal artificial:

- Capa de entrada: se encarga de recibir las características y el número de perceptrones depende del número de características que tenga el conjunto de datos.
- Capas ocultas: son las capas intermedias entre la entrada y la salida. Se encargan de aprender representaciones y características más abstractas y complejas a partir de los datos de entrada.
- Capa de salida: se encarga de devolvernos el resultado tras procesar la red neuronal toda la información. El número de perceptrones dependerá del número de clases que necesite el problema. Para un problema de clasificación binaria habría solo un perceptrón con la función de activación *sigmoid* o si fuera un problema de clasificación multiclase habría un perceptrón por clase.

El número de neuronas por capa indica el ancho de la red mientras que el número de capas de esta indica la profundidad (*depth*).

2.1.3. ¿Cómo aprende una red neuronal artificial?

Como se ha visto en el punto 2.1.1, el aprendizaje de las redes neuronales se traduce en la actualización de los pesos. A continuación, se explicará en profundidad cómo se ajustan los pesos para obtener unos resultados satisfactorios.

2.1.3.1. Optimización

La optimización desde el punto del AP es un proceso cuyo objetivo es encontrar los valores óptimos de los pesos que permitan minimizar la función de error (*error function*), coste (*cost function*) o pérdida (*loss function*). La función de pérdida mide las diferencias entre la salida de la red neuronal (predicción) y la etiqueta del objetivo verdadera (*ground-truth*). La función de pérdida permite cuantificar cuán bueno es un modelo prediciendo. A continuación, se explicarán algunos algoritmos de optimización.

Gradiente Descendente

El «descenso de gradiente» es uno de los métodos más utilizados para encontrar el punto mínimo de una función y el precursor de la optimización mediante el gradiente (*gradient-based optimization*).

La derivada de una función en un punto x es la pendiente o gradiente de la recta tangente a la función en dicho punto. Por el «Criterio de la Derivada Primera» obtenemos que:

- Si $f'(x) < 0$ entonces la función disminuye hacia la derecha.
- Si $f'(x) > 0$ entonces la función disminuye hacia la izquierda.
- Si $f'(x) = 0$ entonces x es un punto crítico o punto estacionario. Un punto crítico o estacionario puede ser un máximo, un mínimo o un punto de inflexión.

En la figura 2.3 se puede ver un ejemplo del cálculo del gradiente descendente mediante derivadas. La función utilizada es $f(x) = \frac{1}{2}x^2$ y su derivada es $f'(x) = x$.

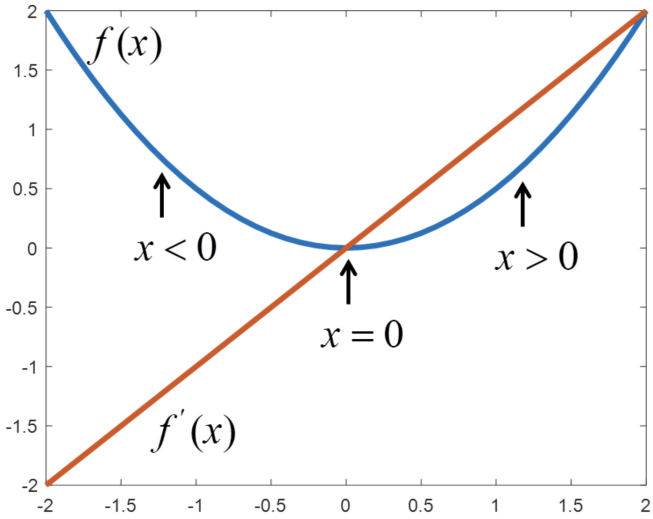

Figura 2.3: Gradiente descendente. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

En este caso obtenemos que $f'(x) = 0$ cuando x toma el valor 0 y es un mínimo local debido a que no es posible disminuir $f(x)$ mediante pasos infinitesimales. Si el mínimo local fuera el punto más bajo de todo $f(x)$ además sería mínimo global. El punto ideal sería el mínimo global, pero por lo general es necesario conformarse con un mínimo que tenga un comportamiento similar al mínimo global.

La mayoría de funciones a minimizar en el ámbito del aprendizaje automático tienen múltiples entradas y una única salida $f : \mathbb{R}^n \to \mathbb{R}$. Es por esto que para calcular el gradiente se necesita aplicar el concepto de derivada parcial $\frac{\partial}{\partial x_i} f(x)$ que mide cómo varía la función $f(x)$ según aumenta el valor de x_i en el punto x. Por lo tanto, el gradiente $\nabla_x f(x)$ es el vector que contiene todas las derivadas parciales y un punto crítico es un punto $(c_1, c_2, \ldots, c_n)$ donde cada derivada parcial de $f(x)$ con respecto a una variable $(x_1, x_2, \ldots, _x n)$ es 0.

El vector $\nabla_x f(x)$ se caracteriza por señalar la dirección en la que moverse para que la función f aumente. Esta característica es muy importante ya que realizando el movimiento opuesto en el sentido del descenso del gradiente $-\nabla_x f(x)$ podríamos acercarnos a un punto crítico y así minimizar la función objetivo. Los pasos del algoritmo son:

1. Inicialización los pesos w con valores aleatorios.
2. Repetir hasta alcanzar un mínimo aproximado.
   - 2.1 Establecer la tasa de aprendizaje ε con un valor pequeño fijo o utilizar la búsqueda lineal (*line search*) que consiste en evaluar $f(x - \varepsilon \nabla_x f(x))$ para múltiples valores de ε y seleccionar el que proporciona el valor de la función más pequeño.
   - 2.2 Cálculo del gradiente utilizando la ecuación 2.7.

$$\nabla_x J(w) = \left(\frac{\partial J(w)}{\partial x_1}, \frac{\partial J(w)}{\partial x_2}, \ldots, \frac{\partial J(w)}{\partial x_n} \right) \tag{2.7}$$

   - 2.3 Actualización de los pesos w utilizando la tasa de aprendizaje (*learning rate*) ε que permite controlar el tamaño de los pasos que se dan en cada iteración t. Para actualizar los pesos se utiliza la ecuación 2.10.

$$w(t+1) = w(t) - \varepsilon \nabla J(w(t)) \tag{2.8}$$

En algunos casos se puede evitar ejecutar el algoritmo resolviendo la ecuación $\nabla_x f(x) = 0$ donde se obtendría el punto crítico de la función f directamente.

Descenso del Gradiente Estocástico (SGD, *Stochastic Gradient Descent*)

La forma de la función objetivo a minimizar es la definida en la ecuación 2.9.

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} J_i(w) \tag{2.9}$$

Los pasos del algoritmo son:

1. Inicialización los pesos w con valores aleatorios.
2. Repetir hasta alcanzar un mínimo aproximado.
   2.1 Selección de una observación aleatoria del conjunto de datos de entrenamiento $J_i(w)$.
   2.2 Cálculo del gradiente de la observación escogida en el paso previo $\nabla J_i(w)$.
   2.3 Actualización de los pesos w utilizando la tasa de aprendizaje (*learning rate*) ε que permite controlar el tamaño de los pasos que se dan en cada iteración t. Para actualizar los pesos se utiliza la ecuación 2.10.

$$w(t+1) = w(t) - \varepsilon \nabla J_i(w(t)) \tag{2.10}$$

La mayor ventaja del optimizador es la eficiencia ya que solo calcula el gradiente sobre un elemento del conjunto de datos de entrenamiento $J_i(w)$ en vez de tener que calcular el gradiente sobre todo el conjunto como su predecesor *Gradient Descent*. Por otra parte, esto puede suponer un problema debido a que los valores de los pesos podrían fluctuar tras cada iteración dificultando la convergencia. La convergencia es el estado que alcanza un modelo cuando no mejora con entrenamiento adicional.

Descenso del Gradiente Estocástico con Momento (SGDM, *Stochastic Gradient Descent with Momentum*)

El SGDM es una variante del optimizador SGD que surge a raíz de la necesidad de que la razón de aprendizaje no fuera fija. Como la razón de aprendizaje era fija, suponía que con valores altos se tiende hacia la divergencia de los datos y con valores pequeños la convergencia era lenta. La solución que propusieron fue hacer que la tasa de aprendizaje disminuyera a medida que aprendía haciéndola dependiente del número de datos o iteraciones. Todo esto se traduce en que la actualización de los pesos se hará utilizando una combinación lineal del gradiente y la actualización previa como se puede ver en la ecuación 2.12.

$$\Delta w(t) = \alpha \Delta w(t-1) - \varepsilon \nabla J_i(w(t-1))$$

$$w(t) = w(t-1) + \Delta w(t-1) \tag{2.11}$$

$$w(t) = w(t-1) + \alpha \Delta w(t-1) - \varepsilon \nabla J_i(w(t-1)) \tag{2.12}$$

Siendo ε la tasa de aprendizaje, $\alpha \in [0, 1]$ el momento constante que controla la velocidad de actualización de $\Delta w(t-1)$.

Propagación de la Raíz Media Cuadrática (RMSprop, *Root Mean Square Propagation*)

El algoritmo de optimización RMSProp utiliza una tasa de aprendizaje diferente por parámetro al contrario que el algoritmo SGD y esta se puede adaptar a la función de pérdida a minimizar. Para ello mantiene una media móvil del cuadrado de los gradientes por parámetro que se calcula con la ecuación 2.13.

$$v(t) = \beta_2 v(t-1) + (1-\beta_2)(\nabla J_i(w(t-1)))^2 \tag{2.13}$$

Cada peso se actualiza individualmente realizando la división elemento a elemento como se puede ver en la figura 2.14. ε es una constante que evita la división por 0. Los pesos con gradientes grandes reducen su tasa de aprendizaje para evitar cambios drásticos en el proceso de optimización mientras que aquellos con gradientes pequeños la aumentan debido a que los cambios no afectarán significativamente a la función de pérdida.

$$w(t) = w(t-1) - \frac{\alpha \nabla J_i(w(t-1))}{\sqrt{v(t-1)} + \varepsilon} \tag{2.14}$$

Estimación del Momento Adaptativo (ADAM, *Adaptive Moment Estimation*)

El algoritmo de optimización ADAM es muy similar en cuanto a la actualización de los pesos al algoritmo RMSProp con momento como se puede ver en la ecuación 2.15.

$$\begin{aligned} m(t) &= \beta_1 m(t-1) + (1-\beta_1)\nabla J_i(w(t-1)) \\ v(t) &= \beta_2 v(t-1) + (1-\beta_2)(\nabla J_i(w(t-1)))^2 \\ w(t) &= w(t-1) - \frac{\alpha m(t-1)}{\sqrt{v(t-1)} + \varepsilon} \end{aligned} \tag{2.15}$$

Según Kingma y Ba (2015) ADAM, es "fácil de aplicar, eficiente desde el punto de vista computacional, tiene pocos requisitos de memoria, es invariable frente al reescalado diagonal de los gradientes y se adecúa bien a problemas grandes en cuestión de datos y/o parámetros".

Retropropagación de errores

La retropropagación o *backpropagation* es el principal procedimiento en el aprendizaje supervisado para entrenar redes neuronales con capas ocultas.

El primer paso del algoritmo es calcular el error en capa de salida, de forma que este error se propaga hacia atrás a través de las distintas neuronas y sus conexiones.

2.2. Redes neuronales convolucionales (RNC)

Las RNC, como se ha visto previamente, funcionan de forma similar a las redes neuronales de propagación hacia adelante. En este apartado veremos en profundidad la arquitectura de las RNC capa por capa.

2.2.1. Arquitectura de una red neuronal convolucional

Las RNCs tienen tres tipos de capas principales: capa convolucional, capa de agrupación y capa completamente conectada. A medida que se añaden capas a una RNC, aumenta su complejidad y le permite identificar una mayor superficie de las imágenes. Mientras

que las capas más superficiales (más cerca de la capa de entrada) se centran en identificar características simples como colores y bordes, las capas más profundas pueden reconocer objetos más complejos hasta identificarlos. A continuación, se explican las estructuras básicas de estos modelos.

A) Capa de entrada

La capa de entrada o *input layer* de una RNC es la primera capa de la red y su función es recibir los datos de entrada y transmitirlos a las capas contiguas. La capa de entrada no posee parámetros entrenables a diferencia de las otras capas. En el caso de las RNCs, los datos de entrada normalmente son imágenes. La capa de entrada permite definir el tamaño de los datos de entrada con el vector $[h, w, c]$ donde h y w son la altura y el ancho que deben tener las imágenes y c es el número de canales de las imágenes.

B) Capa convolucional

La capa convolucional o *convolutional layer* es la capa que caracteriza a las RNC y les da nombre.

La capa convolucional es la capa encargada de realizar la mayor parte de los cálculos. Dada una entrada x y un núcleo (*kernel*) de convolución k, obtendremos una salida s denominado mapa de características (*feature map*) como se puede observar en la ecuación 2.16.

$$s(t) = \int (x * w)(t) \tag{2.16}$$

A este proceso se le conoce como convolución que da lugar al nombre de la capa. La entrada x es un vector o matriz multidimensional y el núcleo k es, generalmente, un vector o matriz multidimensional cuyos parámetros se ajustan durante el aprendizaje. Dichas estructuras se conocen como tensores.

Las convoluciones se realizan sobre más de un eje a la vez como puede ser en el caso de las imágenes I donde necesitan ser tratadas con un núcleo bidimensional K como se puede ver en la ecuación 2.17.

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \tag{2.17}$$

En la figura 2.4 podemos ver un ejemplo de convolución con el núcleo K aplicado sobre un tensor bidimensional que podría ser una imagen I. El resultado de aplicar el núcleo de dimensión 3x3 a la imagen con dimensión 6x7 es una imagen de tamaño 4x5. Para mantener la dimensión inicial de la imagen, se podría aplicar la operación relleno con ceros o *zero-padding* para ampliar la imagen original con 2 filas (arriba y abajo) y dos columnas (izquierda y derecha). El desplazamiento o *stride* en el ejemplo es de una unidad, pero el valor podría ser más de una unidad si así se desea.

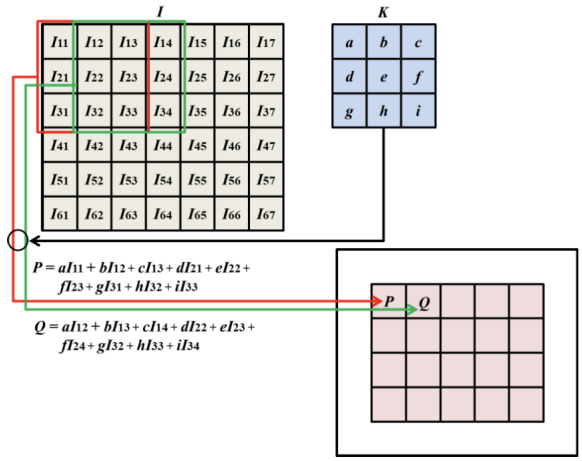

Figura 2.4: Convolución bidimensional. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

La dimensión o_j de una capa de convolución a lo largo del eje j tiene las siguientes propiedades: dimensión de entrada en el eje j, i_j; dimensión del núcleo en el eje j, k_j; desplazamiento del núcleo en el eje j, s_j; número de ceros concatenados al comienzo y al final del eje j, p_j.

En la figura 2.5 se puede ver el proceso de una convolución bidimensional (N = 2), con i1 = i2 = 5 correspondiente al tamaño de una imagen (I) de dimensión 5x5 y k1 = k2 = 3 correspondiente a un núcleo (K) de dimensión 3x3, con desplazamiento (*stride*) de una unidad (s1 = s2 = 1) y sin relleno de ceros (p1 = p2 = 0). Generalmente, el valor de las propiedades i, k, s y p es el mismo para todos los ejes.

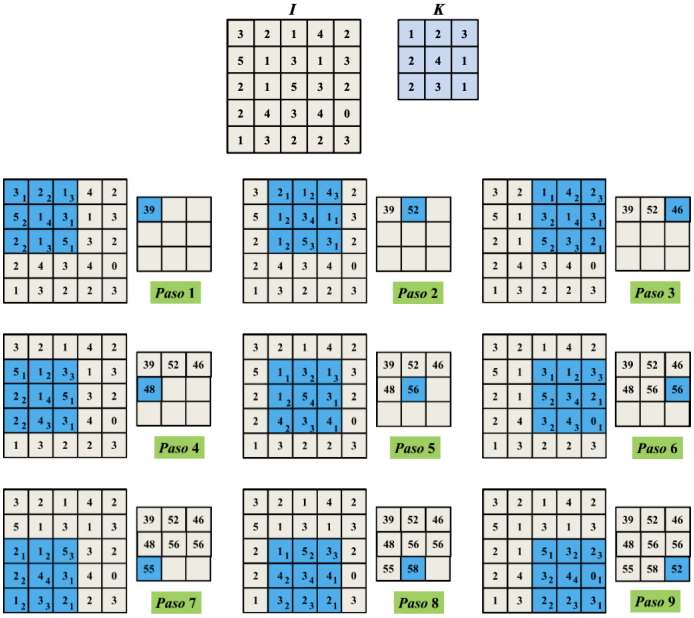

Figura 2.5: Ejemplo convolución bidimensional para N=2, i1=i2=5, k1=k2=3, s1=s2=1, p1=p2=0. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

Cada elemento del tensor resultante se obtiene realizando el producto de cada elemento del núcleo y el elemento de la entrada sobre el que se solapa y sumando los resultados.

C) Capa de agrupamiento

La capa de agrupamiento o *pooling* generalmente es usada para reducir la dimensionalidad haciendo que la posición de la ventana se actualice de acuerdo a los desplazamientos. Se suelen utilizar operaciones como el máximo (*max*), donde se divide la entrada en ventanas y cuya salida es el máximo de la ventana, o la media (*average*) de la ventana donde la salida es el resultado de aplicar esta operación sobre la ventana.

Algunos de los elementos de la ventana pueden quedar fuera de los elementos de entrada si la ventana se sitúa en las proximidades de los píxeles de borde. Para evitarlo, la entrada debe extenderse con valores de cero, lo que previamente se ha denominado *zero-padding*.

En la figura 2.6 se puede ver un ejemplo de agrupamiento máximo (*max pooling*) con relleno con ceros (S, p = 1 en horizontal) y un desplazamiento de dos unidades.

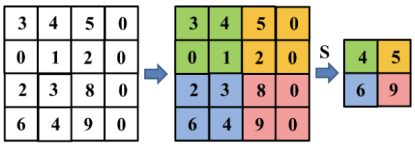

Figura 2.6: Ejemplo de agrupamiento máximo con relleno con ceros. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

El agrupamiento permite hacer la representación aproximadamente invariante a pequeñas traslaciones de la entrada, es decir, las salidas sobre las que se haya aplicado *pooling* no cambiarán, aunque se traslade la entrada con un pequeño desplazamiento. En la práctica resulta útil para aquellos casos donde se intente identificar si una característica está presente en vez de que la característica se encuentre en una localización espacial concreta.

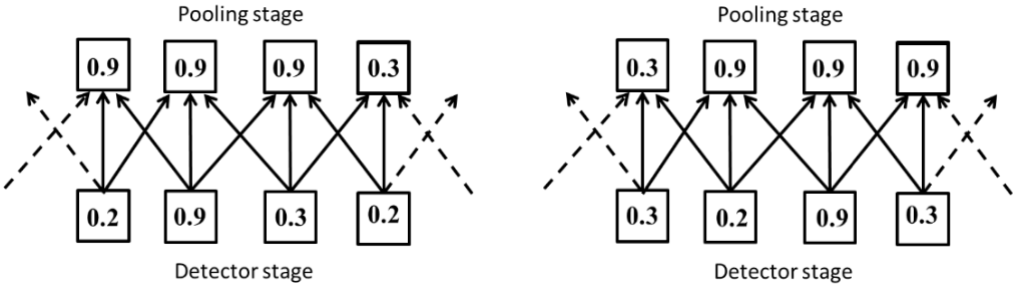

Figura 2.7: Invarianza del agrupamiento máximo. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

La subfigura de la derecha de la figura 2.7 ha sido desplazada un píxel respecto a la

subfigura de la izquierda. Como se puede observar, solo la mitad de los valores de salida de la capa de agrupamiento han cambiado. Esto se debe a que la operación agrupamiento máximo solo tiene en cuenta el máximo valor en la vecindad y no su localización exacta.

Cuando se realiza un agrupamiento sobre las salidas de convoluciones parametrizadas separadamente, las características pueden aprender qué transformaciones son invariantes a transformaciones de la entrada. En la figura 2.8 se puede observar que hay tres filtros aprendidos para detectar el número 3. Cuando en la entrada aparece un 3, independientemente de su orientación, la unidad detectora correspondiente se activa y la unidad de agrupamiento máximo obtiene un valor máximo de activación.

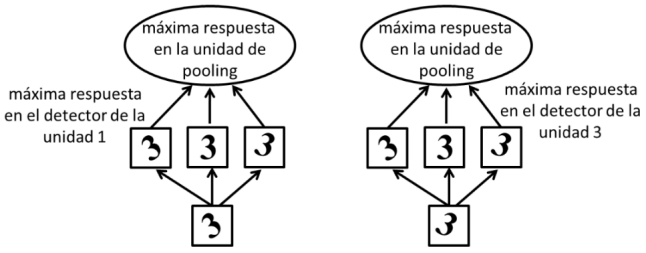

Figura 2.8: Ejemplo invarianza aprendida. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

D) Capa densa

La capa densa o completamente conectada (*full-connected layer*) es la encargada de clasificar la entrada en función de las características extraídas a través de sus capas anteriores y los núcleos utilizados. El nombre de la capa se describe así misma ya que cada nodo de la capa densa está conectada a todos los nodos de la capa anterior. Por otro lado, mientras que las capas convolucionales y de agrupación suelen utilizar la función ReLU, las capas densas utilizan la función *softmax*.

E) Capa de desconexión

La capa de desconexión o *dropout* permite desactivar un conjunto de neuronas en cada iteración del entrenamiento con el objetivo de reducir el sobreajuste u *overfitting*, es decir, evitar que modelo se adapte al conjunto de datos de entrenamiento y no sea capaz de generalizar a nuevos datos. En la figura 2.9 se puede observar un ejemplo de sobreajuste.

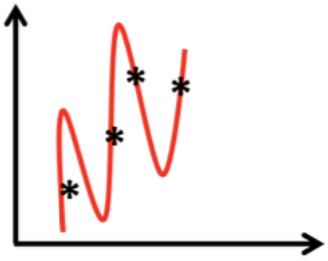

Figura 2.9: Ejemplo de sobreajuste. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

El conjunto de neuronas se puede desactivar de forma aleatoria o se le puede asignar un hiperparámetro p, probabilidad de supervivencia durante la fase de entrenamiento, que permita disminuir la importancia del peso durante la fase test. En la figura 2.10 se muestran nodos desactivados y nodos con el peso de activación atenuado por la probabilidad p.

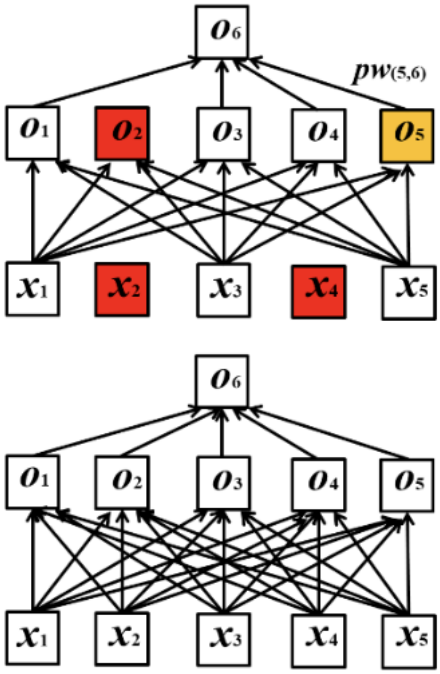

Figura 2.10: Dropout. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

El concepto *dropout* surge a raíz de que, en redes de aprendizaje profundas, al actualizar los pesos, alguna de las conexiones alcanza una capacidad predictiva superior a otras. Al actualizar en cada iteración solamente una fracción de las conexiones, permite un mejor ajuste de los pesos.

F) Capa de normalización

La distribución de las entradas en cada capa cambia durante el entrenamiento en función de los cambios de las capas anteriores. Esto produce que el entrenamiento se ralentice al requerir tasas de aprendizaje bajas y hace difícil entrenar modelos con saturaciones no lineales. Esto se conoce como desplazamiento covariable interno (*internal covariate shift*). Las capas de normalización solventan este problema mediante la normalización de las capas de entrada.

La normalización se realiza sobre cada mini-lote (*mini-batch*), haciendo posible utilizar tasas de aprendizaje altas e incrementando la necesidad de utilizar capas de desconexión. Ioffe y Szegedy (2015) proponen la siguiente estrategia para la normalización por lotes (batch normalization). En lugar de aplicar conjuntamente la normalización en las capas de entrada y salida, esta se aplica de forma independiente en cada mapa de características escalares. Para una capa con entrada d-dimensional, $x = x_1, \ldots, x_d$ se normaliza cada dimensión según la ecuación 2.18.

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{Var[x^{(k)}]} \tag{2.18}$$

La normalización puede cambiar lo que la capa representa, es decir, al normalizar las entradas de una capa que utiliza una función no lineal, como la sigmoide, se pueden restringir los valores a un régimen más lineal perdiendo parte de la capacidad no lineal. Para paliar este problema, la transformación aplicada debe representar la transformación identidad. Para ello, se introducen los parámetros $\gamma^{(k)}$ y $\beta^{(k)}$, como se puede ver en la ecuación 2.19, que escalan y aplican una traslación sobre el valor normalizado y, además, se ajustan durante el entrenamiento junto al resto de parámetros.

$$x^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta(k) \tag{2.19}$$

Debido a que es prácticamente imposible realizar la normalización sobre el conjunto total de datos de entrenamiento, se utilizan mini-lotes para el entrenamiento. Por cada mini-lote se calcula la media y la varianza. Considérese un mini-lote M de dimensión m y una activación particular x, $M = x_1, \ldots, x_m$. Sean los valores normalizados $\hat{x_1}, \ldots, \hat{x_m}$ y sus transformaciones lineales $y_1, \ldots, y_m$ y la transformación de normalización del mini-lote $MN_{\gamma,\beta} : x_1, \ldots, x_m \rightarrow y_1, \ldots, y_m$. El Algoritmo-1 MN tiene como entradas los valores x en un mini-lote, M, γ y β y como salida $y_i = MN_{\gamma,\beta}(x_i)$ donde:

$$\text{Media del mini-lote: } \mu_M = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\text{Varianza del mini-lote: } \sigma_M^2 = \frac{1}{m}\sum_{i=1}^{m} x_i - {\mu_M}^2$$

$$\text{Normalización: } \hat{x_i} = \frac{x_i - \mu_M}{\sqrt{\sigma_M^2 + \mathcal{E}}}$$

Escalado y traslación: $MN_{\gamma,\beta}(x_i) \equiv y_i = \gamma\hat{x}_i + \beta$

2.3. Detectores de objetos

En este apartado se introducen los detectores de objetos en su parte conceptual, a continuación, se introducen varios tipos de detectores de objetos existentes en la literatura y por último los utilizados en este trabajo, todos ellos tomando como referencia Pajares et al. (2021).

2.3.1. ¿Qué son los detectores de objetos?

La detección de objetos es un método o técnica de visión artificial que permite reconocer y localizar objetos en imágenes o videos mediante RNC. El objetivo es dotar a las máquinas de la capacidad de reconocer y localizar objetos de interés en imágenes de la misma forma que lo haría un humano. La base de la detección de objetos es la determinación de regiones de interés que se pasan a la RNC para su clasificación obteniendo la probabilidad de que la región contenga un objeto de una determinada clase y seleccionándose el de mayor probabilidad.

Los modelos de detección de objetos, según el esquema propuesto por Bochkovskiy et al. (2020), constan de una columna vertebral o red troncal y una cabeza. Para la columna vertebral se utilizan distintos tipos de detectores como pueden ser VGG, ResNet, ResNeXt o DenseNet para GPU y SqueezeNet, MobileNet o ShuffleNet para CPU (Pajares et al., 2021). Por otra parte, siguiendo esta misma referencia, en la cabeza se pueden utilizar detectores de un estado (predicción densa), como pueden ser YOLO, SDD, RetinaNet, OverFeet, así como CornetNet o FCOS que no utilizan *anchor boxes*, o de dos (predicción dispersa) como pueden ser R-CNN o sus variantes Fast, Faster o Mask así como R-FCN, Libra R-CNN o RepPoints siendo este último el único que no utiliza los que se conoce como *anchor boxes*. Algunos detectores incluyen capas intermedias para recopilar mapas de características denominadas cuello. En la figura 2.11 se muestra la estructura mencionada.

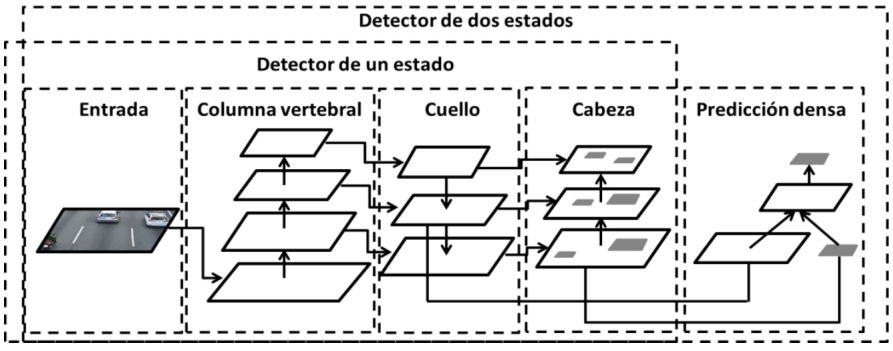

Figura 2.11: Esquema general de los detectores de objetos. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

Solapamiento de regiones y precisión

Para determinar la similitud entre dos rectángulos independientemente de la unidad de medida se define el coeficiente o IoU o índice de Jaccard (1908). El índice IoU propone que dos rectángulos son coincidentes cuando tienen un solapamiento de la unidad y no coincidentes cuando tengan un solapamiento nulo como se puede ver en la figura 2.12.

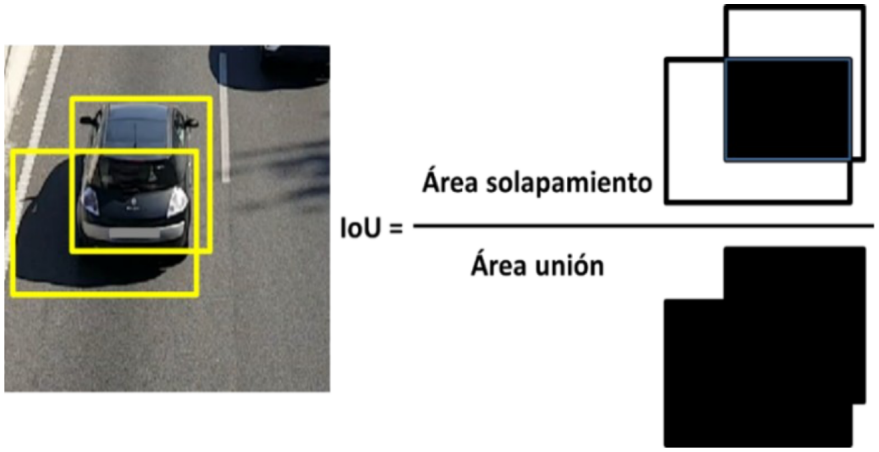

Figura 2.12: Índice IoU. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

Se define la puntuación de confianza (*confidence score*) como la probabilidad p_o de que un *anchor box* contenga un objeto. Una predicción puede considerarse:

- *Verdadero Positivo* (VP). Para que una predicción se considere VP se debe cumplir:
  a) p_0 es mayor que un determinado umbral.
  b) La clase predicha coincide con la clase especificada como *ground truth*.
  c) El IoU del *bounding box* predicho es mayor que un determinado umbral.
- *Verdadero Negativo* (VN). Una predicción se considera VN cuando el valor de p_0 de una detección que se espera que no detecte nada es menor que un determinado umbral.
- *Falso Positivo* (FP). Para que una predicción se considere FP se deben incumplir los criterios b y c de las predicciones VP.
- *Falso Negativo* (FN). Una predicción se considera FP cuando el valor de p_0 de una detección que se espera que sea un *ground truth* es menor que un determinado umbral.

La precisión P se define como el número de verdaderos positivos divido por la suma de verdaderos positivos y falsos positivos como se puede observar en la ecuación 2.20.

$$P = \frac{VP}{VP + FP} \tag{2.20}$$

La medida sensibilidad, exhaustividad o *Recall* (R) se define como el número de verdaderos positivos divido por la suma de verdaderos positivos y falsos negativos como se puede observar en la ecuación 2.21.

$$R = \frac{VP}{VP + FN} \tag{2.21}$$

Utilizando la precisión y el *recall* se puede calcular la curva PR (P y R representando los ejes de ordenadas y abscisas respectivamente) que permite evaluar la precisión de los detectores. Tanto la precisión como el *recall* pueden tomar valores en el rango [0,1]. Calculando el área bajo la curva PR se obtiene la métrica AP, pero al ser un cálculo computacionalmente costoso se realiza una interpolación como se puede ver en la ecuación 2.22.

$$AP = \int_0^1 precision(r)dr \rightarrow AP = \sum_{i=1}^{n-1}(r_{i+1} - r_i)p_int(r_{i+1}) \tag{2.22}$$

Siendo $r_i(r_1, r_2, \ldots, r_n)$ los valores de R en orden ascendente y p_int el valor máximo de precisión para cualquier valor en R definido en la ecuación 2.23.

$$p_int(r) = \max_{r' \geq r} p(r') \tag{2.23}$$

El cálculo del AP se realiza para una clase. En un problema de detección de objetos se pueden tener K clases, por lo que se define la media de la precisión promedio o mAP como la media de AP sobre todas las clases.

Otra métrica útil es *Average Recall* (AR) o *recall* promediado sobre todos los $IoU \ni [0{,}5, 1{,}0]$ y se define como dos veces el área bajo la curva *recall-IoU*,

$$AR = 2 \int_{0,5}^{1,0} recall(o)do$$

donde o es IoU y $recall(o)$ es el correspondiente $recall$.

Para detectores de objetos que pueden tener K clases, se define la media del *recall* promedio o *mean Average Recall* (mAR) como la media de AR sobre todas las clases.

Anchor boxes

Un *bounding box* es en definitiva un rectángulo definido por sus coordenadas, ya sea utilizando las coordenadas superiores izquierda y derecha o bien el centro más su ancho y alto. Los *bounding boxes* permiten determinar la categoría y la localización de los objetos dentro de una imagen. Por otra parte, los *anchor boxes* son rectángulos en la imagen

permiten determinar si las regiones contienen objetos de interés. En el caso de existir un objeto, se trata de ajustar los bordes de los *anchor boxes* hacia el rectángulo *ground truth* que define el objeto verdadero.

En definitiva, la diferencia entre los *anchor boxes* y los *bounding boxes* es que los *anchor boxes* definen una propuesta de región delimitada por un *bounding box* donde el objetivo de la red es determinar los parámetros que permitan ajustar el *anchor box* al *ground truth*.

La metodología propuesta por Zhang et al. (2020) para definir *anchor boxes* consiste en generar múltiples rectángulos con diferentes dimensiones y relaciones de aspectos centrados en cada pixel de la imagen. Dada una imagen con ancho y alto W y H respectivamente, se generan por cada pixel de la imagen *anchor boxes* con diferentes formas centradas en ese píxel (x_0, y_0). Una vez definido el centro, se define un *anchor box* de anchor (w) y alto (h) dados. La forma de los distintos rectángulos es más o menos factibles dependiendo del tipo de objetos a detectar.

Desde el punto de vista del entrenamiento de redes neuronales, se considera cada *anchor box* como un ejemplo de entrenamiento. Cada *anchor box* tiene asignada la categoría del objeto contenido en el *anchor box* y el desplazamiento (*offset*) del *ground truth* relativo al *anchor box*. El proceso consiste en generar *anchor boxes*, predecir las categorías y desplazamientos para cada *anchor box*, ajustar el *anchor box* para obtener los *bounding boxes* y finalmente filtrarlos para obtener los más prometedores.

Dados los *anchor boxes* $A_1, A_2, \ldots, A_n$ y los *ground truth bounding boxes* $B_1, B_2, \ldots, B_m$ siendo $m \leq n$. Se define la matriz X donde el elemento x_{ij} es el IoU del *anchor box* A_i al *ground truth bounding box* B_j. Primero se busca el elemento de mayor valor en la matriz X al que le corresponde una fila y columna (i_1, j_1) donde se asigna B_{j1} a A_{i1}. Tras esto, se descartan todos los elementos en la fila i_1 y columna j_1 en X. Este proceso se realiza hasta que se hayan descartados todos los elementos de la columna m de X, donde se habrá asignado un *ground truth bounding box* a cada *anchor box*. El segundo paso se trata de encontrar el *bounding box* B_j con el mayor valor IoU con *anchor box* A_i y asignar B_j a A_i siempre y cuando el IoU resultante sea mayor que un determinado umbral prefijado. En el ejemplo de la figura 2.13, se supone que máximo valor en X es x_{23} por lo que se asigna B_3 a A_2, descartando los elementos de la fila 2 y la columna 3 de la matriz X. Tras esto, el máximo valor en las celdas sombreadas es x_{51} por lo que se asigna B_1 a A_5, descartando los elementos de la fila 5 y la columna 1 de la matriz X. Seguidamente se selecciona el máximo valor en las celdas sombreadas x_{12} con asignación de B_2 a A_1. Al quedarse sin asignación A_3 y A_4, se busca el máximo valor de cada fila siempre y cuando superen un umbral determinado.

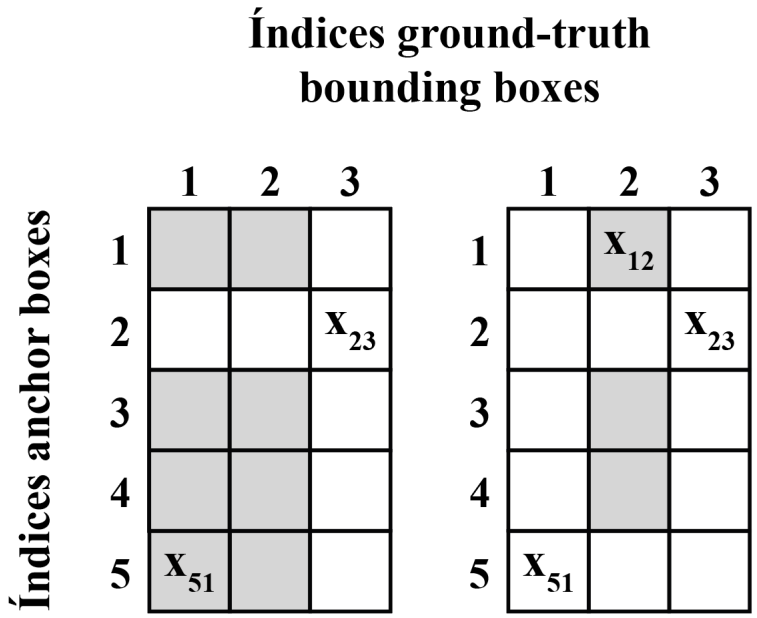

Figura 2.13: Asignación de *ground truth bounding boxes* a *anchor boxes*. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

La asociación de los *anchor boxes* a los *ground truth bounding box* se realiza a la vez que la asignación del desplazamiento. Sea un *anchor box* A con centro (p_x, p_y) con ancho p_w y alto p_h y un *ground truth bounding box* B con centro (g_x, g_y) con ancho g_w y alto g_h obtenemos que el desplazamiento se calcula utilizando la ecuación 2.24.

$$(\frac{g_x - p_x}{p_w}, \frac{g_y - p_y}{p_h}, \log \frac{g_w}{p_w}, \log \frac{g_h}{p_h}) \tag{2.24}$$

Si un *anchor box* no tiene asignado un *ground truth bounding box*, se le asigna directamente la categoría fondo (*background*).

En la figura 2.14 aparecen dos objetos, Peluche-1 y Peluche-2, además del fondo de la imagen. En la imagen se muestran dos *ground truth bounding boxes* B_1 y B_2 correspondientes al Peluche-1 y Peluche-2 y cinco *anchor boxes* A_1, A_2, A_3, A_4 y A_5. El valor de IoU es el mayor de todos para los pares A_5-B_2 y A_2-B_1, por lo que se le asigna a A_1 la etiqueta de Peluche-2 y a A_2 la etiqueta de Peluche-1. Después se exploran los *anchor boxes* no etiquetados y se obtiene que a los *anchor boxes* A_1 y A_4 se les asigna la categoría fondo por no superar el umbral determinado mientras que al *anchor box* A_3 se le asigna la categoría Peluche-2 por superar el umbral determinado.

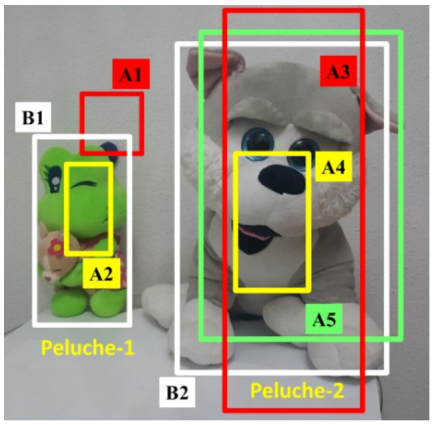

Figura 2.14: Índice IoU. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

Es posible que se realicen predicciones muy similares para el mismo objeto. Utilizando el método supresión no-máxima (NMS, *Non-Maximum Supression*) permite eliminar las predicciones muy similares de *bounding boxes*. El método NMS se basa en los siguientes pasos:

1. Se ordenan las predicciones por su confianza de mayor a menor excluyendo los clasificados como fondo generando una lista L.
2. Se selecciona el *bounding box* B_1 con el mayor nivel de confianza y se establece como referencia.
3. Se eliminan todos los *bounding boxes* predichos que no sea de referencia con un IoU con respecto a B_1 mayor que un determinado umbral.
4. Se realizan los pasos 2 y 3 con el resto de *bounding boxes* $(B_2, B_3, \ldots, B_m)$ predichos hasta que todos hayan sido evaluados.

Al terminar el proceso, el IoU entre dos elementos de la lista L será inferior al valor umbral preestablecido.

2.3.1.1. Modelos de detectores de dos etapas

Los modelos de dos etapas identifican las regiones candidatas que podrían contener un objeto en su primera fase y clasifican los objetos dentro de las regiones en su segunda fase. Los modelos de dos etapas proporcionan resultados más precisos, pero son más lentos que los modelos de una etapa.

Region Based CNN (R-CNN)

El modelo R-CNN es un modelo de dos etapas basado en el trabajo de Girshick et al. (2014), donde la primera etapa consiste en seleccionar un conjunto de regiones candidatas

a contener un objeto de una clase determinada. La selección se puede realizar mediante:

- Distinción de zonas en la imagen por diferenciación con el entorno circundante *objectness* (Alexe et al., 2012).
- Agrupaciones por color, textura, geometría o concentración de bordes (Endres y Hoiem, 2010).
- Agrupación de pequeñas subregiones o *regionlets* (Wang et al., 2015).
- Agrupaciones de bordes (*edge-boxes*) como en Zitnick y Dollár (2014).

Una vez seleccionadas las regiones, estas se redimensionan para ser suministradas a la red AlexNet (Krizhevsky et al., 2012) que demanda una dimensión de imagen de 227x227x3 píxeles. La RNC obtiene un vector de características (4096) para cada región, por ejemplo, la salida proporcionada por la red en la capa fc_6. Este vector de características pasa a un clasificador, que puede ser la propia red o por ejemplo una Máquina de Vectores Soporte (SVM, *Support Vector Machine*), (Cherkassky y Mulier, 1998; Vapnik, 2000), para que determine la clase del objeto. Los rectángulos de las regiones propuestas se utilizan para estimar los parámetros mediante regresión lineal utilizando las características de la capa *pool5* de AlexNet, de manera que el rectángulo del objeto que ha sido clasificado se reajusta. El modelo RNC realiza un procesamiento por cada región de interés (ROI, *Region of Interest*) al contrario que los modelos de una etapa que solo procesan la imagen una sola vez.

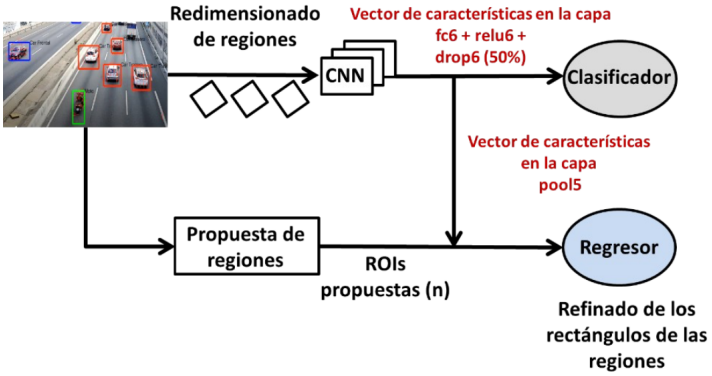

Figura 2.15: Arquitectura del detector R-CNN. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

El éxito del detector depende, según Pang et al. (2019), de tres aspectos:

- Las regiones seleccionadas deben ser representativas.
- Las características visuales extraídas se deben utilizar por completo.
- La función objetivo ha de ser óptima.

Como pueden existir ciertos desequilibrios en el proceso de entrenamiento debido a los tres aspectos anteriores, estos mismos autores proponen integrar tres componentes para paliar el problema:

- Muestreo equilibrado de IoU, donde se extraen muestras acordes a su IoU con el *ground truth* asignado.
- Pirámide de características balanceadas.
- Función de pérdida L_1 balanceada que promueve gradientes cruciales con el objetivo de reequilibrar la clasificación involucradas y la localización tanto general como precisa.

2.3.1.2. Modelos de detectores de una etapa

Los modelos de una etapa generan una predicción para cada región de la imagen utilizando *anchor boxes* y las predicciones se decodifican para generar las *bounding boxes* de los objetos. Los modelos de una etapa son mucho más rápidos que los de dos etapas, pero es posible que no alcancen el mismo nivel de precisión sobre todo cuando la imagen contiene objetos pequeños.

YOLO

El modelo YOLO es un modelo de una etapa donde la imagen completa se procesa una vez en vez de procesar las regiones por separado siendo su principal ventaja la velocidad de procesado. Aunque existen múltiples versiones de YOLO, en este apartado se profundizará en la versión base YOLOv1 (Redmon et al., 2016).

El esquema de YOLOv1 es simple. Una única red de tipo RNC predice los *bounding boxes* junto a sus probabilidades de pertenencia a una clase. El procedimiento consiste en dividir la imagen en una rejilla que contiene $S \times S$ celdas. Si el objeto cae sobre una de las celdas, esta será la responsable de la detección de ese objeto. Cada celda predice B *bounding boxes* y sus correspondientes valores de confianza que representan la seguridad del modelo de que el *bounding box* contenga un objeto. La confianza se define como $P(objeto)IoU^{verdadero}_{predicho}$. Si el objeto no existe en la celda, la confianza es cero y en caso contrario, lo que se desea es que el valor sea igual IoU entre el *bounding box* predicho y el *ground truth bounding box*. Cada *bounding box* consta de 5 predicciones: x, y, w, h y la confianza. Las coordenadas (x, y) son el centro del *bounding box* relativo a los límites de la celda y w y h son el ancho y el alto respectivamente. El ancho y el alto son predichos en relación a la imagen completa. Cada celda predice C probabilidades condicionadas de clase que contiene un objeto, $P(C_i|objeto)$. El cálculo de la confianza por clase para cada rectángulo se puede calcular utilizando la ecuación 2.25.

$$\text{Confianza clase} = P(C_i|objeto)P(objeto)IoU^{verdadero}_{predicho} = P(C_i)IoU^{verdadero}_{predicho} \tag{2.25}$$

El modelo plantea un problema de regresión donde la imagen se divide en una rejilla de dimensión $S \times S$ celdas, para cada celda se predicen B *bounding boxes* junto a sus correspondientes valores de confianza de clase y las C probabilidades condicionadas de pertenencia a una clase. Las predicciones resultantes se codifican como un *tensor* de dimensión $S \times S \times (5B + C)$.

Una cosa a destacar es que, aunque el detector predice varios *bounding boxes* por cada celda, solo se predice un único objeto por celda.

La arquitectura propuesta por Redmon et al. (2016) y tomada de Pajares et al. (2021) está inspirada en GoogleNet (Szegedy et al., 2014). El modelo consta de 24 capas convolucionales seguidas de 2 capas totalmente conectadas, los módulos *inception* se sustituyen por capas de reducción 1x1 (para reducir las dimensiones del espacio de características) seguidas por capas convoluciones de dimensión 3x3. La salida que produce es un *tensor* de dimensión 7x7x30. En la figura 2.16 se muestra la arquitectura de la completa.

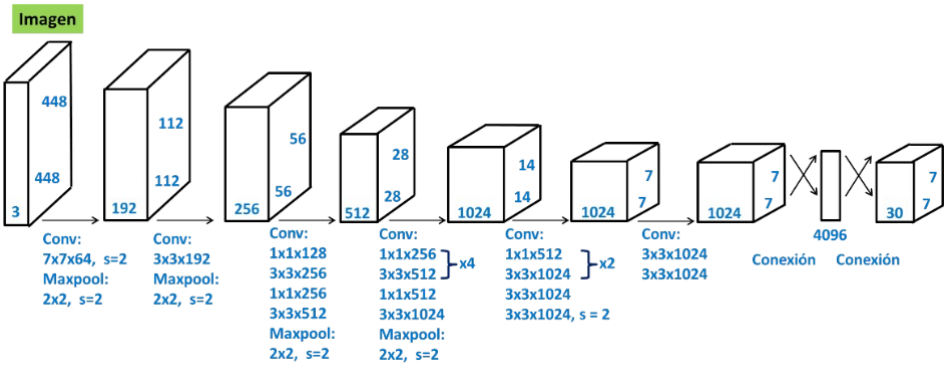

Figura 2.16: Arquitectura de la RNC del detector YOLO. Fuente: Aprendizaje Profundo (Pajares et al., 2021)

.

Para determinar la pérdida para el VP, lo ideal es que entre todos los *bounding boxes* predichos por cada celda solo sea uno de ellos el responsable de identificar el objeto. Es por esto por lo que se selecciona el de mayor IoU con el *ground truth*. YOLO utiliza la suma del error cuadrático entre las predicciones y el *ground truth* para calcular la pérdida. La función de pérdida está compuesta por:

a) Pérdida de clasificación. La función de pérdida en cuanto a la clasificación, definida en la función 2.26, se calcula como el error cuadrático de las probabilidades condicionadas por cada clase, donde $O_i^{obj} = 1$ si el objeto aparece en la celda i, $\hat{p}_i(c)$ es la probabilidad de clase condicionada para la clase c en la celda i y S^2 es el número de celdas de la rejilla en la que se divide la imagen original.

$$L_{clasification} = \sum_{i=0}^{S^2} O_i^{obj} \sum_{c \in clases} (p_i(c) - \hat{p}_i(c))^2 \tag{2.26}$$

b) Pérdida de localización. La función de pérdida en cuanto a la localización, definida en la función 2.27, mide los errores en la localización y dimensiones del *bounding box*

predicho para detectar un objeto, donde $O_{ij}^{obj} = 1$ si el j-ésismo *bounding box* en la celda i es el responsable de detectar el objeto y λ_{coord} incrementa el peso para la pérdida en las coordenadas del *bounding box*.

$$L_{\text{localization}} = \lambda_{\text{coord}} \left[\sum_{i=0}^{S^2} \sum_{j=0}^{B} O_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] + \sum_{i=0}^{S^2} \sum_{j=0}^{B} O_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \right] \tag{2.27}$$

c) Pérdida de confianza. La función de pérdida en cuanto a la confianza se mide utilizando la parte izquierda de la ecuación 2.28 si el *bounding box* contiene un objeto y la parte derecha en caso contrario, donde $\hat{C}_i$ es la confianza del *bounding box* j en la celda i, $O_{ij}^{obj} = 1$ si el j-ésimo *bounding box* en la celda i es responsable de detectar el objeto, O_{ij}^{noobj} es el complemento del elemento anterior y λ_{noobj} decrementa el peso de la pérdida si se detecta fondo.

$$L_{confidence} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} O_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} O_{ij}^{noobj} (C_i - \hat{C}_i)^2 \tag{2.28}$$

Se introducen factores de corrección, λ_{coord} y λ_{noobj}, para evitar que los errores de localización tengan el mismo peso que los errores de clasificación. Para impedir que el modelo converja demasiado rápido, se incrementa la pérdida relativa a la predicción de coordenadas del *bounding box* y a su vez se decrementa la pérdida de las predicciones de los *bounding boxes* que no contienen objetos fijando los valores $\lambda_{coord} = 5$ y $\lambda_{noobj} = 0{,}5$ por ejemplo.

Para que los errores absolutos de *bounding boxes* grandes frente a pequeños no tengan el mismo peso, se utilizan las raíces cuadradas de los anchos y altos. La función de pérdida total es la suma de las tres funciones de pérdida mencionadas anteriormente como se puede observar en la ecuación 2.29.

$$L = L_{clasification} + L_{localitation} + L_{confidence} \tag{2.29}$$

Cuando un objeto grande es detectado por múltiples celdas de la rejilla de la imagen, se aplica NMS para eliminar las detecciones muy similares entre sí.

2.4. Entrenamiento y validación

En este apartado se explican los conceptos básicos relacionados con el entrenamiento y validación de un modelo como son los hiperparámetros o el proceso de validación de un modelo.

2.4.1. Hiperparámetros

Un hiperparámetro es una variable de configuración externa que se utiliza para administrar el entrenamiento de modelos de AA. No existen reglas para seleccionar los hiperparámetros o los valores óptimos de estos, se necesita experimentar para encontrar los valores óptimos de los hiperparámetros. A este proceso se le denomina ajuste de hiperparámetros u optimización de hiperparámetros.

El proceso de ajuste de hiperparámetros puede ser manual o automático. El proceso manual a pesar de ser más lento y tedioso, permite entender mejor cómo afectan al modelos los cambios en los hiperparámetros. El proceso es iterativo y debe probar diferentes combinaciones de hiperparámetros y valores de estos. El objetivo es definir una variable de destino e intentar aumentar o reducir esta variable, por ejemplo, la precisión. Algunos de los hiperparámetros más comúnes son:

- Tasa de aprendizaje (*learning rate*). Como se ha explicado anteriormente, la tasa de aprendizaje es un hiperparámetro que controla la velocidad de aprendizaje del modelo. Para valores altos el modelo aprenderá rápido, pero tenderá a la divergencia en al ajuste. Para valores muy pequeños el modelo convergerá muy lento y podría caer en un mínimo local ocasionando que el proceso se detenga sin llegar a la convergencia. Para solventar estos problemas mencionados, se puede reducir la tasa de aprendizaje a medida que se avance en el entrenamiento o regular la tasa de aprendizaje dinámicamente.
- Nodos de la red neuronal. Número de nodos que tendrá cada capa oculta de la red.
- Capas de la red neuronal. Número de capas ocultas que tendrá la red.
- Tamaño del lote (*batch*). El tamaño del lote o *batch* es un hiperparámetro que permite definir el número de muestras que se deben procesar antes de actualizar los parámetros (pesos) de una red en el modelo. Sus valores deben ser siempre potencias de 2. Si el tamaño del lote posee más de una muestra y es menor que el conjunto de datos de entrenamiento, los lotes se denominan mini-lotes (*mini-batch*).
- Iteración (*iteration*). El número de iteraciones se define como el número de lotes a procesar en una época como se puede observar en la ecuación 2.30.

$$iterations = \frac{\text{NÚMERO TOTAL MUESTRAS}}{\text{TAMAÑO LOTE}} \tag{2.30}$$

- Época (*epoch*). El número de épocas indica el número de veces que el conjunto total de muestras es procesado por el modelo de red neuronal.

2.4.2. Data augmentation

Una de las técnicas para evitar que el modelo sufra sobreajuste es el aumento de datos o (*data augmentation*). La técnica se basa en que los modelos que utilizan conjuntos de datos de entrenamiento más grandes ven reducido el sobreajuste. Si no es posible recopilar más datos que permitan aumentar el conjunto de entrenamiento, se puede incrementar el

tamaño del conjunto de entrenamiento original artificialmente aplicando transformaciones, por ejemplo, rotando, reescalando, recortando o desplazando. Zhang et al. (2018) proponen la creación de muestras nuevas utilizando la interpolación lineal promediada,

$$x\,\% = \lambda x_i + (1-\lambda)x_j$$

$$y\,\% = \lambda y_i + (1-\lambda)y_j$$

donde x_i y x_j son los datos de entrada, y_i y y_j son las etiquetas codificadas de forma numérica y λ pertenece al rango [0,1].

2.4.3. Validación

La validación del modelo de red neuronal proporciona una estimación del rendimiento de este cuando procesa muestras nuevas, es decir, no procesadas con anterioridad. La validación de datos se puede hacer siguiendo dos estrategias:

- Validación durante el proceso de entrenamiento. Matlab permite indicar en las opciones de entrenamiento el conjunto de datos de entrenamiento y la frecuencia en validación en número de iteraciones. En este caso, durante el entrenamiento se calcula la precisión de validación y la pérdida de validación en el conjunto de datos de validación. La ventaja de esta estrategia es que proporciona una monitorización del entrenamiento y, además, permite detener el entrenamiento cuando la pérdida de validación deje de disminuir, es decir, cuando el modelo deje de aprender.
- Validación tras finalizar el proceso de entrenamiento. Tras entrenar el detector, Matlab permite evaluar la efectividad de este utilizando un conjunto de validación. La ventaja de este tipo de entrenamiento es que proporciona una evaluación más imparcial y realidad del rendimiento del modelo cuando se procesen muestras no vistas anteriormente.

2.4.4. Transfer Learning

En el aprendizaje profundo se puede utilizar la transferencia de aprendizaje o *transfer learning* para entrenar una red de forma más rápida y fácil que entrenar la propia red con los pesos inicializados a 0 o al azar. Las características aprendidas transferidas permiten que el entrenamiento necesite un conjunto de datos de entrenamiento menor.

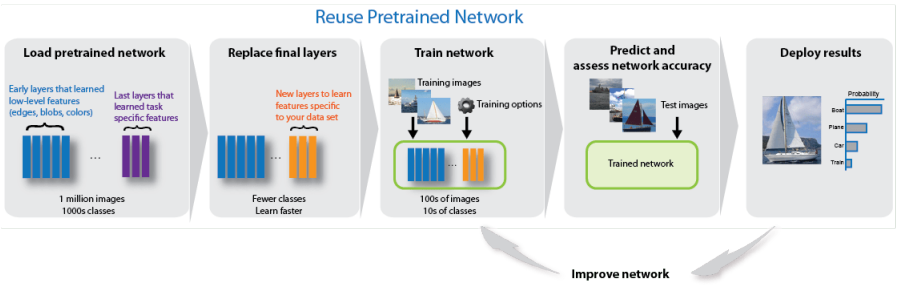

Figura 2.17: Esquema de la transferencia de aprendizaje. Fuente: MathWorks (2023j).

La transferencia de aprendizaje utiliza redes preentrenadas que han sido entrenadas con más de un millón de imágenes y pueden clasificarlas en 1000 objetos. Para poder utilizarlas, se reemplaza la última capa convolucional de la red y la última capa encargada de la clasificación. La tasa de aprendizaje en las capas transferidas se establece un valor pequeño para que aprendan más lentamente y no pierdan la capacidad de detectar las características que poseen mientras que en la capa convolucional modificada se establece un valor mayor para acelerar el aprendizaje. Por último, se realiza el entrenamiento del modelo de red neuronal con el conjunto de datos de entrenamiento nuevo y se realiza la validación para obtener el rendimiento del modelo detectando muestras no vistas anteriormente.

Capítulo 3

Diseño y desarrollo de la aplicación

A continuación, se describen en primer lugar los recursos utilizados, incluyendo los datos utilizados como las herramientas necesarias para el diseño y desarrollo. En segundo lugar, se aborda la implementación de los modelos desarrollados.

3.1. Recursos utilizados

3.1.1. Conjunto de datos

El conjunto de datos utilizado para el entrenamiento y la validación de los detectores, el cual se verá en profundidad a continuación, ha sido el *dataset* MS COCO (Microsoft, 2014).

El *dataset* MS COCO contiene 2.500.000 instancias etiquetadas en 328.000 imágenes con 91 tipos objetos distintos y 82 tienen más de 5.000 instancias etiquetadas como se puede observar en la figura 3.1. A diferencia del popular *dataset* ImageNet (Deng et al., 2009), MS COCO tiene menos categorías, pero más instancias por categoría como se puede observar en la figura 3.3. Esto último permite al modelo de red neuronal aprender modelos de objetos más detallados y tener una localización 2D más precisa. Además, MS COCO tiene un número de instancias etiquetas por imagen mayor que otros datasets que le puede ayudar a aprender información contextual. MS COCO posee más objetos por imagen (7,7) que ImageNet (3,0) y PASCAL (2,3) como se puede observar en la figura 3.2.

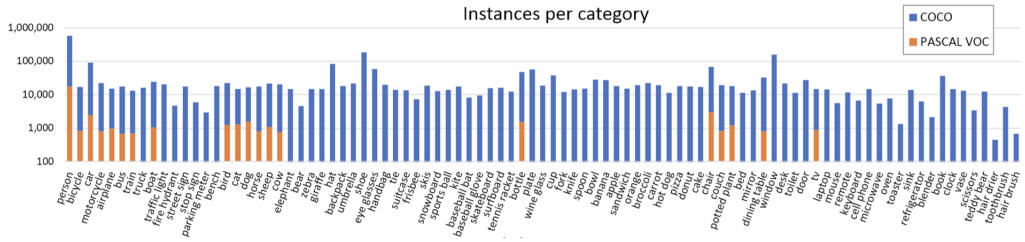

Figura 3.1: Número de instancias por categoría. Fuente: MS COCO (Lin et al., 2015)

.

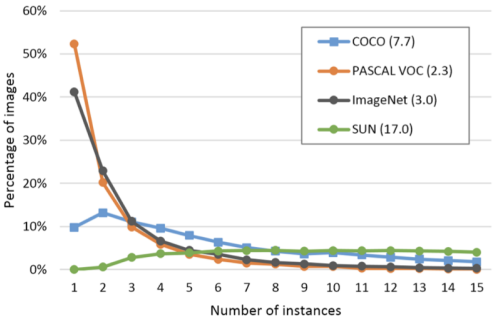

Figura 3.2: Número de instancias por imagen. Fuente: MS COCO (Lin et al., 2015)

.

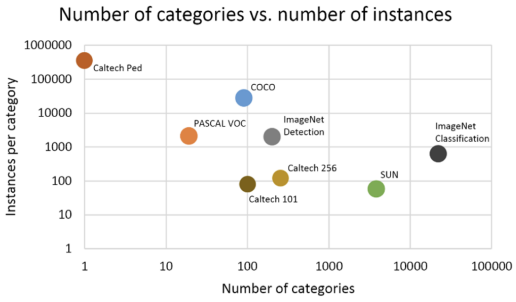

Figura 3.3: Número de categorías frente a número de instancias por categoría. Fuente: MS COCO (Lin et al., 2015)

.

Los objetos en las imágenes han sido etiquetados mediante segmentación por instancia. Esta segmentación por instancia permite una localización precisa ya que cada objeto se delimita por un *bounding box* y una máscara binaria que determina qué píxeles pertenecen al objeto localizado. El *dataset* MS COCO se puede utilizar para la clasificación de imágenes, para el etiquetado semántico de escenas y para la detección de objetos. Este trabajo se centrará en este último.

Para evitar que los entrenamientos de los modelos se puedan alargar en el tiempo dadas las limitaciones de recursos *hardware* propios, se ha optado por utilizar un conjunto de datos reducido del *dataset* MS COCO. Se han limitado tanto el número de categorías como el número de imágenes por categoría. No obstante, el número de imágenes por categoría debe ser el mismo para todas las categorías con el fin de evitar sesgos en el modelo, mejorar la generalización y facilitar la convergencia. Los conjuntos de de entrenamiento y de validación contendrán imágenes pertenecientes a las supercategorías «person» y «animal» (en este caso con 10 categorías) por lo que en total serán 11 categorías. Los conjuntos de entrenamiento, validación y tests poseen 528, 99 y 44 imágenes respectivamente.

Para ello, se ha implementado un programa en Python que permite obtener los conjuntos de entrenamiento y validación especificando las supercategorías y el número de imágenes por categoría. El programa carga el fichero «instances_train2017.json». El fichero posee la estructura que se muestra en la figura A.1. Dentro del contendido del fichero el campo más relevante es «annotations» que contiene información de las instancias etiquetadas por imagen. Las anotaciones para el problema de detección de objetos poseen la estructura que

se puede ver en la figura A.2.

- Identificador (*id*). Identificador único de la anotación.
- Identificador de la imagen (*image_id*). Identificador único de la imagen a la que pertenece la instancia definida en la anotación.
- Identificador de la categoría (*category_id*). Identificador de la categoría a la que pertenece la instancia definida en la anotación.
- Segmentación (*segmentation*). Máscara de segmentación de la instancia. El formato de la segmentación depende del campo *iscrowd*.
- Flag multitud (*iscrowd*). El flag *iscrowd* indica si la instancia representa un objeto único ($iscrowd = 0$, en cuyo caso se utilizan polígonos para delimitar la instancia) o múltiples objetos ($iscrowd = 1$, en cuyo caso se utiliza *Run-Length Encoding* (RLE)).
- Caja delimitadora (*bbox*). La caja delimitadora o *bounding box* delimita la posición de la instancia en la imagen mediante las coordenadas x, y, ancho w y alto h siendo la posición $x = 0$ e $y = 0$ la esquina superior izquierda de la imagen.
- Lista de categorías (*categories*). La lista de categorías relaciona los identificadores de las categorías con el nombre y la supercategoría de estas.

Para la manipulación del fichero «instances_train2017.json» se ha utilizado la librería «pycocotools». Se trata de una librería escrita en Python que permite trabajar con el *dataset* MS COCO facilitando la carga la visualización y el procesamiento de las imágenes, anotaciones y máscaras de segmentación. Se han utilizado las siguientes funcionalidades de la librería:

- Constructor de la clase COCO. La clase COCO es la que facilita la lectura y visualización de las anotaciones. Al constructor de la clase COCO se le pasa la ruta al fichero con extensión *JavaScript Object Notation* (JSON) que contiene las anotaciones para que las cargue en memoria.
- Recuperar categorías. La función «getCatIds()» recupera los identificadores de todas las categorías por defecto. La función permite filtrar las categorías por el nombre de la categoría o de la supercategoría. La función «loadCats()» recupera las categorías correspondientes a los identificadores pasados por parámetro.
- Recuperar imágenes. La función «getImgIds()» recupera todos los identificadores de las imágenes por defecto. La función permite filtrar los identificadores de las imágenes por uno o varios identificadores de imágenes y por uno o varios identificadores de categorías. Por otra parte, la función «loadImgs()» recupera las imágenes correspondientes a los identificadores pasados por parámetro.
- Recuperar anotaciones. La función «getAnnIds()» recupera todos los identificadores de las anotaciones por defecto. La función permite filtrar los identificadores de las anotaciones por uno o varios identificadores de imágenes, por uno o varios identificadores de categorías, por el rango de área o si la instancia representa uno o múltiples objetos. Por otra parte, la función «loadAnns()» recupera las anotaciones correspondientes a los identificadores pasados por parámetro.

Una vez cargado el fichero, se procede a generar los conjuntos de datos. Conviene reseñar que Matlab para poder entrenar los detectores necesita que el conjunto de datos sea una tabla que contenga la ruta de la imagen y las etiquetas ROI por categoría. Para ello, se ha utilizado el siguiente algoritmo:

1. Se establecen las supercategorías a procesar y el número de imágenes por categoría.
2. Se obtienen las categorías pertenecientes a las supercategorías que se han seleccionado para reducir el número de categorías.
3. Se selecciona una de las categorías obtenidas en el punto anterior.
4. Se recuperan los identificadores de todas las imágenes que tienen al menos una instancia clasificada con la categoría seleccionada en el punto anterior.
5. Se filtran los identificadores de las imágenes obtenidos por el número de imágenes por categoría establecido. A continuación, se recuperan las imágenes a partir de los identificadores filtrados.
   - 5.1 Se selecciona una de las imágenes obtenidas en el punto anterior.
   - 5.2 Se recuperan los identificadores de todas las anotaciones de la imagen seleccionada en el punto anterior.
   - 5.3 Se recuperan las anotaciones a partir de los identificadores recuperados en el punto anterior.
     - 5.3.1 Se selecciona una de las anotaciones obtenidas en el punto anterior. Se crea un diccionario vacío que contendrá las nuevas anotaciones a añadir.
     - 5.3.2 Si la categoría de la anotación no se encuentra dentro de las categorías aceptadas, se volverá al punto anterior. En caso contrario, si la categoría no ha sido previamente procesada, se genera una columna asociada a dicha categoría y se inicializa con un array que consta de N elementos, donde N es el producto del número de imágenes por categoría y el número de categorías aceptadas, y cada uno de estos elementos es un objeto vacío «{}».
     - 5.3.3 Si el diccionario con las nuevas anotaciones no contiene la categoría de la anotación, se añade al diccionario un nuevo par del tipo clave-valor donde la clave es el nombre de la categoría y el valor son las coordenadas del *bounding box* de la anotación. En caso contrario, se concatenan las coordenadas del *bounding box* y el valor de la clave existente con punto y coma.
     - 5.3.4 Por último, se añade al conjunto de datos la nueva imagen procesada y las anotaciones detectadas por categoría.
     - 5.3.5 Se repiten los pasos anteriores hasta terminar de procesar las anotaciones de la imagen completa.
   - 5.4 Se repiten los pasos anteriores hasta terminar de procesar las imágenes de la categoría correspondiente.
6. Se repiten los pasos 3, 4 y 5 hasta terminar de procesar las categorías aceptadas.

Finalmente, se genera un fichero con extensión CSV para cada uno de los conjuntos de datos entrenamiento y validación.

3.1.2. Visual Studio Code

VSCode es un editor de código fuente construido sobre el *framework Electron* y desarrollado por Microsoft (2023). Las características del editor son:

- Personalizable. VSCode permite instalar un gran número de extensiones a través de su tienda lo que permite agregar funcionalidades.
- Herramientas de edición de código. VSCode ofrece herramientas que aceleran el trabajo de los desarrolladores y mejoran la calidad del código como resaltado de sintaxis, autocompletado inteligente, navegación rápida entre ficheros, refactorización, formateo automático entre muchas otras.
- Control de versiones. VSCode integra a la perfección el sistema de control de versión Git. Ofrece una interfaz simple que permite interactuar con repositorios de GitHub.
- Depuración. VSCode posee herramientas de depuración que permiten ejecutar el código paso a paso para detectar posibles errores.
- Herramientas de desarrollo. VSCode integra herramientas de desarrollo tales como terminales o herramientas de gestión de paquetes entre otras muchas.
- Documentación y soporte. Dado que VSCode es muy popular entre la comunidad de desarrolladores, las contribuciones a las extensiones son continuas. Además, Microsoft proporciona una amplia documentación que permiten a los desarrolladores exprimir al máximo las capacidades del editor.

VSCode se ha utilizado en el desarrollo del programa que genera los conjuntos de datos de entrenamiento y validación y para la escritura de la memoria de este trabajo.

3.1.3. Python

Python (Rossum, 2023) es un lenguaje de programación ampliamente utilizado, que ofrece multitud de beneficios:

- Los programas escritos en Python, al tener una sintaxis cercana al lenguaje humano, son fáciles de entender por parte de los desarrolladores.
- Se necesitan menos líneas de código que en otros lenguajes de programación para generar desarrollos más grandes.
- Gracias a que es un lenguaje ampliamente utilizado, existen librerías que evitan que los desarrolladores no tengan que escribir desde cero.
- Python permite la interoperabilidad con otros lenguajes como Java o C.
- Existen numerosos recursos para aprender Python tales como libros, documentación *on-line* o videos tutoriales entre otros.

- Los programas escritos en Python se pueden trasladar entre sistemas operativos.

El lenguaje Python ha permitido desarrollar el programa que genera los conjuntos de datos de entrenamiento y validación. Gracias a las librerías «pycocotools» y «pandas», el programa ha podido ser desarrollado de forma rápida y de fácil interpretación. La versión del lenguaje Python que se ha utilizado es la 3.11.3.

3.1.4. Matlab

MATLAB es un entorno de programación, desarrollado por MathWorks (2023a) ampliamente utilizado por científicos para analizar datos, desarrollar algoritmos y crear modelos. Matlab destaca por:

- Lenguaje de programación. Además de ser un entorno de programación es un lenguaje de programación de alto nivel. El entorno facilita la escritura, ejecución y depuración del código.
- Herramientas y bibliotecas. Las *toolboxes* son desarrolladas y revisadas rigurosamente por profesionales y se encuentran totalmente documentadas.
- Visualización de datos y gráficos. Es conocido por su capacidad de visualización de datos mediante herramientas que crean gráficos y facilitan la comprensión de los datos.
- Eficiencia. Posee librerías y algoritmos optimizados que permiten realizar cálculos de manera eficiente.
- Interoperabilidad. Se integra fácilmente con otros lenguajes de programación como C, C++ y Python entre otros.
- Soporte. Como se ha mencionado anteriormente, Matlab se encuentra totalmente documentado. Además, la empresa MathWorks ofrece soporte técnico y actualizaciones regulares.

Las versiones de Matlab que se han utilizado para el entrenamiento y validación de los detectores de objetos son la R2022b y la R2023a. Además, ambas versiones han permitido explorar los conjuntos de datos para comprobar que estén correctamente construidos.

3.1.5. ClickUp

ClickUp (2023) es una aplicación de gestión de tareas que se utiliza para organizar proyectos. ClickUp ofrece una amplia cantidad de funcionalidades:

- Tareas. Permite crear tareas, asignarlas a personas, establecer fechas de inicio y fin y realizar un seguimiento del progreso entre otras cosas.

- Proyectos. Permite crear proyectos, establecer prioridades y relacionar las tareas entre sí.
- Incurrir. Permite iniciar un temporizador para saber cuánto tiempo llevas dedicado a la tarea y cuánto se ha desviado del tiempo estimado.
- Integración. Se integra con otras herramientas como Slack, Google Drive y GitHub entre otras.
- Estadísticas. Permite crear paneles que muestren métricas relacionadas con los proyectos, tareas y/o personas.

Indicar finalmente, que ClickUp ha sido utilizado para planificar la realización de la presente memoria.

3.1.6. Hardware

A continuación, se detalla el hardware utilizado para realizar el entrenamiento y las evaluaciones de los detectores de objetos desarrollados. El MacBook Pro con el chip Apple Silicon M1 Pro es el equipo que se ha utilizado para entrenar y evaluar los detectores de objetos. El procesador M1 Pro cuenta con una arquitectura de 5 nanómetros y está compuesto por una CPU (*Central Processing Unit*) de 8 núcleos con 6 núcleos de rendimiento y 2 de eficiencia, así como una GPU (*Graphic Processing Unit*) de 14 núcleos. Incorpora un procesador neuronal (*Neural Engine*) de 16 núcleos diseñado específicamente para acelerar tareas relacionadas con la IA. Además, dispone de una RAM (*Random Access Memory*) de 16GB de alta velocidad. El sistema operativo que se ha utilizado es macOS Ventura 13.5.

3.1.7. Librerías

Para el desarrollo del trabajo se han utilizado librerías de terceros que han facilitado tanto la selección de un conjunto de datos reducido como la implementación y evaluación de los detectores de objetos. Las librerías utilizadas son:

- Computer Vision Toolbox: biblioteca de Matlab que proporciona algoritmos, funciones y aplicaciones para diseñar y probar sistemas de visión por ordenador, visión 3D y procesamiento de vídeo. Permite realizar la detección y el seguimiento de objetos, así como la detección y extracción de características (MathWorks, 2023b).
- Deep Learning Toolbox: biblioteca de Matlab que proporciona un marco para diseñar e implementar redes neuronales profundas (MathWorks, 2023c).
- Image Processing Toolbox: biblioteca de Matlab que proporciona algoritmos y apps de flujo de trabajo para procesar, visualizar y analizar imágenes y desarrollar algoritmos (MathWorks, 2023d).
- Parallel Computing Toolbox: biblioteca de Matlab que permite paralelizar los cálculos realizados mediante procesadores multinúcleo, GPUs y clústeres de ordenadores (MathWorks, 2023e).

- Statistics and Machine Learning Toolbox: biblioteca de Matlab que funciones y apps para describir, analizar y modelar datos mediante el uso de la estadística descriptiva y gráficas (MathWorks, 2023f).
- Deep Learning Toolbox Model for AlexNet Network: biblioteca de Matlab que contiene un modelo con la arquitectura de RNC AlexNet preentrenado (MathWorks, 2023g).
- Deep Learning Toolbox Model for MobileNet-v2 Network: biblioteca de Matlab que contiene un modelo con la arquitectura de RNC MobileNet-v2 preentrenado (MathWorks, 2023h).
- Deep Learning Toolbox Model for ResNet-50 Network: biblioteca de Matlab que contiene un modelo con la arquitectura de RNC ResNet-50 preentrenado (MathWorks, 2023i).
- Shutil: biblioteca de Python que ofrece operaciones de alto nivel en archivos como la copia o la remoción (Documentation, 2023).
- Numpy: biblioteca de Python para realizar operaciones matemáticas eficientes mediante el uso de matrices multidimensionales conocidas como «arrays» (Numpy, 2023).
- Pandas: biblioteca de Python de código libre esencial en la limpieza, transformación y análisis de datos en el ámbito del análisis de datos y la ciencia de datos. Se caracteriza por utilizar estructuras de datos flexibles como el *DataFrame*, estructura bidimensional similar a una tabla, o la Serie que representa una columna de un DataFrame (Pandas, 2023).
- Pycocotools: biblioteca de Python que ofrece las funciones necesarias para interactuar con el *dataset* MS COCO (Girshick, 2018).

3.2. Implementación de los detectores de objetos

En este apartado se describe la implementación realizada para entrenar los detectores R-CNN y YOLO.

3.2.1. Carga de los conjuntos de datos

El primer paso antes de entrenar los detectores es la carga en memoria de los conjuntos de datos. Se ha implementado la función «loadCSV()» en Matlab de forma que, dada la ruta del documento CSV que contiene el conjunto de datos y el separador usado en el documento, devuelve la *ground truth table*. Esta función permite cargar los conjuntos de datos «train.csv», «val.csv» y «test.csv» para entrenamiento, validación y test respectivamente.

3.2.2. Implementación del detector de objetos R-CNN

Como se ha visto en el punto 2.3.1 en relación a R-CNN, estos detectores de objetos utilizan una RNC para clasificar las regiones propuestas de una imagen. Para evitar tener que entrenar una RNC desde cero, se utiliza la transferencia de aprendizaje a partir de un modelo preentrenado como se muestra en la figura A.3.

Una vez cargada la red preentrenada, se eliminan sus últimas tres capas. A continuación, se crea una nueva red con las capas transferidas, una capa totalmente conectada con tantos nodos como categorías haya en el conjunto de datos y la tasa de aprendizaje elevada para acelerar su aprendizaje, una capa *softmax* para convertir la salida de la capa anterior en probabilidades para cada clase y por último una capa de clasificación.

A continuación, se entrena el detector R-CNN utilizando la función «trainRCNNObjectDetector()». Los parámetros que recibe la función son el conjunto de datos «train», las capas de la nueva RNC y los opciones de entrenamiento como se puede ver en la figura A.4.

Entre las opciones de entrenamiento se encuentran el tamaño de mini-lote (*MiniBatchSize*), el número de épocas (*MaxEpochs*), la tasa de aprendizaje inicial (*InitialLearnRate*), los rangos de superposición negativo (*NegativeOverlapRange*) y positivo (*PositiveOverlapRange*) que indican el solapamiento que debe haber entre el *bounding box* predicho y el *ground truth bounding box* para considerar como error o acierto la predicción.

Por último, en la figura A.5 se puede ver cómo se evalúa el rendimiento del detector realizando predicciones utilizando la función «detect()» cuyos parámetros son el detector y la imagen a predecir. Una vez obtenidas las predicciones, se calculan las métricas AP, *recall* y precisión utilizando la función «evaluateDetectionPrecision()». El resultado que devuelve es por categoría, es decir, devuelve un array con la precisión de cada categoría, por ejemplo.

3.2.3. Implementación del detector de objetos YOLO

Al igual que el detector de objetos R-CNN, YOLO también utiliza una RNC. Sin embargo, en el caso de YOLO, y concretamente YOLOV4, que es la versión utilizada, se divide la imagen en una cuadrícula y la RNC clasifica cada una de las celdas de la cuadrícula realizando todo el proceso en una sola etapa. Para evitar tener que entrenar una RNC desde el principio, se utiliza la transferencia de aprendizaje.

Una vez cargada la red preentrenada, se eliminan sus últimas tres capas. A continuación, se crea una nueva red con las capas transferidas, una capa totalmente conectada con tantos nodos como categorías haya en el conjunto de datos y cuya tasa de aprendizaje sea elevada para acelerar su aprendizaje, una capa *softmax* para convertir la salida de la capa anterior en probabilidades para cada clase y por último una capa de clasificación que determine la clase del objeto detectado. Pasando a la función «yolov4ObjectDetector()» la red preentrenada, el número de clases a identificar, los *anchor boxes*, el tamaño de entrada y el nombre de las capas encargadas de la extracción de características se crea el detector de objetos YOLO. Para estimar los *anchor boxes* se ha utilizado la función

«estimateAnchorBoxes()» que, dado un conjunto de datos y el número de *anchor boxes*, devuelve los *anchor boxes* estimados y el IoU promedio que indica cuán bien se solapan *anchor boxes* estimados con los los *anchor boxes* del conjunto de datos.

A continuación, se entrena el detector de objetos YOLO utilizando la función «trainYOLOv4ObjectDetector()». Los parámetros que recibe la función son el conjunto de datos «train», la nueva RNC y las opciones de entrenamiento.

Al igual que el detector de objetos R-CNN, entre las opciones de entrenamiento se encuentran el tamaño de mini-lote (*MiniBatchSize*), el número de épocas (*MaxEpochs*) o la tasa de aprendizaje inicial (*InitialLearnRate*). Adicionalmente, también se encuentran como opciones de entrenamiento de YOLO los datos de validación (*ValidationData*) y la frecuencia de validación (*ValidationFrequency*) que permiten indicar si durante el entrenamiento se realizará la validación del modelo.

Por último, se evalúa el rendimiento del detector realizando predicciones en imágenes de test utilizando la función «detect()». Los parámetros que se le pasan a la función son el detector entrenado y el conjunto de datos de prueba.

Una vez obtenidas las predicciones, se calculan las métricas AP, *recall* y precisión utilizando la función «evaluateDetectionPrecision()». El resultado que devuelve es por categoría, es decir, devuelve un array con la precisión de cada categoría, por ejemplo.

Capítulo 4

Análisis de resultados

En este capítulo se analizan los resultados obtenidos durante el proceso de entrenamiento de los detectores de objetos. Además, se explica y determina el conjunto de hiperparámetros que permite lograr el mejor resultado con el objetivo de obtener un modelo con una mayor precisión. En todos los casos se especifican las distintas opciones de configuración de los modelos, así como los mencionados parámetros e hiperparámetros utilizados en cada caso.

4.1. Resultados

Como se ha comentado en el punto 3.1.1, el conjunto de datos que se ha utilizado es un conjunto reducido del *dataset* MS COCO y está formado por imágenes correspondientes a 11 clases. Como se ha mencionado en el punto 3.2, los detectores de objetos que se han seleccionado para su implementación y su análisis son R-CNN y YOLO. En la tabla 4.1 se pueden ver las arquitecturas de las RNC utilizadas en cada detector de objetos como columna vertebral de este y encargada de la clasificación de los objetos localizados en las imágenes.

Detector	**RNC**
R-CNN	AlexNet MobileNetV2 ResNet-50
YOLO	ResNet-50 MobileNetV2

Tabla 4.1: Arquitecturas de RNC por detector de objetos

La métrica que se ha utilizado para evaluar el rendimiento de ambos detectores de objetos ha sido el *Average Precision* (AP) debido a que es realmente efectiva en problemas de clasificación con clases desbalanceadas donde hay más muestras de unas clases que de otras. Debido a que se obtiene un AP por cada clase, se utiliza además la métrica mAP que

proporciona una medida global del rendimiento del modelo facilitando así su comparación con otros modelos.

4.1.1. R-CNN

Las arquitecturas de RNC que se han seleccionado para hacer de columna vertebral del detector de objetos R-CNN son AlexNet, MobileNetV2 y MobileNetV2. Para mejorar la eficiencia del entrenamiento, se hace uso de la técnica de transferencia de aprendizaje utilizando una versión preentrenada de estas RNCs. Esta versión preentrenada ha sido llevada a cabo con más de un millón de imágenes procedentes de la base de datos de ImageNet. Para evaluar el rendimiento del detector de objetos se utilizan imágenes nunca antes vistas. Cabe destacar que el detector de objetos R-CNN no permite la evaluación del conjunto de datos de validación durante el entrenamiento sino que tras finalizar este, se debe realizar un proceso aparte para esto. A continuación, se muestran los resultados obtenidos utilizando los hiperparámetros de la tabla 4.2 con cada una de las RNCs que constituyen la columna vertebral.

Hiperparámetro	**Valor**
Optimizer	SGDM
Momentum	0.9
Initial Learn Rate	0.001
Learn Rate Schedule	*piecewise*
Learn Rate Drop Factor	0.1
Learn Rate Drop Period	1
L2 Regularization	0.0001
Max Epochs	4
Mini Batch Size	128
Shuffle	*every-epoch*

Tabla 4.2: Hiperparámetros utilizados en el entrenamiento de los modelos del detector de objetos R-CNN

A) AlexNet

En la figura 4.1 se puede ver el progreso del entrenamiento. En la parte superior aparece la evolución de la precisión (*accuracy*), de forma que el entrenamiento será tanto mejor cuanto mayor sea ésta, teniendo en cuenta que el máximo valor posible es el 100%. Se observa en la figura cómo la precisión crece ligeramente, es decir evoluciona positivamente, hasta estabilizarse con valores oscilantes entre el 80-90%. Por otro lado, en la parte inferior de la figura 4.1 se representa la función de pérdida. Esta función, definida previamente, al tratarse de una medida del error resulta ser tanto mejor cuanto más se aproxime a cero. Se observa, cómo en las primeras iteraciones comienza con valores altos, para estabilizarse posteriormente a partir de la segunda época.

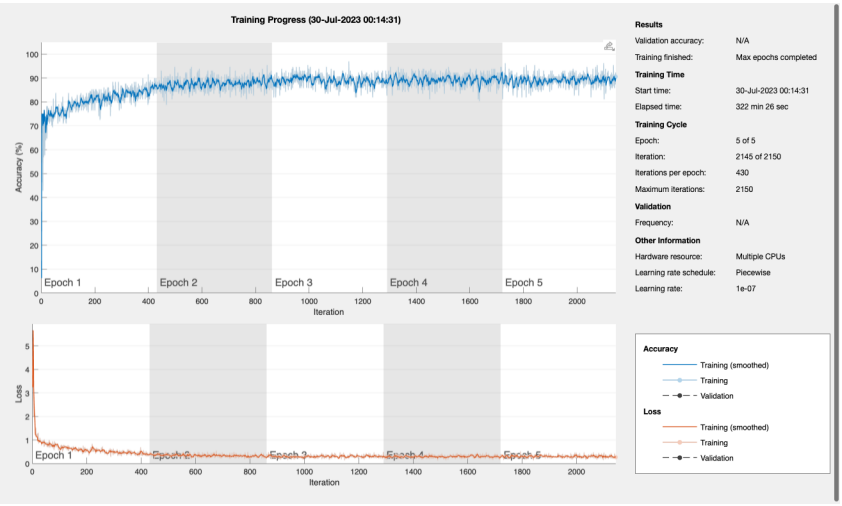

Figura 4.1: Entrenamiento del detector de objetos R-CNN utilizando la arquitectura AlexNet

Tras el entrenamiento se pueden extraer los resultados mostrados en la tabla 4.3, que expresan la bondad del proceso.

Época	**Iteración**	**Precisión (%)**	**Pérdida**	**Tasa de aprendizaje**
1	1	6,25	3,24	$1{,}0e^{-3}$
2	430	85,16	0,44	$1{,}0e^{-4}$
3	860	85,94	0,30	$1{,}0e^{-5}$
4	1290	82,81	0,48	$1{,}0e^{-6}$
5	2145	90,62	0,26	$1{,}0e^{-7}$

Tabla 4.3: Métricas obtenidas durante el entrenamiento del detector de objetos R-CNN utilizando la arquitectura AlexNet

En la tabla 4.4 se pueden ver los resultados obtenidos a tras evaluar el rendimiento del detector de objetos utilizando imágenes nunca vistas antes.

Clase	**AP** (%)
Person	3,54
Bird	3,32
Cat	3,93
Dog	1,11
Horse	29,11
Sheep	18,35
Cow	0,00
Elephant	0,00
Bear	0,00
Zebra	5,29
Giraffe	0,00

Tabla 4.4: Precisión media obtenida por clase durante la evaluación del detector de objetos R-CNN utilizando la arquitectura AlexNet

Los resultados muestran el AP obtenido por cada clase que oscila entre 0-100%, donde un valor más alto indica una mejor precisión en la detección de esa clase. Como se puede observar, el rendimiento del modelo es relativamente bajo ya que los valores de AP para la mayoría de clases son bajos. El modelo es más efectivo detectando las clases «Horse» y «Sheep» en comparación con el resto de las clases; tiene dificultades para detectar las clases «Person», «Bird», «Cat», «Dog» y «Zebra» y es incapaz de detectar las clases «Cow», «Elephant», «Bear» y «Giraffe». Por otro lado, se obtiene que el mAP resultante es de 5,88% confirmando el bajo rendimiento del modelo. Este puede atribuirse a diversas razones como el tamaño limitado del conjunto de datos de entrenamiento, la posibilidad de que la RNC con la arquitectura AlexNet no sea capaz de extraer las características necesarias para una detección precisa en todas las clases, la dificultad del modelo para generalizar (*overfitting*) o a la incorrecta estimación de los *bounding boxes*, entre otras razones.

B) MobileNetV2

En la figura 4.2 se puede ver el progreso del entrenamiento. Al igual que el modelo anterior que utiliza la arquitectura AlexNet, también se observa en la figura cómo la precisión crece ligeramente hasta estabilizarse con valores oscilantes entre el 80-90% durante el inicio de la primera época. Por otro lado, en la parte inferior de la figura 4.2 se observa cómo en las primeras iteraciones la función de pérdida comienza con valores altos, para estabilizarse posteriormente a partir de la segunda época.

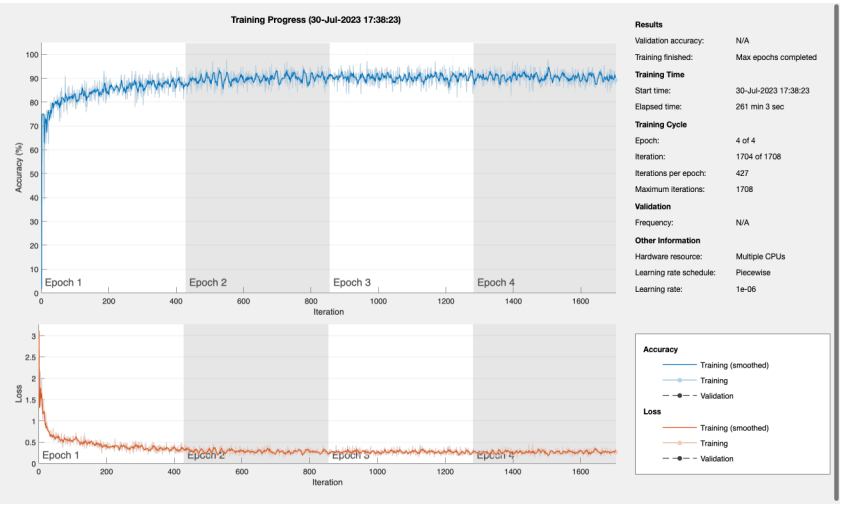

Figura 4.2: Entrenamiento del detector de objetos R-CNN utilizando la arquitectura MobileNetV2

Tras el entrenamiento se pueden extraer los resultados mostrados en la tabla 4.5, que expresan la bondad del proceso.

Época	**Iteración**	**Precisión (%)**	**Pérdida**	**Tasa de aprendizaje**
1	1	1,56	3,12	$1{,}0e^{-3}$
2	430	88,28	0,30	$1{,}0e^{-4}$
3	860	91,41	0,25	$1{,}0e^{-5}$
4	1704	89,06	0,26	$1{,}0e^{-6}$

Tabla 4.5: Métricas obtenidas durante el entrenamiento del detector de objetos R-CNN utilizando la arquitectura MobileNetV2

En la tabla 4.6 se pueden ver los resultados obtenidos a tras evaluar el rendimiento del detector de objetos utilizando imágenes nunca vistas antes.

Clase	**AP (%)**
Person	7,50
Bird	6,75
Cat	4,30
Dog	8,58
Horse	21,27
Sheep	10,53
Cow	0,00
Elephant	6,25
Bear	0,00
Zebra	8,37
Giraffe	18,59

Tabla 4.6: Precisión media obtenida por clase durante la evaluación del detector de objetos R-CNN utilizando la arquitectura MobileNetV2

Como se puede observar, el rendimiento del modelo es bajo ya que los valores de AP para la mayoría de clases son también bajos. El modelo es más efectivo detectando las clases «Horse» y «Giraffe» en comparación con el resto de las clases; tiene dificultades para detectar las clases «Person», «Bird», «Cat», «Dog», «Zebra», «Elephant» y «Sheep» y es incapaz de detectar las clases «Cow» y «Bear». El nuevo modelo puede detectar dos clases adicionales en comparación con el modelo anterior, lo que sugiere un rendimiento ligeramente superior. Esto se refleja en el mAP de 8,37%, el cual también muestra una ligera mejora. Esta mejora se debe a que la arquitectura MobileNetV2 es más profunda y especializada en comparación con la arquitectura AlexNet. Por otro lado, y como en el caso anterior, el bajo rendimiento puede atribuirse a diversas razones como el tamaño limitado del conjunto de datos de entrenamiento, la posibilidad de que la RNC con la arquitectura MobileNetV2 no sea capaz de extraer las características necesarias para una detección precisa en todas las clases, la dificultad del modelo para generalizar (*overfitting*) o a la incorrecta estimación de los *bounding boxes*, entre otras razones.

C) ResNet-50

En la figura 4.3 se puede ver el progreso del entrenamiento. Al igual que en los dos modelos anteriores, la precisión crece ligeramente hasta estabilizarse con valores oscilantes entre el 80-90% durante el inicio de la primera época. Por otro lado, en la parte inferior de la figura 4.3 se observa cómo en las primeras iteraciones la función de pérdida comienza con valores altos, para estabilizarse posteriormente a partir de la segunda época.

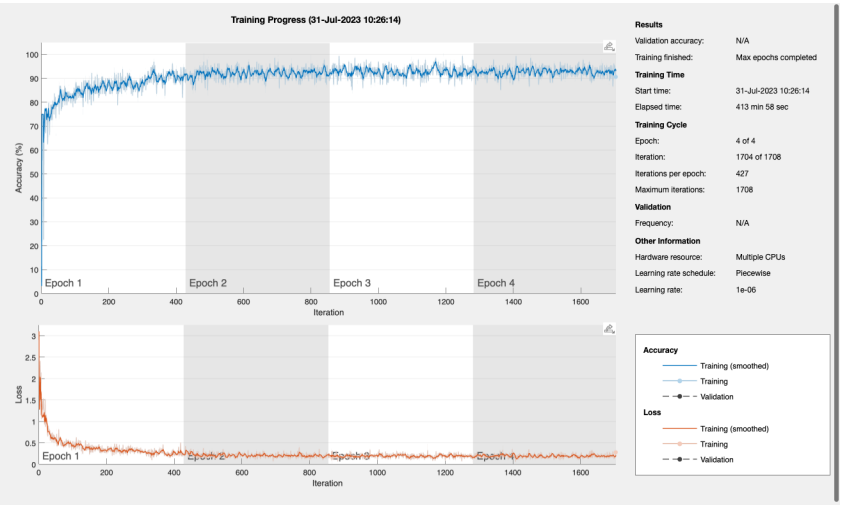

Figura 4.3: Entrenamiento del detector de objetos R-CNN utilizando la arquitectura ResNet-50

Tras el entrenamiento se pueden extraer los siguientes resultados mostrados en la tabla 4.7, que expresan la bondad del proceso.

Época	**Iteración**	**Precisión (%)**	**Pérdida**	**Tasa de aprendizaje**
1	1	3,12	3,10	$1{,}0e^{-3}$
2	430	89,06	0,27	$1{,}0e^{-4}$
3	860	91,41	0,21	$1{,}0e^{-5}$
4	1704	90,62	0,28	$1{,}0e^{-6}$

Tabla 4.7: Métricas obtenidas durante el entrenamiento del detector de objetos R-CNN utilizando la arquitectura ResNet-50

En la tabla 4.8 se pueden ver los resultados obtenidos a tras evaluar el rendimiento del detector de objetos utilizando imágenes nunca vistas antes.

Clase	**AP (%)**
Person	9,40
Bird	3,97
Cat	2,27
Dog	9,27
Horse	16,08
Sheep	17,54
Cow	1,15
Elephant	2,08
Bear	0,00
Zebra	33,78
Giraffe	8,97

Tabla 4.8: Precisión media obtenida por clase durante la evaluación del detector de objetos R-CNN utilizando la arquitectura ResNet-50

Como se puede observar, el rendimiento del modelo vuelve a ser bajo ya que los valores de AP para la mayoría de clases son igualmente bajos. El modelo es más efectivo detectando las clases «Zebra» y «Sheep» en comparación con el resto de las clases; tiene dificultades para detectar las clases «Person», «Bird», «Cat», «Dog», «Zebra», «Elephant», «Sheep» y «Cow» y es incapaz de detectar la clase «Bear». El nuevo modelo puede detectar una clase adicional en comparación con el modelo anterior, lo que sugiere un rendimiento ligeramente superior. Esto se refleja en el mAP de 9,50%, el cual también muestra una ligera mejora. Esta mejora se debe a que la arquitectura ResNet-50 tiene más capas (profundidad) que le permiten capturar características más complejas y abstractas. Por otro lado, el bajo rendimiento puede atribuirse, como en los casos anteriores, a diversas razones como el tamaño limitado del conjunto de datos de entrenamiento, la dificultad del modelo para generalizar (*overfitting*) o a la incorrecta estimación de los *bounding boxes*, entre otras razones.

Tras obtener el rendimiento de cada modelo, se puede observar que entrenar y evaluar un detector de objetos R-CNN es computacionalmente muy costoso y lento al tener que pasar cada región propuesta por la RNC para que se extraiga las características y se clasifique. También se puede inferir que todos los modelos poseen un bajo rendimiento siendo ligeramente superior el modelo que utiliza la arquitectura de RNC ResNet-50 ya que es el modelo con mayor mAP y ha logrado detectar todas las clases menos una, la

clase «Bear». A continuación, se muestran los resultados obtenidos al intentar mejorar el rendimiento del este modelo modificando los hiperparámetros optimizador y tamaño del lote.

Optimizador	**mAP** (%)
SGDM	9,50
RMSProp	5,17
ADAM	4,87

Tabla 4.9: mAP por optimizador del detector de objetos R-CNN utilizando la arquitectura ResNet-50

Como se puede ver en la tabla 4.9 el optimizador que obtiene mejor rendimiento es SGDM, ya que, funciona bien en problemas con conjuntos de datos pequeños.

Tamaño del lote	**mAP** (%)
128	9,50
256	7,40

Tabla 4.10: mAP por tamaño del lote del detector de objetos R-CNN utilizando la arquitectura ResNet-50

A pesar de que un mayor tamaño de lote puede proporcionar un gradiente más estable e incluso acelerar la convergencia en algunos casos, en problemas con un conjunto de datos pequeño puede producir *overfitting* provocando que el modelo no generalice correctamente. En este caso, en la tabla 4.10 se observa que al aumentar el tamaño del lote el rendimiento del modelo ha empeorado.

Finalmente se concluye, en términos generales, que el rendimiento del modelo no se ha logrado mejorar como se esperaba. Se podría haber suavizado el problema de *overfitting* aplicando técnicas de *data augmentation* o aumentando el conjunto de datos de entrenamiento, pero al no disponer de recursos computacionales suficientes ha sido imposible llevarlo a cabo. Por otro lado, el utilizar un conjunto de datos de entrenamiento provoca que el detector de objetos realice una estimación de los *bounding boxes* poco precisa ocasionando que el modelo no obtenga un buen rendimiento.

4.1.2. YOLO

Las arquitecturas de RNC que se han seleccionado para construir de columna vertebral del detector de objetos YOLO son ResNet-50 y MobileNetV2. Para mejorar la eficiencia del entrenamiento, se hace uso de la técnica de transferencia de aprendizaje utilizando una versión preentrenada de estas RNCs. Esta versión preentrenada ha sido entrenada más de un millón de imágenes de la base de datos de ImageNet. Para evaluar el rendimiento del detector de objetos se utilizan imágenes nuevas, es decir nunca antes utilizadas en el entrenamiento y validación. El detector de objetos YOLO sí permite la evaluación del conjunto de datos de validación durante el entrenamiento. A continuación, se muestran los

resultados obtenidos utilizando los hiperparámetros de la tabla 4.11 con cada una de las RNCs que constituyen la columna vertebral.

Hiperparámetro	**Valor**
Optimizer	SGDM
Momentum	0.9
Initial Learn Rate	0.00001
Learn Rate Schedule	*piecewise*
Learn Rate Drop Factor	0.1
Learn Rate Drop Period	10
L2 Regularization	0.0001
Max Epochs	40
Mini Batch Size	4
Shuffle	*every-epoch*
Validation Frequency	300
Batch Normalization Statistics	*moving*
Reset Input Normalization	*false*

Tabla 4.11: Hiperparámetros utilizados en el entrenamiento de los modelos del detector de objetos YOLO

A) ResNet-50

La figura 4.4 representa la evolución de la función de pérdida durante el entrenamiento. Esta función, definida previamente, al tratarse de una medida del error resulta ser tanto mejor cuanto más se aproxime a cero, tal y como se ha indicado previamente. Se observa, cómo en las primeras iteraciones comienza con valores altos, para estabilizarse posteriormente a partir de la quinta época. La línea azul representa la función de pérdida del entrenamiento mientras que la línea roja representa la función de pérdida de la validación.

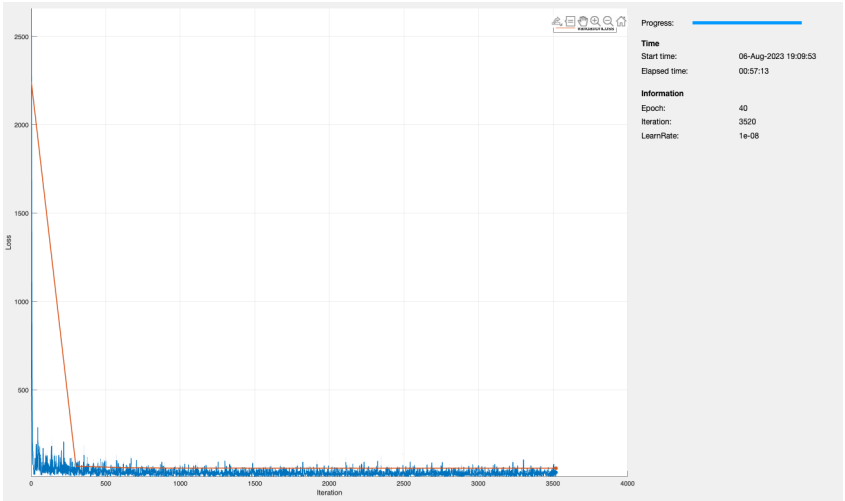

Figura 4.4: Entrenamiento del detector de objetos YOLO utilizando la arquitectura ResNet-50

Tras el entrenamiento se pueden extraer los siguientes resultados mostrados en la tabla 4.12, que expresan la bondad del proceso.

Época	**Iteración**	**Pérdida de validación**	**Tasa de aprendizaje**
4	300	67,62	$1{,}0e^{-5}$
14	1200	56,59	$1{,}0e^{-6}$
24	2100	56,21	$1{,}0e^{-7}$
35	3000	56,20	$1{,}0e^{-8}$
38	3300	56,19	$1{,}0e^{-8}$

Tabla 4.12: Métricas obtenidas durante el entrenamiento del detector de objetos YOLO utilizando la arquitectura ResNet-50

En la tabla 4.13 se pueden ver los resultados obtenidos a tras evaluar el rendimiento del detector de objetos utilizando imágenes nuevas, que no se han utilizado previamente ni en entrenamiento ni en validación.

Clase	**AP** (%)
Person	0,00
Bird	0,00
Cat	0,00
Dog	0,00
Horse	0,00
Sheep	0,00
Cow	0,00
Elephant	0,00
Bear	0,00
Zebra	0,00
Giraffe	0,00

Tabla 4.13: Precisión media obtenida por clase durante la evaluación del detector de objetos YOLO utilizando la arquitectura ResNet-50

Como se puede observar, el rendimiento del modelo es nulo ya que los valores de AP para todas las clases es cero. El modelo no es capaz de detectar ninguna de las clases. Esto puede atribuirse a diversas razones tales como el tamaño limitado del conjunto de datos de entrenamiento, la dificultad del modelo para generalizar (*overfitting*) o a la incorrecta estimación de los *bounding boxes*, entre otras razones. La correcta estimación de los *bounding boxes* toma especial relevancia en YOLO debido a que el detector no los estima automáticamente teniendo en cuenta el conjunto de datos de entrenamiento, sino que es un hiperparámetro que se puede modificar para mejorar el rendimiento de los modelos. Uno de los problemas que surge al aumentar el número de *bounding boxes* a utilizar es que aumenta también la complejidad del cómputo y por ende el tiempo de entrenamiento. Dados los resultados obtenidos y los problemas derivados de la naturaleza del detector, no se han podido evaluar los distintos modelos de YOLO disponibles y anteriormente mencionados.

Capítulo 5

Conclusiones y Trabajo Futuro

El trabajo ha alcanzado exitosamente todos los objetivos planteados inicialmente. Se llevó a cabo la exploración, selección, limpieza y adaptación de un conjunto de datos adecuado según los recursos disponibles. Además, se establecieron los fundamentos del aprendizaje profundo, lo que permitió una comprensión adecuada del funcionamiento de los detectores de objetos. Se adquirió el conocimiento necesario para entrenar y evaluar modelos de detectores de objetos utilizando la herramienta Matlab, mediante un análisis detallado de ejemplos paso a paso.

Durante la realización del trabajo, se profundizó en la distinción entre los detectores de objetos de una etapa y dos etapas. Como resultado, se implementaron y evaluaron distintas configuraciones de los detectores de objetos R-CNN y YOLO, y se comparó su rendimiento. Una vez se obtuvo el modelo de detector de objetos con mejor rendimiento para cada tipo, se procedió a tratar de mejorar los resultados mediante la variación de los hiperparámetros optimizador y tamaño del lote. No obstante, se llegó a la conclusión de que se estaba experimentando *overfitting* debido al uso de un conjunto de datos limitado. Se determinó como la forma más conveniente para superar este problema, la aplicación de técnicas de *data augmentation* o la expansión del conjunto de datos inicial para abordar este problema. Dado que los recursos computacionales eran escasos, ambas opciones resultaron inviables debido al aumento significativo del coste computacional.

Conviene reseñar al respecto, el hecho de que los rendimientos obtenidos en todos los casos no han sido suficientemente satisfactorios. Por lo tanto, resulta imprescindible investigar otras alternativas para resolver el problema, como la utilización de conjuntos de datos alternativos o la experimentación con configuraciones de hiperparámetros diversas. No obstante, desde el punto de vista técnico, teniendo en cuenta la naturaleza de este tipo de trabajos, se ha conseguido llevar a cabo el diseño e implementación de los detectores mencionados para su aplicación a un determinado *dataset*, habiendo establecido las bases para su utilización con carácter general en cualquier *dataset* y para la utilización de otros detectores de objetos existentes dentro de las dos categorías analizadas, esto es, una etapa y dos etapas.

A pesar de haber alcanzado los objetivos inicialmente propuestos, se han identificado varias líneas de mejora en las que trabajar en el futuro:

- Debido a los recursos disponibles se limitó el conjunto de datos para reducir los tiempos de entrenamiento. Utilizando técnicas de *data augmentation* y expandiendo el conjunto de datos inicial se podría solventar el problema de *overfitting*.
- Solucionar el problema de estimación de *anchor boxes* para el detector de objetos YOLO para poder evaluar su rendimiento correctamente.
- Utilizar hardware más potente que permita reducir las limitaciones encontradas durante el entrenamiento y le evaluación de los modelos
- Dentro de los detectores de objetos de dos etapas existen variantes del detector R-CNN como pueden ser Fast R-CNN, Faster R-CNN o Cascade R-CNN que mejoran la eficiencia y la velocidad del detector. Por otro lado, se podrían haber estudiado otros detectores de una etapa como puede ser el *Single Shot Detector* (SSD).
- Entrenar los modelos utilizando todas las clases disponibles en el *dataset* MS COCO en vez de un conjunto limitado debido a los recursos disponibles para analizar el rendimiento obtenido de los modelos cuando el número de clases aumenta.
- Utilizar diferentes configuraciones de hiperparámetros con el fin de mejorar el rendimiento de los modelos de detectores de objetos.

Chapter 5

Conclusions and Future Work

The work has successfully achieved all the initially set objectives. The exploration, selection, cleaning, and adaptation of a suitable dataset were carried out according to the available resources. Additionally, the fundamentals of deep learning were established, allowing for a proper understanding of how object detectors work. The necessary knowledge for training and evaluating object detector models using the Matlab tool was acquired through a detailed analysis of step-by-step examples.

Throughout the work, a distinction between single-stage and two-stage object detectors was explored in depth. As a result, different configurations of R-CNN and YOLO object detectors were implemented and evaluated, and their performance was compared. Once the best-performing object detector model was obtained for each type, attempts were made to improve the results by varying optimizer hyperparameters and batch size. However, it was concluded that overfitting was occurring due to the use of a limited dataset. The most suitable approach to overcome this issue was determined to be the application of data augmentation techniques or expanding the initial dataset to address this problem. Since computational resources were limited, both options proved to be unfeasible due to a significant increase in computational cost.

It is worth noting that the performance achieved in all cases has not been sufficiently satisfactory. Therefore, it is essential to explore other alternatives to address the issue, such as the use of alternative datasets or experimenting with different hyperparameter configurations. However, from a technical standpoint, considering the nature of such work, the design and implementation of the mentioned detectors for application to a specific dataset have been successfully carried out, laying the foundations for their general use with any dataset and for the use of other existing object detectors within the two analyzed categories, i.e., single-stage and two-stage.

Despite having achieved the initially proposed objectives, several areas for improvement have been identified for future work:

- Due to the available resources, the dataset was limited to reduce training times. Using data augmentation techniques and expanding the initial dataset could address

the overfitting problem.

- Resolve the issue of anchor box estimation for the YOLO object detector to properly evaluate its performance.
- Utilize more powerful hardware to reduce the limitations encountered during model training and evaluation.
- Within two-stage object detectors, there are variants of the R-CNN detector, such as Fast R-CNN, Faster R-CNN, or Cascade R-CNN, that improve efficiency and speed. On the other hand, one-stage detectors like SSD could have been studied.
- Train the models using all available classes in the MS COCO dataset instead of a limited set due to resource constraints to analyze the model's performance when the number of classes increases.
- Using different hyperparameter configurations in order to improve the performance of object detector models.

Bibliografía

Alexe, B., Deselaers, T. y Ferrari, V. Measuring the objectness of image windows. *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 34(11), páginas 2189–2102, 2012.

Bochkovskiy, A., Wang, C. y Mark-Liao, H. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv*, 2020.

Cherkassky, V. y Mulier, F. *Learning from Data: Concepts, Theory, and Methods*. Wiley, New York, USA, 1998.

Chollet, F. *Deep Learning with Python*. Manning, 2017.

ClickUp. ClickUp, Disponible on-line `https://clickup.com`, 2023.

Deng, J., Dong, W., Socher, R., Li, L., Li, K. y Fei-Fei, L. Imagenet: A large-scale hierarchical image database. En *Proc. Computer Vision Pattern Recognition*. Miami, FL, USA, 2009.

Documentation, P. Python Documentation, Disponible on-line `https://docs.python.org/es/3/library/shutil.html`, 2023.

Endres, I. y Hoiem, D. Category independent object proposals. En *Computer Vision – ECCV 2010* (editado por K. Daniilidis, P. Maragos y N. Paragios), vol. 6315 de *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2010.

Girshick, R. Pandas, Disponible on-line `https://pypi.org/project/pycocotools/#description`, 2018.

Girshick, R., Donahue, J., Darrell, T. y Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. En *Proc. 2014 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'14)*, páginas 580–587. 2014.

Ioffe, S. y Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167v3*, 2015.

Jaccard, P. Nouvelles recherches sur la distribution florale. *Bull. Soc. Vaudoise Sci. Nat.*, vol. 44, páginas 223–270, 1908.

Kingma, D. y Ba, J. Adam: A method for stochastic optimization. En *Proc. 3rd International Conference on Learning Representations (ICLR 2015)*, páginas 1–15. 2015.

Krizhevsky, A., Sutskever, I. y Hinton, G. Imagenet classification with deep convolutional neural networks. En *Proc. 25th Int. Conf. on Neural Information Processing Systems (NIPS'12)*, vol. 1, páginas 1097–1105. 2012.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L. y Dollár, P. Microsoft coco: Common objects in context. En arXiv [cs.CV]. http://arxiv.org/abs/1405.0312, 2015.

MathWorks. Matlab, Disponible on-line https://es.mathworks.com, 2023a.

MathWorks. The MathWorks, Disponible on-line https://es.mathworks.com/help/vision/, 2023b.

MathWorks. The MathWorks, Disponible on-line https://es.mathworks.com/products/deep-learning.html, 2023c.

MathWorks. The MathWorks, Disponible on-line https://es.mathworks.com/products/image.html, 2023d.

MathWorks. The MathWorks, Disponible on-line https://es.mathworks.com/products/parallel-computing.html, 2023e.

MathWorks. The MathWorks, Disponible on-line https://es.mathworks.com/products/statistics.html, 2023f.

MathWorks. The MathWorks, Disponible on-line https://es.mathworks.com/matlabcentral/fileexchange/59133-deep-learning-toolbox-model-for-alexnet-network, 2023g.

MathWorks. The MathWorks, Disponible on-line https://es.mathworks.com/matlabcentral/fileexchange/70986-deep-learning-toolbox-model-for-mobilenet-v2-network, 2023h.

MathWorks. The MathWorks, Disponible on-line https://es.mathworks.com/matlabcentral/fileexchange/64626-deep-learning-toolbox-model-for-resnet-50-network, 2023i.

MathWorks. Introducción a la transferencia del aprendizaje. En MathWorks. https://es.mathworks.com/help/deeplearning/gs/get-started-with-transfer-learning.html, 2023j.

McCarthy, J. What is artificial intelligence? 2004.

Microsoft. Common objects in context. En cocodataset.org. https://cocodataset.org/, 2014.

Microsoft. Visual Studio Code, Disponible on-line https://code.visualstudio.com, 2023.

Numpy. Numpy, Disponible on-line https://numpy.org, 2023.

Pajares, G., Herrera, P. y Besada, E. *Aprendizaje Profundo*. RC-Libros, Madrid, 2021.

Pandas. Pandas, Disponible on-line https://pandas.pydata.org, 2023.

Pang, J., Chen, C., Shi, J., Feng, H., Ouyang, W. y Lin, D. Libra r-cnn: Towards balanced learning for object detection. En *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, páginas 821–830. 2019.

Redmon, J., Divvala, S., Girshick, R. y Farhadi, A. You only look once: Unified, real-time object detection. En *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, páginas 779–788. Las Vegas, USA, 2016.

Rossum, G. V. Python, Disponible on-line `https://www.python.org`, 2023.

Rumelhart, D. E., Hinton, G. E. y Williams, R. J. Learning representations by back-propagating errors. *Nature*, vol. 323(6088), páginas 533–536, 1986.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. y Rabinovich, A. Going deeper with convolutions. *Computing Research Repository*, 2014.

Vapnik, V. *The Nature of Statistical Learning Theory*. Wiley, New York, 2000.

Wang, X., Ming, Y., Zhu, S. y Lin, Y. Regionlets for generic object detection. *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 37(10), páginas 2071–2084, 2015.

Yu, B. y Malan, D. J. Lecture 5 - CS50's introduction to artificial intelligence with Python, Disponible on-line `https://cs50.harvard.edu/ai/2020/notes/5/`, 2020.

Zhang, A., Lipton, Z., Li, M. y Smola, A. Dive into deep learning. Disponible en línea, 2020.

Zhang, H., Cisse, M., Dauphin, Y. y Lopez-Paz, D. Mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412v2*, 2018.

Zitnick, C. y Dollár, P. Edge boxes: Locating object proposals from edges. En *Computer Vision-ECCV* (editado por S. I. Publishing), páginas 391–405. 2014.

Glosario

AA *Aprendizaje Automático*. 1, 2, 4, 37

ADAM *Adaptive Moment Estimation*. 20, 58

AI *Artificial Intelligence*. 7–9

ANN *Artificial Neural Network*. XIII

AP *Aprendizaje Profundo*. XI, 2–5, 13, 29, 49, 50

AP *Average Precision*. 51, 54–57, 60

AR *Average Recall*. 29

CLIPS *C Language Integrated Production System*. 4, 9

CNN *Convolutional Neural Network*. 10

CPU *Central Processing Unit*. 47

CSV *Comma-Separated Values*. XXI, 44, 48, 75, 76

DL *Deep Learning*. XIII, 8, 9

GPU *Graphic Processing Unit*. 47

IA *Inteligencia Artificial*. 1–4, 47

IDE *Integrated Development Environment*. 3, 8, 9

IoU *Intersección sobre la unión*. XX, 28–32, 34, 35, 50

JSON *JavaScript Object Notation*. 43

mAP *mean Average Precision*. XXIII, 29, 51, 54, 57, 58

mAR *mean Average Recall*. 29

ML *Machine Learning*. 1, 7–9

MLP *Multilayer Perceptron*. XIX, 15–17

MS COCO *Microsoft Common Objects in Context*. XX, 41–43, 48, 51, 62, 64, 71, 72, 75

NMS *Non-Maximum Supression*. 32, 36

R-CNN *Regiones con CNN*. XI, XIII, XVI, XX, XXIII, 27, 32, 33, 48–58, 61–64, 73–75

RAM *Random Access Memory*. 47

ReLU *Rectified Linear Unit*. 15, 24

RLE *Run-Length Encoding*. 43

RMSProp *Root Mean Square Propagation*. 19, 20, 58

RNA *Red Neuronal Artificial*. 13, 14

RNC *Red Neuronal Convolucional*. XX, XXI, XXIII, 4, 20, 21, 27, 33–35, 48–52, 54, 56–59, 72, 76

ROI *Region of Interest*. 33, 44

SGD *Stochastic Gradient Descent*. 18, 19

SGDM *Stochastic Gradient Descent with Momentum*. 19, 52, 58, 59

SSD *Single Shot Detector*. 62, 64

SVM *Support Vector Machine*. 33

Tanh *Hyperbolic Tangent*. 15

FN *Falso Negativo*. 28

FP *Falso Positivo*. 28

VN *Verdadero Negativo*. 28

VP *Verdadero Positivo*. 28, 35

VSCode *Visual Studio Code*. 45

YOLO *You Only Look Once*. XI, XIII, XVI, XX, XXIII, XXIV, 34, 35, 48–51, 58–63, 75

Apéndice A

Código fuente

Este apéndice contiene partes relevantes del código fuente implementado que se ha referenciado en capítulos anteriores.

```
    {
        "info": info,
        "images": [image],
        "annotations": [annotation],
        "licenses": [license],
    }
    info: {
        "year": int,
        "version": str,
        "description": str,
        "contributor": str,
        "url": str,
        "date_created": datetime,
    }
    image: {
        "id": int,
        "width": int,
        "height": int,
        "file_name": str,
        "license": int,
        "flickr_url": str,
        "coco_url": str,
        "date_captured": datetime,
    }
    license: {
        "id": int,
        "name": str,
        "url": str,
    }
```

Figura A.1: Estructura del fichero que contiene las anotaciones del *dataset* MS COCO

```
annotation: {
    "id": int,
    "image_id": int,
    "category_id": int,
    "segmentation": RLE or [polygon],
    "area": float,
    "bbox": [x,y,width,height],
    "iscrowd": 0 or 1,
}
categories: [
    {
        "id": int,
        "name": str,
        "supercategory": str,
    }
]
```

Figura A.2: Estructura de una anotación del *dataset* MS COCO

```
load('alexnet.mat');
lgraph = layerGraph(net);
lgraph = replaceLayer(
    lgraph,
    net.Layers(length(net.Layers) - 2).Name,
    fullyConnectedLayer(
        12,
        'WeightLearnRateFactor', 20,
        'BiasLearnRateFactor', 20
    )
);
lgraph = replaceLayer(
    lgraph,
    net.Layers(length(net.Layers) - 1).Name,
    softmaxLayer()
);
lgraph = replaceLayer(
    lgraph,
    net.Layers(length(net.Layers)).Name,
    classificationLayer()
);
```

Figura A.3: Carga del modelo preentrenado con la arquitectura de RNC AlexNet para realizar la transferencia de aprendizaje en Matlab

```
% Set training options
options = trainingOptions(
    'sgdm', ...
    'Momentum', 0.9, ...
    'InitialLearnRate', 0.001, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.1, ...
    'LearnRateDropPeriod', 1, ...
    'L2Regularization', 0.0001, ...
    'MaxEpochs', 4, ...
    'MiniBatchSize', 128, ...
    'Shuffle','every-epoch', ...
    'Plots','training-progress', ...
    'Verbose', true, ...
    'VerboseFrequency', 430, ...
    'ExecutionEnvironment', 'parallel'
);

% Train an R-CNN object detector
rcnn = trainRCNNObjectDetector(
    train,
    lgraph,
    options
);
```

Figura A.4: Función que permite entrenar el detector de objetos R-CNN dadas unas opciones de entrenamiento

```
numImages = height(val);
results = table(
    'Size',[numImages 3], ...
    'VariableTypes', {'cell','cell', 'cell'}, ...
    'VariableNames', {'Boxes','Scores','Labels'}
);

for i = 1 : numImages
    I = imread(val.path{i});
    [bboxes, scores, labels] = detect(
        rcnn,
        I
    );
    results.Boxes{i} = bboxes;
    results.Scores{i} = scores;
    results.Labels{i} = labels;
end

blds = boxLabelDatastore(val(1:numImages,2:end));
[ap,recall,precision] = evaluateDetectionPrecision(
    results,
    blds
);
```

Figura A.5: Proceso de validación del rendimiento del detector de objetos R-CNN

Apéndice B

Descarga e instalación del proyecto

El código utilizado durante la realización de este trabajo se encuentra disponible en el repositorio *on-line* `https://github.com/jlmpepe/tfm-project`.

Este repositorio contiene varios archivos de código fuente, siendo dos de ellos los pertenecientes a la implementación realizada en Matlab de los detectores de objetos R-CNN y YOLO y otro el programa escrito en Python que genera los conjuntos de datos de entrenamiento, validación y test a partir del *dataset* MS COCO.

Primero se deben generar los conjuntos de datos ejecutando el programa de Python. En el caso de no disponer de un intérprete de Python, se podrá descargar de su página *web* oficial. Antes de ejecutar el programa, se deberán instalar las dependencias utilizando el fichero «Pipfile» utilizando el comando «pipenv install». Tras esto, se puede proceder a ejecutar el programa como se observa en la figura B.1.

```
(coco-dataset) (base) pepe9817@MacBook-Pro-de-Jose coco-dataset % python generate_datasets.py
loading annotations into memory...
Done (t=6.80s)
creating index...
index created!
ID: 1 Name: person
ID: 16 Name: bird
ID: 17 Name: cat
ID: 18 Name: dog
ID: 19 Name: horse
ID: 20 Name: sheep
ID: 21 Name: cow
ID: 22 Name: elephant
ID: 23 Name: bear
ID: 24 Name: zebra
ID: 25 Name: giraffe
Total number of classes: 11
```

Figura B.1: Generación de los conjuntos de datos de entrenamiento, validación y test

En la figura B.2 se pueden ver los ficheros CSV generados tras la realización del anterior paso. El formato de los ficheros generados es el mismo para todos, siendo la primera columna la que contiene la ruta a la imagen y el resto de las columnas (tantas columnas como clases haya) contienen las localizaciones de los objetos dentro de la imagen, es decir, los *bounding boxes* que determinan la posición del objeto dentro de esta.

Figura B.2: Documentos en formato CSV que contienen las *ground truth tables*

Tras generar los conjuntos de datos, se debe abrir el programa Matlab e importar los archivos con extensión «.mlx» y «.m» del repositorio anteriormente mencionados. En el caso de no disponer del programa Matlab, se podrá descargar también de su página *web* oficial. Acto seguido se deberán instalar las librerías mencionadas en el punto 3.1.7 para poder realizar el entrenamiento y la validación de los modelos. En el caso de que no se haya instalado alguna de las librerías específicas, Matlab lanzará una excepción indicando la librería que falta por instalar.

Antes de ejecutar el código, se deben modificar las rutas de las carpetas que contienen los conjuntos de datos que aparecen en la figura B.3, así como la ruta de la carpeta que contiene las imágenes como se puede ver en la figura B.4.

Load Training, Validation and Test Data

```
addpath('/Users/pepe9817/Github/tfm/coco-dataset/');
train = loadCSV("/Users/pepe9817/Github/tfm/coco-dataset/train.csv", '\t');
val = loadCSV("/Users/pepe9817/Github/tfm/coco-dataset//val.csv", '\t');
test = loadCSV("/Users/pepe9817/Github/tfm/coco-dataset/test.csv", '\t');
```

Figura B.3: Carga de los conjuntos de datos

```
% Add path to dataset
addpath('/Users/pepe9817/Github/tfm/coco-dataset/dataset');
```

Figura B.4: Añadir ruta del directorio donde se encuentran las imágenes de los conjuntos de datos

Tanto la arquitectura de RNC utilizada como columna vertebral por el detector de objetos como los parámetros de entrenamiento pueden ser modificados como se puede observar en las figuras B.5 e B.6.

```
% Set ANN arquitecture
net = alexnet;
% net = mobilenetv2;
% net = resnet50;
```

Figura B.5: Cambio de la arquitectura de RNC utilizada como columna vertebral por el detector de objetos

```
% Set training options
options = trainingOptions('sgdm', ...
    'Momentum', 0.9, ...
    'InitialLearnRate', 0.001, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.1, ...
    'LearnRateDropPeriod', 1, ...
    'L2Regularization', 0.0001, ...
    'MaxEpochs', 4, ...
    'MiniBatchSize', 128, ...
    'Shuffle','every-epoch', ...
    'Plots','training-progress', ...
    'Verbose', true, ...
    'VerboseFrequency', 430, ...
    'ExecutionEnvironment', 'parallel');
```

Figura B.6: Parámetros de entrenamiento modificables

Finalmente, se ejecutará el fichero al completo pulsando al botón «*Run*» donde se entrenará y validará el modelo en cuestión. Como resultado, se obtendrá el modelo de detección de objetos listo para ser utilizado y el rendimiento obtenido con dicho modelo.