Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Axuste adaptativo de rexións de interese na detección de tomates en secuencias de vídeo

Estudante: Nicolás Martínez González

Dirección: Carlos Vázquez Regueiro

Xosé Manuel Pardo López

A Coruña, febrero de 2024.

Dedicado a mi novie, mi familia y mis amigos por ser un apoyo enorme y una referencia constante.

Agradecimientos

Gracias a todas las personas que me han respaldado a lo largo de este proceso y lo siguen haciendo con cada uno de mis proyectos.

Resumen

En este proyecto se desarrolla un sistema enfocado en la aplicación de redes convolucionales de última generación para la detección de objetos en condiciones reales de operación. En nuestro caso, la detección de tomates en un invernadero. Este conjunto de datos nos sirve para tratar varias cuestiones como el manejo de datos escasos que reflejan un entorno real como distribución irregular de objetos, oclusión por las hojas o variabilidad en las condiciones de luz, textura y perspectiva de la cámara.

En este trabajo hemos seleccionado, entre los modelos de redes convolucionales novedosos de acceso público, aquellos que mejor se podrían adaptar a nuestro problema. Después de varias pruebas, nos hemos decantado finalmente por EfficientDet, SSD, FasterR-CNN, FCOS y RetinaNet. Los cuatro últimos se encuentran en la librería Torchvision, siendo el primero una implementación pública tomada como referencia por la comunidad de PyTorch.

En base a una serie de experimentos se fueron modificando partes de las redes, estudiando varios métodos de entrenamiento, formas de extraer el máximo partido al reducido conjunto de datos e implementando técnicas como la umbralización para mejorar los resultados de cada modelo.

Abstract

The main point of this project is to develop a system that uses state-of-the-art architectures of convolutional neural networks in order to detect objects in a realistic context. In this case, detecting tomatoes within a greenhouse. For this we take as reference a public dataset that is useful to address issues such as data scarcity and optimization. This dataset presents several caracteristics that differ from general uses datasets, including irregular distribution of objects, occlusion by leaves or unstable light conditions, which we value as it reflects better the real world.

For this task, we chose between publicly available state-of-the-art architectures, selecting those that better suit the problem. After a few tests, we finally chone EfficientDet, SSD, FasterR-CNN, FCOS and RetinaNet, being the former a public implementation taken as reference by the PyTorch community and the others ones inside the Torchvision library.

Along with the experiments we came to introduce several changes to the models, studying different methods to train them, ways to optimize the scant data and implementing techniques to improve the model outputs, such as umbralization.

Palabras clave:

- Inteligencia artificial
- Aprendizaje profundo
- Visión por computador
- Detección de objetos
- Red neuronal convolucional

Keywords:

- Artificial intelligence
- Deep learning
- Computer vision
- Object detection
- Convolutional neural network

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo presentaremos el contexto en el que se plantea el proyecto, los objetivos que persigue, además de las propuestas halladas para llevarlos a cabo, junto al estado del arte y la estructura de la propia memoria.

1.1 Contexto del proyecto

En la última década se ha hecho notable el auge de las soluciones basadas en sistemas con aprendizaje máquina, en específico los que emplean redes profundas. Generalmente los sistemas que las implementan trabajan con un vasto conjunto de información sobre el contexto en que se pretende implantar, sino con una base de datos lo más general posible. Para nuestro problema, nos centramos en la aplicación de estos sistemas para un contexto específico de la agronomía, este es, el de identificar tomates. Para esto hemos tomado como referencia un conjunto de datos público en el que se recogen fotos tomadas por un robot en un invernadero en Oporto y etiquetadas manualmente. Con esto no sólo estamos modelando la tarea de detectar objetos en un entorno particular, sino que estamos teniendo que tener en cuenta tanto el reducido número de imágenes como su calidad.

Con esto, nuestro proyecto pretende ser un estudio de la posibilidad de implantar un sistema con aprendizaje profundo en un contexto lo más fiel a la realidad posible, mitigando los aspectos negativos de los datos disponibles y potenciando las técnicas a nuestra disposición para mejorarlo.

1.2 Objetivos del proyecto

Para este proyecto vamos a poner nuestro foco en el uso de modelos de redes de neuronas convolucionales (CNNs) para la detección de tomates en imágenes tomadas de secuencias de vídeo. Partiendo de la base de emplear una base de datos ya etiquetada conformada por fotos

tomadas en un invernadero, los dos principales objetivos de este proyecto son:

- Comparar el rendimiento y el coste de entrenamiento de arquitecturas CNN recientes empleadas para detectar objetos, prestando especial atención a que se puedan usar en dispositivos de bajas prestaciones.
- Analizar modificaciones en la arquitectura de alguna CNN para conseguir un ajuste adaptativo o más fino de los umbrales usados para determinar las regiones de interés.

1.2.1 Propuesta

La propuesta a la que pretendemos adherirnos se centra en el uso de modelos de redes de neuronas convolucionales ya disponibles, junto con un conjunto de imágenes etiquetadas para llevar a cabo la detección de tomates. En este aspecto cabe describir los principales requisitos del proyecto:

- La elección de los modelos debe estar supeditada a alcanzar la máxima eficiencia con los mínimos recursos, tanto en cuestión de espacio para alojar dicho modelo, como en cuestión del coste computacional del entrenamiento del mismo, y el tiempo necesario para llevarlo a cabo. Lo que se busca es construir un sistema capaz de funcionar en un equipo de bajas prestaciones.
- El conjunto de datos debe ajustarse al contexto que pretendemos modelar. Esto es, imágenes fieles de un entorno real con condiciones de visibilidad variables, distribución irregular de objetos y un número de datos etiquetados no especialmente amplio.

El proyecto en sí se basa en probar que el uso de modelos de redes preentrenadas con conjuntos de datos muy genéricos no son aplicables para contextos específicos como este, precisando un reentrenamiento con la información disponible del problema al que se pretende adaptar.

Aparte de esto, existe el propósito de adaptar estas redes en la medida de lo posible al contexto del problema, así como estudiar la viabilidad de implantar un método de umbralización automática que favorezca la precisión del sistema.

1.3 Trabajo relacionado

Como base para nuestro proyecto hemos estudiado casos con problemas similares al nuestro, siendo el primero el tratamiento de detecciones de objetos parcialmente expuestos.

Como primer trabajo revisamos el artículo “DeepVoting: A Robust and Explainable Deep Network for Semantic Part Detection under Partial Occlusion” [1], el cual trata la problemática de la detección semántica de objetos con oclusión. En él implementa una arquitectura tomando

como base una técnica para inferir la detección de los objetos ocluidos a partir del aprendizaje con objetos completos. De este artículo sacamos referencias de qué criterios tener en cuenta cuando se incluyen modificaciones en arquitecturas ya existentes y qué técnicas tener como referencia para tratar nuestro problema.

Otra publicación que examinamos fue "Occlusion Handling in Generic Object Detection: A Review" [2], la cual propone varias líneas de investigación para modelos que traten imágenes con oclusión. Además de detalles sobre el manejo de datos parciales, esta publicación fue de gran inspiración en torno a la elección de arquitecturas, mencionando Faster R-CNN o SSD entre otras.

Finalmente, hallamos "A survey on deep learning tools dealing with data scarcity: definitions, challenges, solutions, tips, and applications" [3], un estudio sobre la problemática del uso de conjuntos de datos pobres. Gracias a esto, pudimos comprender mejor nuestro problema y descubrir técnicas para paliar el posible sobreajuste de las redes, entre ellas, el aprendizaje por transferencia.

1.4 Estructura de la memoria

En esta sección se explicará la organización seguida para presentar esta memoria.

- Capítulo 1: Introducción. Es el capítulo actual, sirve para presentar las motivaciones detrás de este proyecto, sus objetivos y su descripción general.

- Capítulo 2: Fundamentos teóricos. En este capítulo se comentan los conceptos necesarios para comprender la librería empleada en el proyecto y las familias de modelos de CNN empleadas.

- Capítulo 3: Fundamentos tecnológicos. En este capítulo se explican las tecnologías y herramientas empleadas para llevar a cabo el proyecto.

- Capítulo 4: Metodología. En este capítulo se comentan tanto la planificación del proyecto como los requisitos, desarrollo y costes del mismo.

- Capítulo 5: Análisis de los modelos estudiados. En este capítulo se explica de forma general la estructura de una red de neuronas convolucional y en específico las estructuras de los modelos que hemos estudiado.

- Capítulo 6: Diseño e implementación. En este capítulo se exponen los componentes y funciones diseñados para el funcionamiento del sistema, así como métodos de procesamiento de los datos.

- Capítulo 7: Pruebas. En este capítulo se muestran los resultados de las pruebas ejecutadas sobre el sistema y se analizan dichos resultados.
- Capítulo 8: Conclusiones. En este capítulo final se examina el estado final del proyecto y se comentan posibles líneas de trabajo futuro.

Capítulo 2

Fundamentos teóricos

En este capítulo se definirán los conceptos teóricos en los que se basa el proyecto, incluyendo la Inteligencia Artificial y la detección de objetos mediante visión artificial, las Redes Neuronales Artificiales (RNAs) y sus variantes como las Redes Neuronales Convolucionales (CNNs) o Redes Neuronales de Propuesta de Regiones (R-CNN), aprendizaje por transferencia y las principales técnicas específicas de las CNNs.

2.1 Inteligencia Artificial

La inteligencia artificial [4] (IA) es un campo de la computación, con el propósito de diseñar sistemas que emulen la inteligencia humana en cierta capacidad. Este tema sirve como un paraguas para una gran cantidad de otros campos que abarca, tales como la visión artificial, el reconocimiento de voz, modelos de lenguaje, probadores de teoremas, diversos tipos de aprendizaje y la robótica.

A su vez, la IA cuenta con una infinidad cada vez mayor de aplicaciones tanto a nivel de producción de productos industriales como de uso cotidiano y muy popular, en ámbitos como la domótica, juegos en general o motores de recomendación de contenido.

2.1.1 Visión artificial

La visión artificial es una disciplina derivada de la inteligencia artificial que incluye métodos para procesar imágenes (vídeos y otros), extrayendo de ellas información de la misma forma que haría un humano.

El término en sí engloba un número de técnicas por los cuales un sistema (computadora) obtiene datos de una imagen con el fin de interpretarla de alguna forma. En general, estas técnicas se integran dentro de un sistema con algún tipo de aprendizaje automático; de esta forma se consigue emular el proceso de aprendizaje a base de mostrar imágenes, señalando en cada una qué información extraer.

2.1.2 Detección de objetos

La detección de objetos de un ámbito de la visión artificial enfocado en la identificación semántica de clases. Esto es, dada una imagen, saber reconocer una categoría y acotar la zona donde aparece. De esta forma, la tarea de detección de objetos aúna las dos funciones primarias de una red de neuronas artificiales: la clasificación y la regresión. El caso de la clasificación viene dado por la tarea de identificar a qué clase pertenece el objeto hallado, dejando el de la regresión para generar la caja o cajas que acotan la localización de dicho objeto.

2.2 Redes de Neuronas Artificiales

Las redes de neuronas artificiales (RNAs) son una especialidad del aprendizaje automático/IA con el objetivo de elaborar modelos que imiten el funcionamiento del cerebro a nivel neuronal.

Para esto, las redes están constituidas por nodos (neuronas), conectadas entre sí, simulando así las conexiones sinápticas de neuronas en el cerebro. Dentro de cada nodo/neurona reside un peso, resultado del aprendizaje, que ajusta la forma en la que se comunica con las demás neuronas. Generalmente, estas redes se organizan en capas, cada una con un distinto procesamiento de la entrada que reciben, yendo las señales desde la primera capa(capa de entrada) hasta la última (capa de salida).

El aprendizaje de estas redes se da a través de un proceso de minimización de una función de pérdida, que sirve para ponderar la red en conjunto. De este modo, cada neurona ajusta sus pesos para reducir la pérdida mediante un algoritmo de propagación hacia atrás.

2.2.1 Aprendizaje Profundo

El aprendizaje profundo/deep learning es una colección de métodos de aprendizaje automático, específico de las RNAs, que se usan para describir el aprendizaje representacional.

Las arquitecturas de aprendizaje profundo son aquellas que poseen más de una capar oculta, es decir, una capa que no sea ni de entrada ni de salida, sino que se comunica con una de estas o con otra capa oculta. Lo que se consigue con estas capas ocultas, es modelar el aprendizaje representacional, esto es, asimilar un concepto de forma más abstracta que una asociación sencilla de una entrada a una salida.

2.2.2 CNNs

Las Redes de Neuronas Convolucionales (CNNs) [5], son un tipo de RNAs que siguen el paradigma del aprendizaje profundo. Su peculiaridad reside en su arquitectura iterativa mediante filtros/kerneles, esto es, convolutiva.

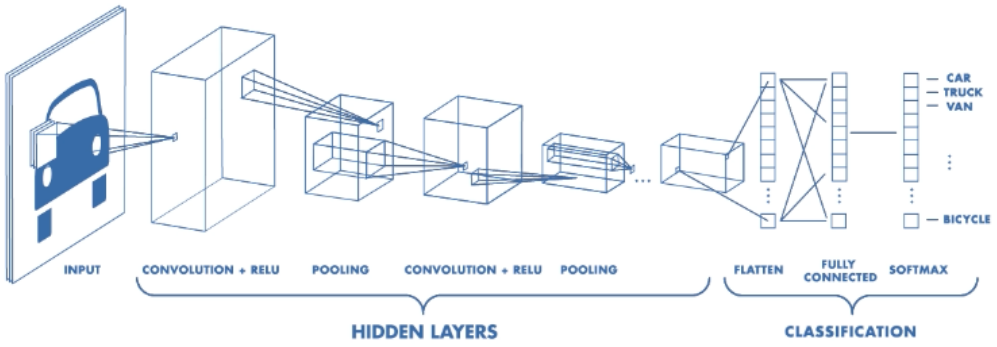


Figura 2.1: Arquitectura básica de una CNN

Igual que las RNAs, la base de su concepto reside en la imitación de estructuras biológicas, en este caso el córtex visual de los animales. La arquitectura de estos sistemas está constituida por varias capas con filtros convolucionales. Estos filtros abstraes la información de entrada, obteniendo características generales de la imagen.

2.2.3 Redes Neuronales de Propuesta de Regiones (R-CNN)

Las redes de neuronas de proposición de regiones (*Region Proposal RNAs/R-CNNs*) son una clase de modelos de RNAs diseñadas originalmente para detectar objetos en imágenes, aunque ciertos modelos desarrollados en la actualidad son capaces de realizar segmentación semántica.

Su funcionamiento se basa en aplicar un mecanismo de selección selectiva para obtener las regiones de interés en la imagen de entrada. Estas regiones se transmiten a la red para obtener las características de salida, aunque en las últimas versiones de algunos modelos el mecanismo de identificación de regiones está integrado en la propia red. Sobre esta base se han llevado a cabo modificaciones que vuelven al modelo más eficiente a la vez que más rápido tanto en tiempo de entrenamiento como de evaluación (Fast R-CNN).

2.3 Aprendizaje por transferencia

El aprendizaje por transferencia o *transfer learning* es un mecanismo usado en inteligencia artificial para transmitir la información ”aprendida” por un sistema en relación a una tarea específica a otro para reutilizarla. Si bien este concepto contiene varias acepciones para otros campos, vamos a centrarnos en la que ocupa al aprendizaje profundo.

Para poner en marcha esta técnica vamos a reutilizar los pesos de los modelos preentrenados con conjuntos de datos genéricos, ImageNet o COCO, por ejemplo, cargando estos pesos en los modelos que utilizaremos, o al menos en sus *backbones*/estructuras básicas.

Con esto lo que pretendemos es que nuestras arquitecturas repliquen la capacidad de los modelos preentrenados de reconocer las formas de ciertos objetos, para luego entrenar las cabezas de clasificación de las mismas para que aprendan a partir de nuestros datos.

Confiamos en este método no solo por heredar la información aprendida por sistemas con procedimientos más rigurosos, sino porque supone una agilización del proceso de entrenamiento de la red mucho mayor. Así, nos ahorramos entrenar el modelo por completo, parte notablemente más costosa del proceso de entrenamiento, y nos centramos en entrenar las cabezas de la red.

2.4 Técnicas específicas de las CNNs

En este apartado se comentan aspectos teóricos y técnicas específicas de las redes de neuronas convolucionales.

2.4.1 Fusión Ponderada de Cuadros

La fusión ponderada de cuadros o *weighted boxes fusion* es una reciente técnica para fusionar las *bounding boxes*/cajas delimitadoras propuesta por R. Solovyev en el artículo "Weighted boxes fusion: Ensembling boxes from different object detection models" [6].

Este proceso se utiliza para reducir el número de predicciones en base a sus bounding boxes, fusionando aquellas cuyo índice de solape supere un umbral especificado. A mayores, permite declarar otro umbral para descartar de esta fusión las *bounding boxes* que tengan una ponderación insuficiente.

El uso que le damos a esta técnica es a la hora de umbralizar los resultados devueltos por cada modelo, no solo teniendo en cuenta la ponderación de cada predicción individual, sino a través del límite a partir del cual consideramos que dos *bounding boxes* podrían ser la misma.

2.4.2 Intersección Sobre Unión/Índice de Jaccard

El Índice de Jaccard es un coeficiente que permite medir el grado de similitud entre dos conjuntos, independientemente del número de elementos. En el contexto de la detección de objetos es empleado para determinar cómo de bien una predicción se alinea con los valores de referencia o verdad fundamental, sirviendo para evaluar la precisión de la detección.

$$Jaccard(U, V) = \frac{|U \cap V|}{|U \cup V|}$$

2.4.3 *Distance-IoU Loss*

Esta función introducida en el artículo [7], la cual trata de especificar una métrica completa de pérdida en la detección de objetos basada en el Índice de Jaccard. Teniendo este índice como referencia, esta función pretende aunar aspectos de otras funciones como la pérdida de intersección sobre unión generalizada o *GIoU Loss* [8] para mejorar la detección.

De esta forma, la función resultante mantiene características comunes como la invarianza al escalado o la posibilidad de detección de la dirección si no existe solapamiento, mejorando en el aspecto de ser capaz de minimizar la distancia entre dos cajas, convergiendo más rápido.

Capítulo 3

Fundamentos tecnológicos

En esta sección comentaremos los contenidos necesarios para comprender la base tecnológica sobre la que se cimienta el proyecto, es decir, tanto los lenguajes de programación, aplicaciones usadas para escribir el código y visualizarlo, además de llevar un historial del mismo, ahondando en el hardware y el software empleados en el desarrollo del proyecto.

3.1 Lenguajes de programación y edición

En esta sección se comentan aspectos del principal lenguaje de programación empleado en la elaboración del código del sistema, además de las librerías y paquetes específicos. Estas últimas están relacionadas con la computación científica y el aprendizaje automático.

3.1.1 Python

Python es un lenguaje de programación de alto nivel interpretado, con un tipado dinámico, caracterizado por su versatilidad y legibilidad. Es un lenguaje que se centra en tener una sintaxis clara y una estructura basada en la identación, lo cual facilita su comprensión de forma intuitiva.

Uno de los principales puntos fuertes de Python es su amplia biblioteca estándar, además de las específicas elaboradas por su comunidad, debido a la propia naturaleza de código abierto de Python. Este aspecto es el que ha fomentado la inclusión de Python en la creación de bibliotecas y marcos de trabajo. Dichas bibliotecas creadas por la comunidad de Python abarcan muchos ámbitos, encontrándose aplicadas a un gran número de ámbitos tales como el desarrollo web, el análisis de datos o la inteligencia artificial.

3.1.2 LaTeX

LaTeX es un sistema de composición de textos dirigido a la creación de documentos científicos debido a su calidad tipográfica. En sí, se trata de un conjunto de instrucciones a partir

de comandos de TeX, que especifican el formato que darle al documento. Debido a su alta utilidad, potencia y facilidad de familiarización, así como el hecho de ser código abierto, hacen que se haya convertido en uno de los estándares para la escritura de documentos dentro del área científica.

3.2 Librerías de programación

En esta sección se mencionan las librerías Python empleadas para desarrollar el código.

3.2.1 NumPy

NumPy es una librería de Python que proporciona un soporte de matrices y arrays multidimensionales, así como de varias funciones matemáticas para realizar operaciones de forma eficiente sobre estos tipos de datos. El principal uso derivado de esta librería radica en el uso de arrays para los datos que maneja el modelo (bounding boxes, scores, clases, ...).

3.2.2 Pandas

Pandas es una librería de Python dedicada al análisis de datos. Pandas introduce varias estructuras de datos para facilitar su manejo, como por ejemplo el *DataFrame*. El *DataFrame* es una estructura tabular que permite el acceso a los datos de forma ordenada, con flexibilidad ante el tipo de datos y optimizada para trabajar con conjuntos de información grandes de forma eficiente. Precisamente por esta razón empleamos la estructura de *DataFrame* para registrar las anotaciones del conjunto de datos.

3.2.3 Tensorflow

TensorFlow es una de las dos principales librerías de código abierto para aprendizaje automático dentro del ecosistema de Python (junto a PyTorch). Diseñada por Google, permite la creación y entrenamiento de redes neuronales de un gran número de aplicaciones: reconocimiento de imágenes, procesamiento de lenguaje natural, aprendizaje profundo, modelos de clasificación/regresión, ...

La principal aplicación de esta librería yace en el hecho de que algunos modelos usados como esqueleto para otros son traducciones a PyTorch de modelos nativos de TensorFlow. También cabe destacar que la librería TensorBoard, detallada en adelante, aunque adaptada para su uso de forma conjunta con PyTorch, está diseñada originalmente por TensorFlow.

3.2.4 PyTorch

PyTorch es la otra de las dos librerías más populares de aprendizaje profundo en Python, diseñada por el grupo de IA de Facebook. Siendo relativamente más joven que TensorFlow, posee una gran popularidad, debido entre otras cosas, a su menor curva de dificultad, mayor extensión de documentación y eficiencia en diferentes aspectos en relación a otras librerías. Igualmente que TensorFlow, PyTorch ofrece un ecosistema de sublibrerías para un amplio rango de propósitos, tales como adaptación a sklearn, mejora de eficiencia en entrenamiento, optimización bayesiana, ….

Si bien la filosofía de PyTorch se basa en "usabilidad sobre rendimiento", la librería incluye numerosas adaptaciones para hacer posible el entrenamiento de forma distribuida tanto en CPU, como en GPU y TPU. En general, su criterio pone en primer lugar la facilidad de uso, flexibilidad y eficiencia de uso de memoria. En general, esta librería se caracteriza por su coherencia tanto interna, a la hora de permitir una completa interoperabilidad entre todos los módulos de su ecosistema, como externa, teniendo de forma nativa integrado el uso de otras librerías como NumPy.

3.2.5 Elección de PyTorch sobre TensorFlow

Para desarrollar el sistema de este hemos decidido usar PyTorch como marco de trabajo. Para tomar la decisión de emplear una librería sobre otra hay varios factores que hemos tenido en consideración:

- Aprendizaje. Si bien ambas librerías tienen una razonable curva de aprendizaje, la sintaxis con PyTorch se asemeja más a la de Python, lenguaje con el que mantengo un alto grado de familiarización, por lo que adaptarme a su uso sería más fácil.

- Eficiencia. Motivo principal para diseñar un sistema adaptable a equipos de bajas prestaciones. En este aspecto, PyTorch cumple este requisito dado que es ambivalente ante sistemas con más o menos recursos.

- Ejecución dinámica del grafo de computación. PyTorch crea el grafo de computación de forma dinámica, esto es, la secuencia de operaciones matemáticas llevadas a cabo por el modelo. Esto denota una gran flexibilidad a la hora del desarrollo de modelos de redes neuronales, pudiendo analizar el sistema de forma modular y crear prototipos y experimentar de forma más sencilla.

- Orientación. Debido al carácter flexible y dinámico de PyTorch, es ampliamente usado en contextos científicos y de investigación.

3.2.6 Torchvision

Debido a la elección de PyTorch como *framework* o entorno de desarrollo del proyecto, Torchvision es el paquete asociado a esta librería, el cual contiene datasets populares, arquitecturas de modelos y transformaciones útiles para el desarrollo de modelos de RNAs. Este paquete nos será útil en el desarrollo del proyecto para poder utilizar arquitecturas de modelos existentes, cargar sus pesos ya entrenados y realizar modificaciones para adaptarlas al contexto de nuestro problema.

3.2.7 Lightning

Lightning [9] es un marco de trabajo para el desarrollo de redes neuronales con aprendizaje profundo, en general, para contextos de investigación sobre aprendizaje automático. Esta librería ha sido de gran ayuda debido a que adapta el código de PyTorch necesario para manejar RNAs tanto en el contexto de entrenamiento y evaluación, como a la hora de diseñar el propio modelo, haciéndolo más flexible a la hora de reutilizar código y mejorar su legibilidad y comprensión.

Una de las principales mejoras que aporta Lightning es la posibilidad de distribuir el entrenamiento en GPUs o en CPU, según los recursos de la máquina en la que se ejecute, además de incluir una adaptación para poder registrar datos del modelo mientras se entrena para visualizarlos posteriormente a través de Neptune.ai. La inclusión de esta herramienta ha sido clave tanto en el diseño del código en el que se basa el sistema como para comprender la librería de PyTorch y en cierta medida el trabajo con modelos de Deep Learning.

3.2.8 Torchmetrics

Torchmetrics [10] es una biblioteca que comprende tanto funciones de evaluación de modelos de aprendizaje profundo como una API que permite el diseño de funciones personalizadas. El principal uso que se le ha dado a esta librería en el proyecto ha sido el de obtener funciones para evaluar modelos, en nuestro caso *MeanAveragePrecision*, la cual devuelve datos de la precisión y el *recall* para diferentes tamaños de *bounding box*.

3.2.9 Albumentations

Albumentations [11] es una librería de aumento de imágenes desarrollada en Python. Esta librería es de gran ayuda a la hora de ampliar artificialmente los conjuntos de datos, pues contiene un gran número de transformaciones que aplicar a cada imagen para modificarla. Entre las transformaciones disponibles existen algunas que cambian valores a nivel de píxel de la imagen, como cambio de valores RGB, HSV, ecualización de histograma y normalización, y otras que hacen cambios geométricos del estilo de translaciones, rotaciones y escalado.

Aunque el uso de librerías de aumento de imágenes sea crucial al trabajar con conjuntos de datos reducidos, es una práctica común al emplear conjuntos más amplios [12], ya que permite corregir fallos que puedan contener o adaptar las imágenes convenientemente al uso que se le quiera dar.

3.2.10 Weighted-Boxes-Fusion

Esta librería implementa métodos de procesamiento de cajas [13], generalmente las producidas por modelos de detección de objetos. Entre las implementaciones disponibles elegimos la que realiza la fusión ponderada de cajas.

3.3 Herramientas auxiliares

En esta sección se comentan otros medios que intervinieron en el desarrollo del proyecto, principalmente plataformas para visualización de los resultados, gestión de los ficheros del sistema y escritura.

3.3.1 Neptune.ai

Neptune.ai [14] es una aplicación web y API de Python que ofrece una plataforma para registro de modelos de aprendizaje automático, así como visualizarlos, compararlos y monitorizarlos. Posee una integración en numerosas librerías de Machine Learning como Keras, TensorFlow, PyTorch, Lightning, Azure ML o Docker. En nuestro caso hemos hecho uso de Neptune.ai para monitorizar los modelos durante su entrenamiento, registrando no solo métricas de precisión y pérdida, sino también hiperparámetros y datos del modelo, tales como el tiempo de entrenamiento, uso de CPU o de memoria.

3.3.2 Overleaf

Overleaf es una plataforma que facilita el proceso de escritura en LaTeX, incluyendo una interfaz amigable, posibilidad de edición concurrente, registro de versiones. Igualmente que LaTeX, debido a su baja curva de aprendizaje y usabilidad, es una opción para la edición de documentos LaTeX ampliamente empleada.

3.3.3 GitHub

GitHub es una plataforma de alojamiento de repositorios, dedicado al control de versiones. Permite la edición remota de proyectos, consulta y recuperación de versiones de ficheros

y administración de ramas de desarrollo, entre otras funciones. El principal uso de esta herramienta para el proyecto es el de poder editar ficheros del sistema desde dos máquinas distintas de forma remota, además del propio control de las versiones de ciertas partes del proyecto.

3.3.4 WinSCP

WinSCP es un cliente de SFTP, entre otros, de código abierto usado para la transferencia de documentos entre equipos remotos, permitiendo también las funciones de *scripting* y administración de archivos. El principal uso que se le ha dado a esta herramienta es para pasar el conjunto de datos y los pesos de los modelos entrenados entre mi equipo local y la máquina empleada para los entrenamientos.

3.3.5 Sublime Text 3

Sublime Text 3 es un shareware de edición de texto con soporte nativo de varios lenguajes de programación y *markup*. La elección de este editor de texto viene dada por su simplicidad, posibilidad de customización y necesidad mínima de recursos. Esta interviene en el proyecto siendo la principal herramienta para diseñar y editar el código del sistema.

3.3.6 GanttProject

GanttProject es una herramienta de gestión de proyectos de código abierto enfocada en la creación y gestión de cronogramas de proyectos. Esta posee una interfaz fácil de usar para la planificación de proyectos, permitiendo la definición de tareas, asignación de recursos y creación de dependencias entre ellas.

Capítulo 4

Planificación y dirección del proyecto

En este capítulo exponemos la metodología seguida para el desarrollo del proyecto y la gestión de la dirección del mismo, teniendo en cuenta los requisitos, recursos usados y costes finales.

4.1 Metodología de desarrollo

En esta sección se comentan los aspectos de la metodología designada para el proyecto. La elección de la metodología de desarrollo del proyecto viene dada por la inexperiencia tanto en el uso de librerías de aprendizaje automático como PyTorch como otras de procesamiento de datos. De esta forma, lo que buscamos es una metodología que nos permita conseguir resultados funcionales en iteraciones rápidas, pudiendo adaptar el rumbo del proyecto hacia la exploración de unas vías u otras en función de dichos resultados.

Teniendo todas estas consideraciones en cuenta, la metodología elegida para desarrollar este proyecto ha sido la metodología ágil, ya que es la que mejor se ciñe a nuestros requisitos.

4.1.1 Agile

La metodología Agile [15] es un conjunto de prácticas y marcos de trabajo reflejados por su manifiesto [16] y recogidos en sus 12 principios [17]. Estas prácticas tienen como objetivo el desarrollo del software de una forma rápida, contemplando la integración de cambios en cualquier fase y estableciendo una comunicación colaborativa con el cliente.

Así, esta metodología permite generar prototipos rápidamente, presentarlos al cliente y adaptarlo a sus requisitos de forma dinámica.

4.1.2 MLOps

MLOps [18] es una metodología de desarrollo centrada en la producción de sistemas en los que se emplean técnicas de aprendizaje automático. El fin de esta metodología es acelerar la adopción del aprendizaje máquina en el desarrollo de software y la entrega rápida de software inteligente. Los procesos de MLOps pasan por tres fases:

- Diseño de la aplicación que use inteligencia artificial. En esta fase se analizan el usuario potencial y las posibles soluciones con inteligencia artificial y se diseña el plan de trabajo.
- Experimentación y desarrollo empleando inteligencia artificial. En esta fase se verifica la aplicabilidad del uso de modelos de aprendizaje automático para el problema, desarrollando un modelo a modo de prueba de concepto. Se prueban diferentes algoritmos disponibles hasta encontrar una solución estable.
- Operaciones de aprendizaje automático. En esta fase se entrega el modelo generado por la fase anterior para ser sometido a las fases de desarrollo propias de las prácticas de DevOps, tales como prueba, versionado, entrega continua y monitorización.

Así, la metodología seguida por el proyecto sería una mezcla de MLOps con un enfoque ágil. De esta forma, los prototipos se generan de forma rápida, estando sujetos a un proceso de desarrollo iterativo en el que se barajan varias opciones y se escogen las mejores en base a experimentos. Estos prototipos se evalúan para tomar decisiones sobre qué modificaciones incluir, permitiendo adoptar otros algoritmos de forma dinámica.

En definitiva, el procedimiento de desarrollo del proyecto se basó en sesiones de seguimiento regulares cada semana, en la que según la fase del desarrollo se confeccionan prototipos, resúmenes de trabajos relacionados o resultados de evaluaciones. Aparte, se diseñaron informes a modo de síntesis del trabajo hecho y el pendiente, además de dudas surgidas durante el proceso

Para poder controlar las versiones de los archivos del proyecto se emplea un repositorio de GitHub en el que residen los ficheros para ejecutar las funciones del sistema. De esta forma, se pueden retomar opciones de implementaciones descartadas y comprobar que los resultados son replicables.

4.2 Estudio de requisitos

Como final de la primera reunión del proyecto se decidieron parte de los requisitos del sistema, los cuales se terminaron de establecer a medida que la familiarización con las herramientas era mayor. Si bien los requisitos funcionales estaban claros al revisar la propuesta de

los tutores, los no funcionales atendían a factores que se tuvieron en cuenta según se desarrollaba el proyecto.

4.2.1 Requisitos funcionales

Los requisitos funcionales especifican las funcionalidades que debe implementar el sistema. En este proyecto hemos considerado los siguientes:

RF1 Las redes empleadas deben ser entrenables con distintas técnicas y debe existir alguna técnica para evaluarlas entre sí.

RF2 Se deben implementar modificaciones al funcionamiento de las redes y/o a su arquitectura para tratar de obtener mejores resultados.

RF3 Las redes deben ser capaces de inferir correctamente a partir de imágenes nuevas.

RF4 Debe existir un método de umbralización de las predicciones de las redes, en específico, se debe estudiar uno que sea capaz de adecuarse a los datos de entrada.

4.2.2 Requisitos no funcionales

Los requisitos no funcionales son cualidades que debe tener el sistema para adecuarse a condiciones de tiempo, eficiencia o calidad. En este proyecto hemos considerado los siguientes:

RNF1 El sistema debe trabajar en base a un conjunto de datos reducido.

RNF2 Tanto el proceso de entrenamiento como de evaluación y de inferencia debe suceder en un tiempo razonable con una dedicación de recursos limitada.

RNF3 Tanto las herramientas como los datos empleados deben ser de acceso público, de forma que se puedan replicar o expandir en un futuro.

RNF4 Se debe priorizar la funcionalidad del sistema sobre otros detalles como el uso de una interfaz gráfica.

RNF5 El proyecto debe ajustarse a la metodología establecida y realizarse en el plazo especificado.

4.3 Fases del proyecto

En esta sección comentaremos las fases que conformaban el proyecto, las tareas de cada fase, problemas encontrados y medidas tomadas en base a estos.

4.3.1 Fase 1: Trabajo relacionado, estudio de herramientas y librerías.

Como fase inicial del proyecto, en este momento lo que se buscaba era una familiarización con el contexto del problema lo más rápida posible, para esto se estudiaron con atención documentos académicos sobre modelos del estado del arte, proyectos con aspectos similares al mío y en general las librerías disponibles para desarrollar el sistema.

Tareas

- Estudiar artículos y publicaciones sobre detección de objetos con aprendizaje profundo.
- Buscar conjuntos de datos coherentes al contexto de nuestro problema.
- Familiarización con las principales librerías de aprendizaje profundo en Python: TensorFlow y PyTorch
- Elección entre las dos anteriores.

4.3.2 Fase 2: Análisis de requisitos y planificación

En esta fase se sentaron las líneas de trabajo principales del proyecto, así como la metodología de desarrollo que se pretendía seguir.

Tareas

- Análisis de requisitos funcionales y no funcionales del sistema.
- Establecimiento de equipos con los que se trabajaría.
- Elección del conjunto de datos que emplearemos para el sistema.
- Marcar un horario de reuniones semanales.
- Establecer una metodología general del proyecto.

Problemas

Debido a que mi equipo no cuenta con las prestaciones necesarias para llevar a cabo de la forma adecuada el entrenamiento de los modelos y otros trabajos computacionalmente intensivos, decidimos solicitar el acceso a una máquina dentro de un servidor dedicado.

4.3.3 Fase 3: Desarrollo de un primer prototipo

En esta fase se llega a una versión temprana del sistema, la cual nos sirve como punto de partida sobre la que elaborar y para entender formas de atacar el problema del proyecto.

Tareas

- Primera versión del conjunto de datos procesado para que el sistema pueda trabajar con él.
- Diseño de una primera versión de los modelos tomados como referencia en base a los trabajos relacionados.
- Entrenamiento con el modelo y producción de ejemplos de inferencias.

Problemas encontrados

A la hora de producir este prototipo decidimos incorporar un modelo del estado del arte, cuya implementación no estaba incluida en una librería de modelos dentro de PyTorch, lo cual dificultaba la comprensión debido a la reducida documentación.

4.3.4 Fase 4: Mejora de las funcionalidades

El fin principal de esta fase giraba en torno a incluir modificaciones en el sistema que mejoraran la eficacia del modelo.

Tareas

- Elección de la distribución del conjunto de datos que mejor se adapte al contexto del problema.
- Pruebas de entrenamientos con diferente número de épocas para comprobar a partir se producía sobreajuste.
- Comprobar qué parámetros de umbralización propios del modelo generaban en este mejores resultados.

Problemas encontrados

En esta fase el principal problema que surgió fue el de decantarse por un enfoque sobre si redimensionar las imágenes del conjunto de datos o recortarlas a fin de adaptarlas al tamaño aceptado por el sistema.

4.3.5 Fase 5: Pruebas y validaciones

Llegados a esta parte del proyecto, ya teníamos cierta experiencia en el entrenamiento de modelos y manejo de predicciones, por lo que el siguiente paso era llegar a evaluarlos con alguna métrica disponible y corroborar que funcionaban correctamente. Para esto, se añadió

una función para las fases de validación del modelo que comparaba las detecciones en base al *benchmark* del conjunto de datos de COCO.

Tareas

- Análisis del sistema con casos de prueba.
- Registro del sistema durante sus fases de entrenamiento y validación para posteriormente visualizarlos.
- Evaluar el sistema en base a su precisión y *recall*.

Problemas encontrados

Al comenzar la fase de implementación de las funciones para evaluar la eficacia del modelo encontramos el inconveniente de adaptar el formato de las predicciones generadas por el modelo al formato de entrada de la función de evaluación.

4.3.6 Fase 6: Resumen y Revisión

Esta etapa estaba contemplada en la planificación inicial pero se alargó más de lo previsto debido a la decisión de expandir los modelos empleados por el sistema.

Tareas

- Reunión de los integrantes del proyecto para evaluar el progreso.
- Decisión de integrar otras arquitecturas en el sistema.
- Propuesta de técnicas para aplicar la umbralización a los resultados.

Problemas encontrados

El principal problema de esta fase fue la toma de decisiones sobre cómo enfocar el proyecto, decantándonos por ampliar su complejidad usando otros modelos, a pesar de que la duración del desarrollo se incrementase.

4.3.7 Fase 7: Desarrollo del segundo prototipo

Fase simétrica a la fase 3, este caso empleando implementaciones de modelos dentro de la librería Torchvision.

Tareas

- Adaptación del conjunto de datos procesado al formato de entrada de los modelos.
- Diseño de una versión básica del sistema en funcionamiento con las cuatro arquitecturas seleccionadas de Torchvision.
- Entrenamiento con el modelo y producción de ejemplos de inferencias.
- Diseño de una función de error propia común para todos los modelos.

Problemas encontrados

Los problemas de esta fase venían dados por la necesidad de familiarización con la librería de Torchvision y su documentación, teniendo que diseñar una función de pérdida común para poder compararlos posteriormente.

4.3.8 Fase 8: Mejoras del segundo prototipo

Fase simétrica a la fase 4, entrenando un número mayor de épocas para comprobar la posibilidad de sobreajuste y métodos distintos de umbralización.

Tareas

- Reentrenamiento de las redes un número mayor de épocas.
- Aplicación de la fusión ponderada de cajas para unir predicciones muy similares.
- Aplicación de umbralización en base a los índices de confianza de cada predicción.

Problemas encontrados

En este caso los principales problemas fueron los de coordinar varias redes para poder monitorizarlas y encontrar una solución común que satisfaga todas.

4.3.9 Fase 9: Evaluación del segundo prototipo

Fase simétrica a la fase 5, aplicando un criterio de evaluación a las nuevas redes.

Tareas

- Empleo de las métricas de COCO para evaluar la precisión de cada red.
- Comparación de los resultados variando umbrales de confianza y de solapamiento.
- Análisis de la monitorización de los entrenamientos de las redes.

Problemas encontrados

El principal obstáculo de esta fase vino por tratar de aplicar las mismas métricas de evaluación empleadas con EfficientDet para los nuevos modelos. Finalmente encontramos una implementación de la precisión media en la librería de Torchmetrics que empleaba las estadísticas de COCO. Otro factor fue la familiarización con la herramienta Neptune.ai para registrar datos de entrenamiento a modo de monitorización y comparación de las redes.

4.3.10 Fase 10: Documentación del proyecto

Como etapa final del proyecto, esta se centró en documentar el proceso de desarrollo del mismo, sirviendo para revisar aspectos específicos pasados por alto.

Tareas

- Organización de las pruebas, gráficos y datos generados durante el desarrollo del proyecto.
- Recopilación bibliográfica de las referencias tomadas en cuenta.
- Escritura de la memoria.

Problemas encontrados

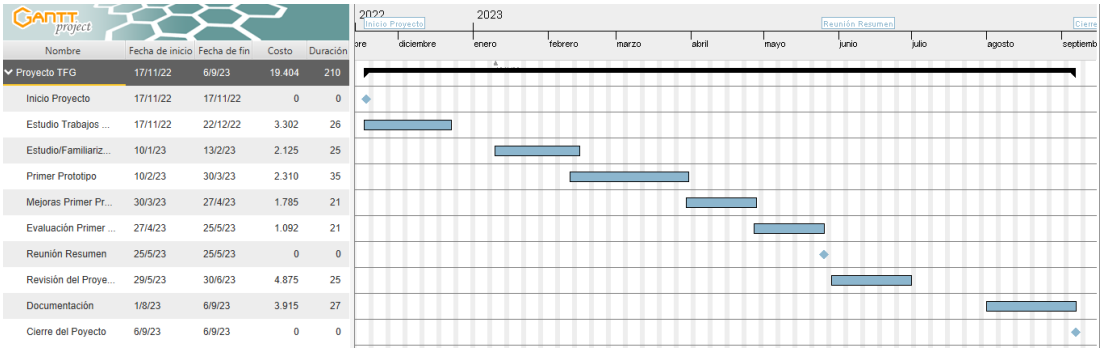
El principal problema durante esta fase fue el ajuste de la redacción de la memoria al plazo de entrega, teniendo que reunir resultados de experimentos, resúmenes de seguimiento y referencias empleadas.

4.4 Planificación y Seguimiento

En esta sección se exponen tanto la planificación inicial diseñada para el proyecto como el resultado final.

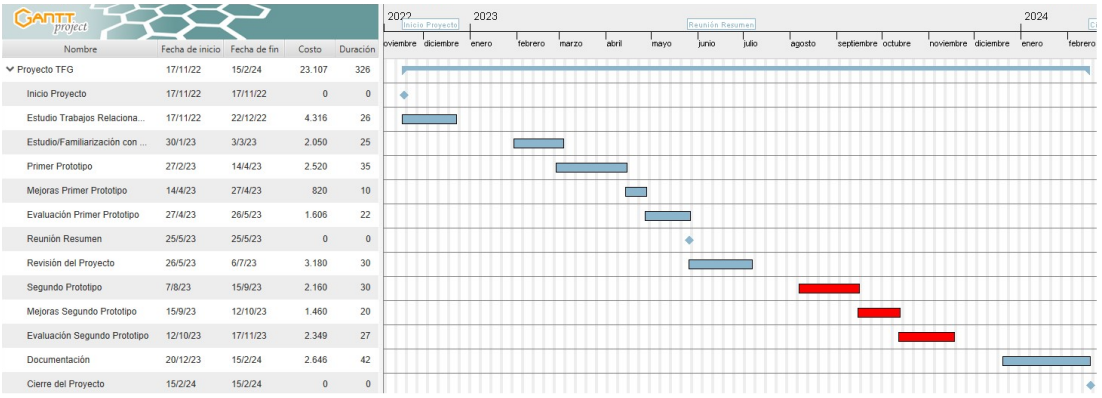
4.4.1 Planificación inicial

En base a la cantidad de horas que estimábamos invertir en el proyecto, diseñamos un esquema de la distribución de las tareas, dejando margen para dos a mayores, una de resumen del progreso y comentario del mismo con ambos tutores y otra de revisión del sistema previa a la documentación. Aparte de esto, tuvimos en consideración parones de vacaciones en las fechas de Navidades y de verano, además de gran parte del mes de enero de 2023 dedicado a la preparación y realización de exámenes del primer cuatrimestre. Con todo, el plan original se puede ver en la figura 4.1



Nombre	Fecha de inicio	Fecha de fin	Costo	Duración
Proyecto TFG	17/11/22	6/9/23	19.404	210
Inicio Proyecto	17/11/22	17/11/22	0	0
Estudio Trabajos ...	17/11/22	22/12/22	3.302	26
Estudio/Familiariz...	10/1/23	13/2/23	2.125	25
Primer Prototipo	10/2/23	30/3/23	2.310	35
Mejoras Primer Pr...	30/3/23	27/4/23	1.785	21
Evaluación Primer ...	27/4/23	25/5/23	1.092	21
Reunión Resumen	25/5/23	25/5/23	0	0
Revisión del Proye...	29/5/23	30/6/23	4.875	25
Documentación	1/8/23	6/9/23	3.915	27
Cierre del Poyecto	6/9/23	6/9/23	0	0

Figura 4.1: Planificación inicial del proyecto



Nombre	Fecha de inicio	Fecha de fin	Costo	Duración
Proyecto TFG	17/11/22	15/2/24	23.107	326
Inicio Proyecto	17/11/22	17/11/22	0	0
Estudio Trabajos Relaciona...	17/11/22	22/12/22	4.316	26
Estudio/Familiarización con ...	30/1/23	3/3/23	2.050	25
Primer Prototipo	27/2/23	14/4/23	2.520	35
Mejoras Primer Prototipo	14/4/23	27/4/23	820	10
Evaluación Primer Prototipo	27/4/23	26/5/23	1.606	22
Reunión Resumen	25/5/23	25/5/23	0	0
Revisión del Proyecto	26/5/23	6/7/23	3.180	30
Segundo Prototipo	7/8/23	15/9/23	2.160	30
Mejoras Segundo Prototipo	15/9/23	12/10/23	1.460	20
Evaluación Segundo Prototipo	12/10/23	17/11/23	2.349	27
Documentación	20/12/23	15/2/24	2.646	42
Cierre del Proyecto	15/2/24	15/2/24	0	0

Figura 4.2: Planificación final del proyecto

4.4.2 Planificación real

Pese a estar siguiendo de forma más o menos fiel el esquema propuesto, durante el periodo de revisión del proyecto se decidió introducir en el sistema el uso de más modelos, lo cual suponía tanto una nueva línea de desarrollo como una etapa de comparativa de todos las arquitecturas. Esto, como es de esperar, dilató la duración del proyecto, pasando de estar esperado para cerrarse en septiembre de 2023 a febrero de 2024.

El resultado final de la planificación se puede ver en la figura 4.2. En rojo se resaltan las nuevas tareas y su duración.

4.5 Recursos del proyecto

Como se comentó anteriormente, para del desarrollo del proyecto fueron necesarios dos equipos para dos tareas distintas.

El primero, mi equipo portátil personal con un procesador Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, el cual fue empleado para la mayor parte del diseño del sistema, dado que

Costes	Coste-trabajo (€)	Esfuerzo (horas)	Duración (días)
Estimado	19.404 €	390	210
Real	23.107 €	520	326

Tabla 4.1: Comparativa de los costes estimados y reales del proyecto.

no podía llevar a cabo el entrenamiento de modelos con millones de parámetros en un tiempo razonable.

El segundo, una máquina virtual alojada en un servidor con un procesador AMD Ryzen Threadripper 3970X 32-Core Processor, fue el dedicado a realizar las labores de entrenamiento, validación y pruebas, debido a su capacidad de soportar y ejecutar este tipo de tareas en tiempos relativamente bajos. Esto nos permitió, a su vez, comprobar el funcionamiento del sistema al procesar las operaciones con una CPU o con varios GPUs.

A la hora de realizar el cómputo del coste material del proyecto hemos decidido no tener en cuenta los equipos previamente mencionados, puesto que el portátil ya había sobrepasado su tiempo de amortización y la máquina virtual se contempla como donación para el proyecto.

4.6 Costes del proyecto

Para calcular el coste del proyecto relacionado con los recursos humanos se han tenido cuatro perfiles en cuenta: un analista, un consultor, un programador junior/desarrollador y un jefe de proyecto.

Tomando como base el sueldo medio en España de cada uno de estos perfiles de trabajador, hemos establecido un coste de 45€/hora para el analista y el consultor, 35€/hora para el programador junior y 55€/hora al jefe del proyecto.

Como se muestra en la tabla 4.1, el hecho de posponer el cierre del proyecto aumentó considerablemente el coste total, pasando de 19.404€ a 23.107€. Igualmente, vemos un notable aumento en el número de horas de trabajo dedicadas, lo cual se puede atribuir al hecho de que se realizó un segundo prototipo, el cual debía incluir mejoras y revisiones.

Capítulo 5

Arquitecturas de los Modelos Analizados

En este capítulo se expone la estructura de las redes de neuronas convolucionales, en adelante CNNs, los componentes principales de este proyecto. Comenzaremos con nociones sobre el esquema general de las CNNs para abordar las arquitecturas que hemos empleado para llevar a cabo los experimentos del proyecto. En cada sección se analizará el paradigma detrás de cada modelo y su arquitectura, así como características que los diferencian entre sí. Dichos modelos son de acceso público, tanto su arquitectura como sus pesos preentrenados. Dichos modelos son SSD, FasterRCNN, RetinaNet, FCOS y EfficientDet.

5.1 Arquitectura de las CNNs

Como se ha explicado previamente 2, las CNNs son una clase específica de redes de aprendizaje profundo, por lo que aparte de una capa de entrada y una de salida, están compuestas por una o varias capas ocultas. El nombre de estas redes viene dado debido a que dentro de estas capas ocultas se encuentra siempre al menos una capa de convolución.

5.1.1 Capas de una CNN

En esta sección se detallan las tipologías de capas más comunes en la arquitectura de CNNs

Capa convolucional

Es la base del funcionamiento de la CNN y la que conlleva mayor carga de computacional.

La convolución es el proceso de superponer una matriz/kernel sobre todas las porciones de la imagen de su tamaño con el fin de combinarlas en base a su producto escalar. Mediante

este proceso se identifican características presentes en la imagen y se transmiten a la siguiente capa.

Así, se obtiene la información representativa de la imagen, extrayendo de ella las características presentes en el kernel. Este kernel se define para reconocer patrones de formas, siendo la parte principal que se ajusta en el proceso de entrenamiento para adaptarse a los patrones que se hace que aprenda.

Normalmente, como una capa aparte, se suele añadir un rectificador (ReLU) a modo de función de activación de ciertos pesos de una neurona, permitiendo que la red sea capaz de identificar patrones más complejos.

Capa de agrupación

Esta base se encarga de reducir la dimensionalidad de los datos, fusionando el resultado de dos neuronas anteriores como entrada para una siguiente. Se suelen ubicar siguiendo a las capas convolucionales, de forma que se disminuya la complejidad computacional de la red. Con esta concentración de la información la red se queda con las características más importantes de la entrada, a la vez que se palía la redundancia de datos y el sobreentrenamiento u *overfitting*.

Según el modo de agrupación, las capas pueden ser de *MaxPooling* o *AveragePooling*, dependiendo de si se toman los valores máximos o un promedio.

Capa densa/*Fully-connected layer*

Las capas densas o completamente conectadas son conocidas como el modelo de capa tradicional de las RNAs, ya que en ellas cada neurona está conectada a cada neurona de la capa anterior y de la posterior. La función de esta capa es la de tratar las propiedades de forma global, imprimiendo una visión completa de la entrada.

Debido al tipo de entrada, esta clase de capas habita estar precedido por una capa o mecanismo de allanamiento o *flatten* que convierte los resultados multidimensionales a vectores de una dimensión.

5.1.2 Estructuras complejas dentro de una CNN

A partir de las capas, las cuales sirven como bloque básico de procesamiento de la información, dentro de la arquitectura de las CNNs es posible discernir módulos compuestos por capas que realizan una función en conjunto.

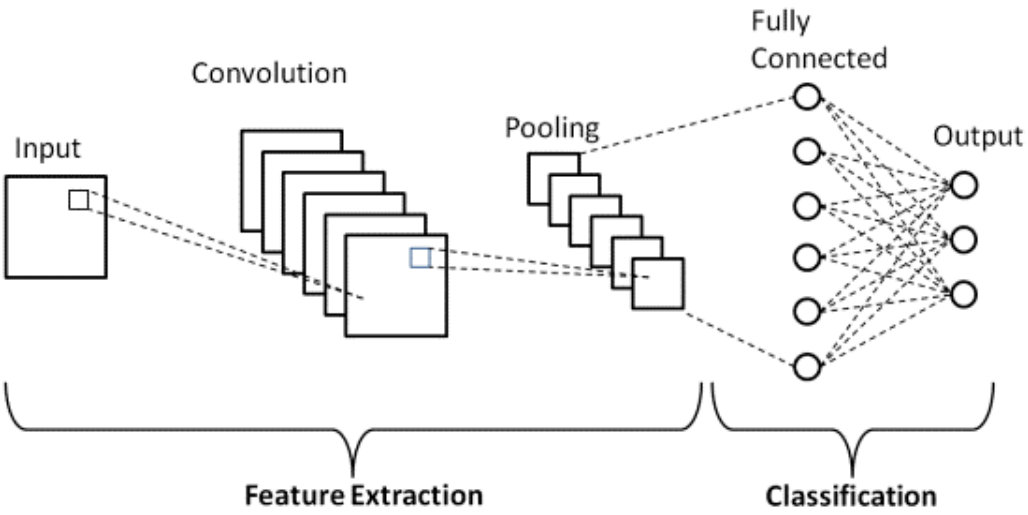


Figura 5.1: Capas de una CNN

Columna vertebral/*backbone*

Es el componente principal de la CNN, ya que se ocupa de identificar las características presentes en la imagen. En general están formadas por capas de convolución y de agrupación. Es común que las CNNs con cierto nivel de complejidad reutilicen modelos de arquitecturas y existentes como ResNet, VGG o MobileNet. Además de las arquitecturas, se suelen utilizar también sus valores entrenados con conjuntos de datos inmensos, de forma que se pueda replicar el proceso de identificación de características de los modelos mencionados.

Cuello/*Neck*

Este módulo sirve como preprocesador inmediato de las características extraídas por la columna vertebral, refinando la información de las mismas. Puede estar compuesto por capas convolucionales, funciones de agrupación espacial u otros métodos que mejoren la representación de las características. Como parte intermedia entre la columna vertebral y la cabeza sirve como módulo encargado de jerarquizar los datos de la imagen.

Cabeza

Es el componente encargado de emitir las predicciones en base a la información obtenida por la columna vertebral y procesada por el cuello. La naturaleza de cada cabeza atiende a la tarea que desempeñe, de este modo, por ejemplo, la cabeza de una red para la detección de objetos suele dividirse en dos: una para localizar cada objeto y otra para clasificarlo.

Del mismo modo, su estructura depende de la tarea que realice, pudiendo estar compuesta

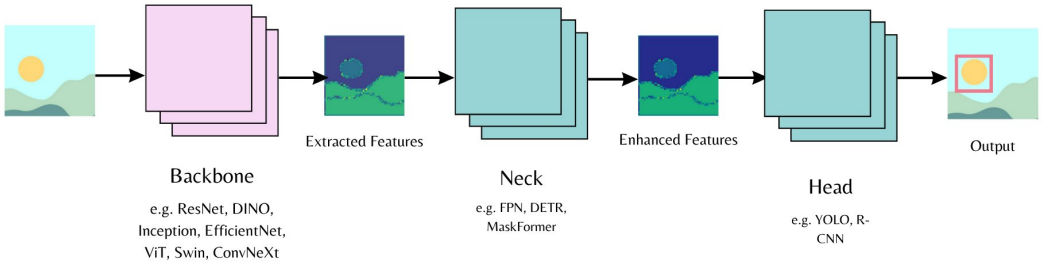


Figura 5.2: Módulos de una CNN

por capas densas, funciones de activación y otras operaciones.

5.2 EfficientDet

EfficientDet es un modelo diseñado por un equipo de investigadores de Google Research, entre los cuales se destaca a T. Mingxing, P. Ruoming y L. V. Quoc por introducirlo en el artículo de 2020 "EfficientDet: Scalable and Efficient Object Detection" [19]. Esta arquitectura se basa en la adición de modificaciones a una previa, EfficientNet, para alcanzar notables mejoras en eficiencia y precisión, resultados considerados del estado del arte.

5.2.1 BiFPN o Red Bidireccional de Pirámide de Características.

Es un módulo diseñado específicamente para la arquitectura de EfficientDet, implementando el paradigma de Red de Pirámide de Características con un nuevo formato. La Red de Pirámides de Características, en adelante FPN (*Feature Pyramid Network*) es un componente crucial en las arquitecturas de detección de objetos, implementada con regularidad por modelos de aprendizaje profundo.

El carácter piramidal se debe a que esta trabaja con imágenes a diferentes escalas de resolución/tamaño, lo cual facilita su identificación a diferentes niveles de abstracción. Con esto, se establecen fusiones de las características extraídas, las cuales se integran de abajo arriba o al revés, estableciendo jerarquías entre estos niveles de abstracción. De esta forma, el modelo es capaz de explotar la información contextual de los niveles a menor escala y los detalles de grano fino de los niveles con mayor resolución.

Trabajando sobre esto, los cambios que propone BiFPN a esta estructura se basan en aumentar el número de conexiones entre los niveles de esta pirámide. Así, la información de características fluye en ambas direcciones de los niveles, permitiendo que la información contextual y en detalle se retroalimenten entre sí, favoreciendo la capacidad de reconocimiento de los objetos a varias escalas.

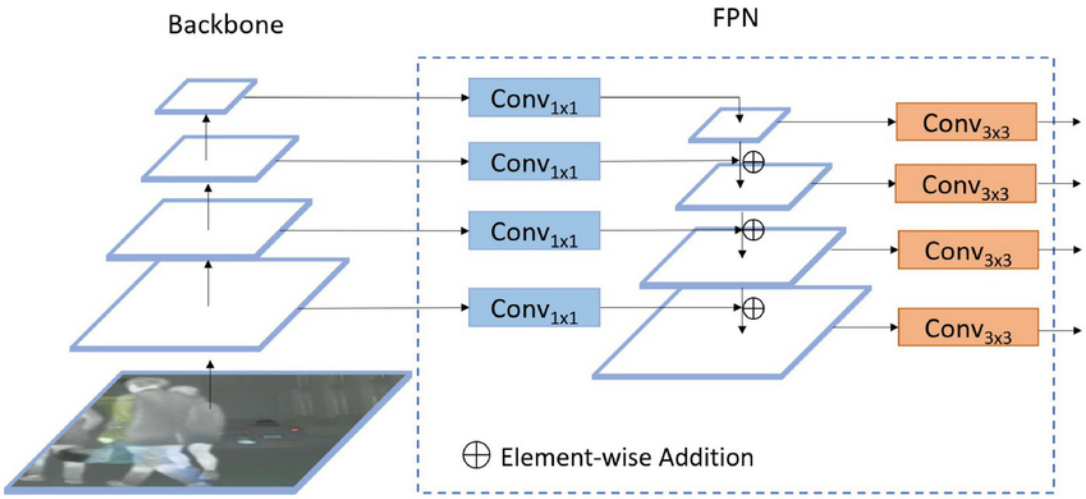


Figura 5.3: Funcionamiento de una FPN

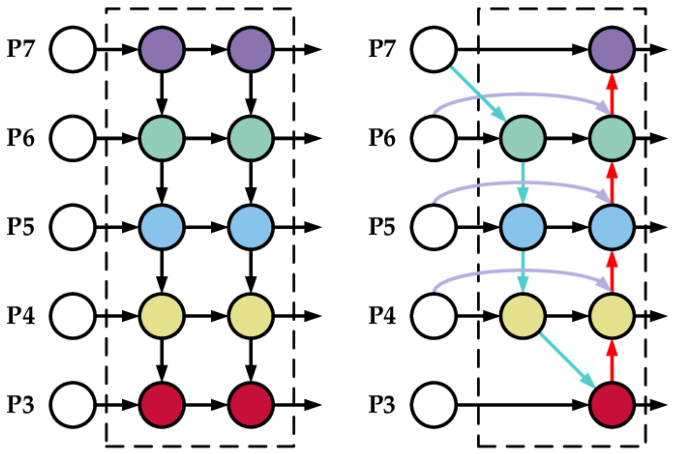


Figura 5.4: Estructuras de FPNs, izq. FPN, der. BiFPN

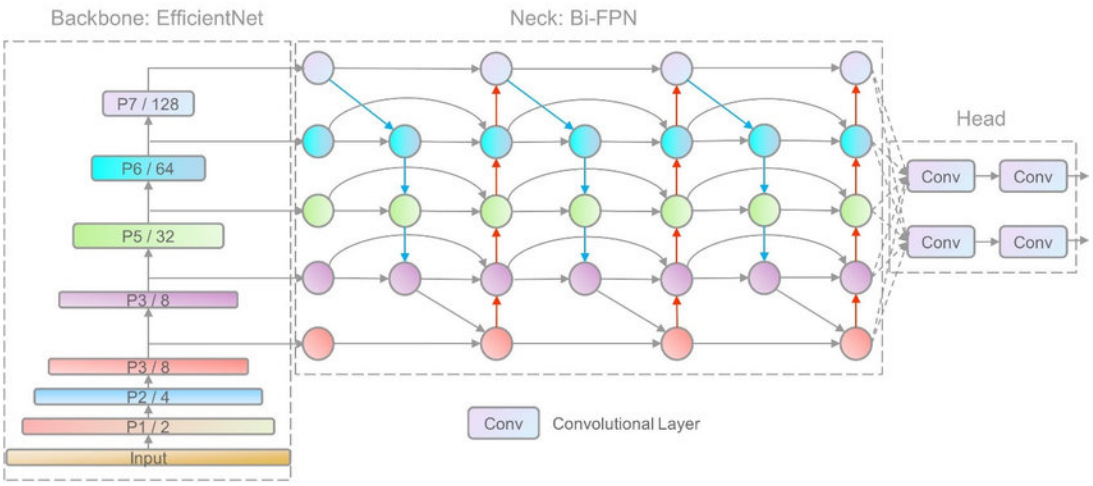


Figura 5.5: Arquitectura del modelo EfficientDet

5.2.2 Arquitectura de EfficientDet

columna vertebral

Ejemplificando lo mencionado en la sección previa, EfficientDet toma como base la arquitectura del modelo EfficientNet para extraer características a varias escalas de resolución.

Cuello

El módulo que cumple esta función en este caso es la mencionada BiFPN, que aporta una forma novedosa de organizar jerárquicamente la información de las imágenes de entrada.

Cabeza

EfficientDet, como otros modelos de arquitecturas diseñados para la detección de objetos, posee dos cabezas: una para clasificación y otra para regresión. La cabeza de clasificación utiliza una función de pérdida focal, la cual se expandirá en la siguiente sección.

5.3 RetinaNet

RetinaNet es un modelo redes de neuronas convolucionales basado en regiones, presentado por primera vez por T. Lin, P. Goyal, R. Girshick, H. Kaiming y P. Dollár en 2017 en la Conferencia Internacional sobre Visión por Ordenador (ICCV). El artículo en el que figuraba trataba de mostrar una mejora a los métodos de detección de objetos, la pérdida focal. Es un modelo denominado de “una etapa”, ya que detecta las clases en la imagen a la vez que genera las *bounding boxes*.

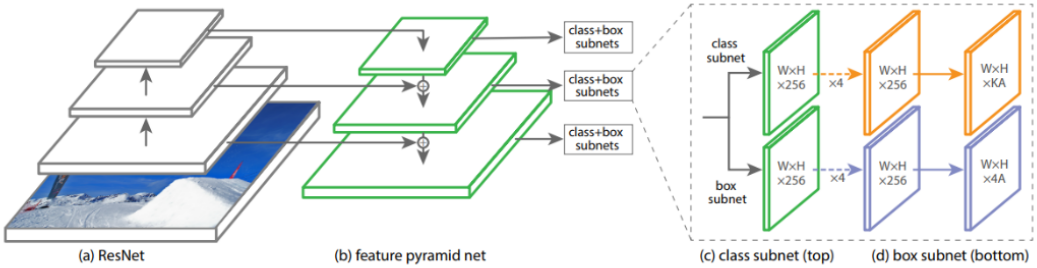


Figura 5.6: Arquitectura del modelo RetinaNet

5.3.1 Pérdida Focal

La Pérdida Focal o *Focal Loss* [20] es una función de pérdida diseñada para tener en cuenta la detección descompensada de objetos en primer y segundo plano durante el entrenamiento. La peculiaridad de esta función es la ponderación positiva de casos difíciles de discernir, centrándose más en estos a fin de forzar al modelo a aprender de casos más ambiguos en lugar de los más claros.

5.3.2 Arquitectura

La arquitectura de RetinaNet (figura 5.6) está compuesta por dos módulos, un *columna vertebral* (en general cualquier CNN sirve) que lleva a cabo la extracción de características, y dos subredes para tareas específicas. Estas subredes llevan a cabo la convolución de las clases identificadas y de las cajas/*bounding boxes* generadas.

Columna vertebral

La arquitectura de esta red no impone una columna vertebral específica, ya que sirve cualquier otra CNN, siendo uno de los más populares ResNet.

Cuello

Como módulo de procesamiento intermedio, RetinaNet emplea una FPN basada en conexiones laterales de las características de alto y bajo nivel.

Cabeza

Como se ha expuesto en esta sección, la cabeza de clasificación implementa la pérdida focal para producir las probabilidades de cada clase detectada. En el caso de la cabeza de regresión, esta genera las *bounding boxes* mediante un mecanismo basado en puntos de anclaje.

5.4 SSD: Single Shot Multibox Detector

Single Shot Detector, por sus siglas y en adelante, SSD, es un marco (*framework*) de detección de objetos, a la vez que un modelo de CNN que lo implementa. El concepto de SSD estuvo desarrollado por desarrolladores de Google, siendo publicado por Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu y Alexander C. Berg en el artículo "SSD: Single Shot MultiBox Detector" [21].

El enfoque de esta red (figura 5.7) se basa en producir un número fijo de cajas, además de sus porcentajes de confianza a cada instancia identificada en la imagen, para luego reducir estos resultados mediante una Supresión de No Máximos (NMS).

5.4.1 Puntos claves del modelo

- Mapas de características a varias escalas. La red incluye capas convolucionales de tamaño decreciente a lo largo de su arquitectura para permitir predicciones a varias escalas. El modelo de predicción de cada capa es diferente, permitiendo adaptar las funciones que mejor funcionen con cada resolución.

- Predicciones en un paso. Los objetos se detectan en un solo paso "hacia delante" (*forward*), haciendo la red más eficiente que otros modelos que pasan por una fase de propuesta de regiones.

- Cajas y relaciones de aspecto por defecto. Debido a que la red genera cajas para localizar cada objeto en cada uno de sus niveles, estas se corresponden con tamaños predefinidos, las cuales se ajustan posteriormente.

5.4.2 Arquitectura de SSD

Como la mayoría de los modelos propuestos, SSD permite el uso de cualquier red como columna vertebral, concentrando sus puntos fuertes en la cabeza.

Columna vertebral

El *columna vertebral* empleado por SSD puede ser cualquiera de los modelos usualmente empleados como base, siendo los más populares VGG y MobileNet en específico.

Cuello

SSD no posee un modelo de cuello complejo como tal, estando compuesto por una serie de capas convolucionales adicionales.

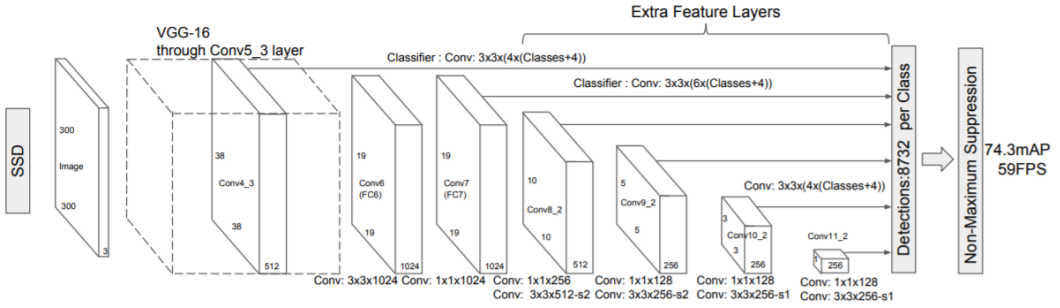


Figura 5.7: Arquitectura del modelo SSD

Cabeza

Del mismo modo que otro modelo de CNN, SSD cuenta con una cabeza de clasificación y otra de regresión. La peculiaridad de esta red reside en que las predicciones se realizan a varias escalas de forma simultánea, identificando los objetos de menor tamaño en las primeras capas y los de mayor tamaño en las últimas. Las confianzas en las *bounding boxes* se calculan usando la función *softmax* y sus localizaciones mediante *L1Loss*.

5.5 Faster R-CNN

Faster Region-based Convolutional Neural Network (figura 5.8), resumido como Faster-RCNN, es un novedoso modelo de red basada en el concepto de R-CNN o red de neuronas convolucional basada en regiones, introduciendo notables mejoras en relación a modelos anteriores. Fue presentado en el artículo de 2015 "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks" [22] de la mano de Shaoqing Ren, Kaiming He, Ross Girshick y Jian Sun. Se ha convertido en un *benchmark* de la detección de objetos, alcanzando resultados del estado del arte tanto en precisión como en rapidez.

5.5.1 Puntos claves del modelo

- Redes de propuesta de regiones. Son un mecanismo por el que se generan *bounding boxes* hipotéticas al procesar una imagen, aportando un porcentaje de posibilidad de aparición de objetos a las regiones analizadas.
- Anclas invariantes a translación. Son variables del modelo empleadas para señalar un objeto de forma inequívoca en relación a la posición de este dentro de la imagen.
- Anclas multiescala y referencias de regresión. Son la técnica empleada por los modelos para adaptar la escala y forma de las *bounding boxes* durante el entrenamiento en

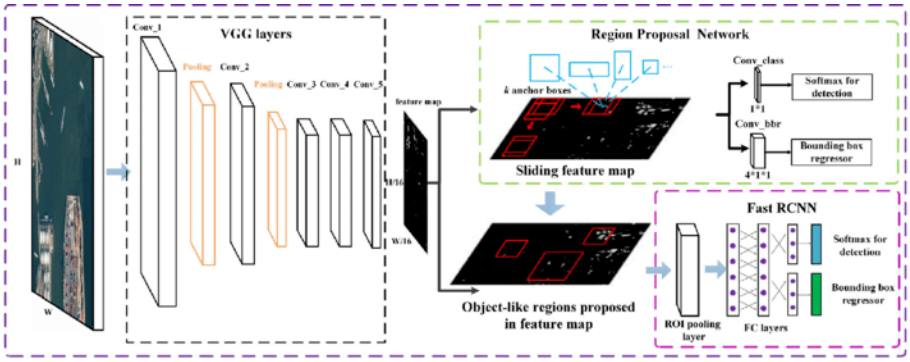


Figura 5.8: Arquitectura del modelo Faster R-CNN

relación a un punto fijo para cada objeto.

5.5.2 Arquitectura de Faster R-CNN

A diferencia de otras redes como SSD, Faster R-CNN realiza las predicciones en ”dos pasos”, esto es, mediante un módulo de propuesta de regiones donde es más posible encontrar objetos y posteriormente generando elaborando las detecciones de los mismos en base a eso.

Columna vertebral

De la misma forma que la mayoría de estas redes, la columna vertebral suele ser otra CNN preentrenada con un conjunto de datos suficientemente amplio (ResNet, VGG, MobileNet, ...).

Cuello

Como elemento principal de este módulo se encuentra una red de características piramidal, ver figura 5.3. En este caso estando integrada con una red de propuesta de regiones, como se puede apreciar en la figura 5.8. Esta segunda red trabaja con las características identificadas por la FPN para, mediante una serie de capas convolucionales, obtener en qué zonas de la imagen es más probable encontrar objetos.

Cabeza

Del mismo modo que los otros modelos, esta red compone su cabeza de una propia para la tarea de clasificación y otra para la tarea de regresión, estando compuestas principalmente por capas convolucionales y capas densas.

5.6 FCOS

FCOS es un modelo de detección de objetos propuesto por T. Zhi, S.Chunhua, C. Hao y H. Tong en su artículo de 2019 “FCOS: Fully Convolutional One-Stage Object Detection” [23]. Este artículo propone una arquitectura de CNNs basada en la detección de objetos “en una etapa” sin la necesidad de generación de anclas para localizar cada objeto.

5.6.1 Puntos claves del modelo

- Arquitectura completamente convolucional, lo que permite al modelo procesar la imagen
- Predicción multinivel con FPN [24]. Método por el que se extraen características de la imagen a varios niveles de resolución, a fin de poder identificar mejor los objetos teniendo en cuenta su variabilidad en escala.
- Enfoque sin anclas. El modelo rechaza el paradigma de asignar anclas a cada objeto, generando *bounding boxes*, coeficientes de clase y de objetividad a cada píxel de la imagen procesada.
- Centralidad. FCOS emplea una estrategia para eliminar las *bounding boxes* detectadas de baja calidad a través de una capa paralela a la de clasificación y la de regresión, para localizar mejor la ubicación de cada objeto.

5.6.2 Arquitectura de FCOS

Columna vertebral

La función de la columna vertebral en esta arquitectura, igual que las anteriores redes, puede estar desempeñada por otra CNN preentrenada.

Cuello

Para el módulo del ”cuello” FCOS emplea una FPN5.3 para generar un mapa de características, refinando la información obtenida por la columna vertebral.

Cabeza

El módulo de la ”cabeza” de esta red está compuesto por tres cabezas: una de clasificación para predecir la clase del objeto, una de regresión para predecir la posición del objeto y una de centralidad para predecir la proximidad de una posición al centro del objeto.

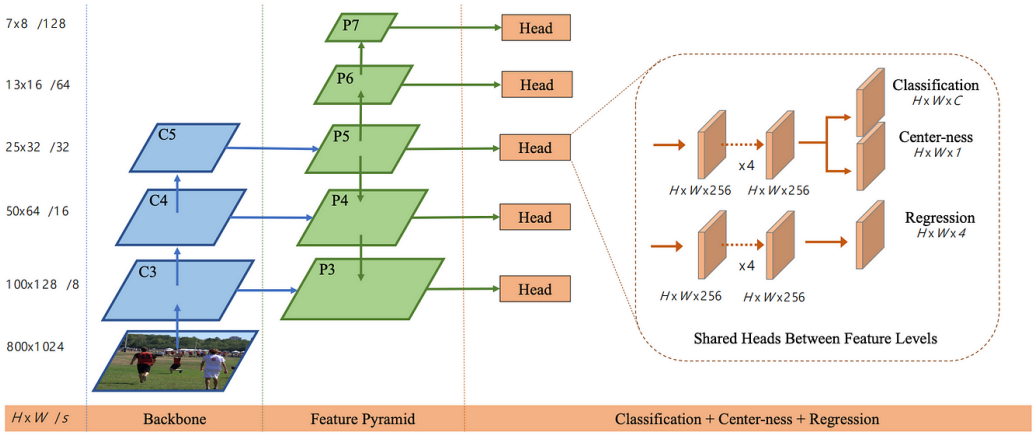


Figura 5.9: Arquitectura del modelo FCOS

5.7 Proceso de entrenamiento

En esta sección se comenta el procedimiento que se sigue para hacer que un modelo de red aprenda en base a un conjunto de datos, abordando las partes de este proceso.

5.7.1 Organización de los datos

Como fase preliminar en la que se concentran los datos a partir de los cuales se pretende que la red aprenda. Generalmente en esta fase se divide el conjunto de datos en entrenamiento, validación y pruebas, además de ajustar los datos a los parámetros de entrada del modelo.

5.7.2 Carga del modelo

En esta fase se especifica la arquitectura del modelo a usar, cargando versiones específicas o pesos preentrenados. Aparte de los valores por defecto del modelo, se establecen parámetros importantes como el ratio de aprendizaje o el número de clases. Aquí se define también el comportamiento de la red ante una entrada (*forward*).

5.7.3 Función de pérdida

Es el mecanismo por el cual la red es capaz de medir cuánto se alejan las predicciones de los valores reales. Para los casos de detección de objetos, normalmente se calcula como la suma del error de clasificación del objeto y el de regresión de sus *bounding boxes*. Es crucial, ya que estipula con qué severidad el modelo tratará de corregirse en base a su predicción.

5.7.4 Optimizador

Módulo en el que se establece el método de optimización que minimice la función de pérdida, teniendo en cuenta factores como el ratio de aprendizaje del modelo, el decaimiento de los pesos, ….

5.7.5 Ciclo de entrenamiento

Es el proceso principal del aprendizaje, en él se pasan los lotes de información al modelo y se calcula la precisión de sus predicciones en base a los valores de verdad. Esta secuencia de procesado de datos, cálculo de pérdida y ajuste del modelo se denomina paso.

Cuando la red ha procesado todos los datos disponibles dedicados a entrenamiento, se dice que ha concluido una época y por lo general se relaciona un número alto de épocas de entrenamiento como un proceso más exhaustivo de aprendizaje.

5.7.6 Evaluación/validación

Mecanismo por el cual se comprueba la precisión del modelo sobre un conjunto de datos desconocido.

Es común que dentro del ciclo de entrenamiento se incluya una fase de validación posterior a cada época para comprobar que el modelo no se está sobreajustando, es decir, aprendiendo sobre datos ya conocidos perdiendo la capacidad de generalización.

Capítulo 6

Diseño e implementación

En este capítulo se tratan los detalles del diseño del sistema de forma general, abordando sus componentes principales. Esto es, el conjunto de datos tomado como referencia, las arquitecturas de CNNs implementadas y la integración de ambas cosas con mejoras y adaptaciones al contexto del problema.

6.1 Conjunto de datos relacionados

En esta sección abordamos los aspectos principales del conjunto de datos que empleamos para describir el entorno en el que teóricamente se pondría a pruebas el sistema. Así, analizamos las otras opciones barajadas, el conjunto que finalmente usamos y las modificaciones que añadimos.

Para comenzar a diseñar el sistema fue preciso tomar como referencia un conjunto de datos a modo de modelar con él el problema que tratábamos de tratar. Con este fin, tuvimos en consideración varios conjuntos, llevando a cabo un proceso de selección con un objetivo relativamente simple: las imágenes deberían ajustarse al contexto de un entorno real en el que el sistema se pudiera emplear.

6.1.1 Laboro Tomato

Laboro Tomato [25] es un conjunto de datos conformado por imágenes de tomates en diferentes estados de maduración, además de dos tamaños de tomate diferentes, empleado tanto para detección como para segmentación.

La propia colección reside en un proyecto con un modelo asociado, entrenado con el mismo. Este se muestra relativamente fiable a la hora de clasificar semánticamente los tomates, a pesar de carecer de ningún ajuste fino, sirviendo básicamente como prueba de concepto de una posible aproximación a la detección de tomates.

Figura 6.1: Imágenes del dataset de *Tomatoes Detection*

El principal detalle que nos hizo descartar este conjunto ha sido que las fotografías de los tomates es que parecen estar hechas a racimos individuales (ver figura 6.1), fijándose a veces en uno o dos tomates solamente, en general con unos niveles de luz que favorecen notablemente la identificación de cada uno, sin apenas oclusión por las hojas. Asumimos que esto es necesario para que la segmentación sea viable, pero va en contra de la visión que tenemos asociada al proyecto, por tanto, no lo tuvimos en cuenta.

6.1.2 TomatOD

Similar al ejemplo anterior, TomatOD [26] es el conjunto de datos reunido para entrenar el modelo homónimo [27] con imágenes tomadas en un invernadero. Está conformado por 277 imágenes con 2418 anotaciones, representando tomates en tres grados de maduración, generalmente de poco tamaño. En la figura 6.2 podemos ver algunos ejemplos.

El repositorio que lo aloja recoge información estadística sobre la distribución de tamaños de los objetos, números de categorías por imagen e instancias anotadas por cada imagen. Esto facilita el proceso de comparación con los conjuntos que tomamos como referencia.

Pese a que las fotos se aproximaban a la idea inicial que teníamos para modelar el contexto del problema, decidimos descartar el conjunto debido a que las imágenes se encontraban editadas/preprocesadas (ver por ejemplo la figura 6.3) para, por lo que asumimos, ocultar artificialmente algunos tomates.

6.1.3 Tomatoes Detection Computer Vision Project

Dataset conformado por 1790 imágenes (948 antes de ser aumentadas), residente en Roboflow de forma pública. El conjunto de datos contiene imágenes de tomates generalmente verdes y rojos, con gran detalle de cada tomate de manera individual. De igual forma que

Figura 6.2: Imágenes del conjunto de datos de TomatOD

Figura 6.3: Detalle de imagen del conjunto de datos editada.

Figura 6.4: Imágenes del dataset de Laboro Tomato

Laboro Tomato, está asociado a un modelo entrenado con el propio dataset, disponible a su vez que el conjunto en Roboflow. Si bien la iluminación varía relativamente a lo largo de las imágenes, esto podemos asumir que se debe a que fueron tomadas en dos puntos distintos del día, estando algunas sujetas a luz natural y otras a luz artificial.

Esto constituye uno de los dos motivos que nos hizo descartar el conjunto de imágenes. El otro es el enfoque puesto en cada tomate, a veces ocupando un solo objeto toda la imagen, eliminando la posibilidad de oclusión, a la vez de poder detectar varios tomates en una imagen (figura 6.4).

6.1.4 RpiTomato Dataset.

RpiTomato Dataset [28] es un conjunto de imágenes anotadas tomadas en un invernadero con un mecanismo similar al que tomamos de referencia. Las fotos están sacadas a una resolución de 4056x3040 píxeles y las anotaciones están en formato Pascal VOC, hechas manualmente en base solo a la clase ”tomate”, pese a que se contengan imágenes de tomates en distintos grados de maduración. Pese a la alta calidad de las imágenes y un formato igual al conjunto que tomamos como referencia, decidimos descartar esta colección debido a que las imágenes se centran en racimos pequeños de tomates, siendo a veces imágenes individuales.


Figura 6.5: Imágenes del conjunto RpiTomato Dataset

6.2 Conjunto de datos final

Eventualmente, nos terminamos inclinando por usar el dataset en el que se basan los artículos [29] y [30] debido a que se adhería al contexto en el que veíamos desplegado nuestro sistema. Las principales características de las imágenes de este conjunto de datos son las siguientes:

- Condiciones de luz irregulares. Las imágenes deben reflejar la vista de un robot que transcurra el invernadero sin poder enfocarse en cada rama.
- Alto número de tomates por foto en general (esto nos sirve para comprobar la aplicabilidad de la umbralización), aparte de distribución irregular, permitiendo objetos en varios planos.
- Ángulo de la cámara único ajustado a la altura del robot que tomó las fotografías.
- Oclusión frecuente de tomates por hojas de la planta, típico de un invernadero real.
- Cierto solapamiento entre las imágenes, aunque sin llegar a ser un vídeo.

Si bien ciertos detalles como la inclusión de tomates en varios estados de maduración y aspectos del etiquetado (mejorable, comentar más adelante como posible parte del sistema) son algo negativos, no los consideramos suficientes como para descartar el conjunto.

6.2.1 Descripción del conjunto de datos

En este apartado comentaremos los detalles más relevantes del dataset empleado en este proyecto, tanto a nivel de imágenes como de anotaciones.


Figura 6.6: Ejemplos de imágenes con condiciones de iluminación variables.


Figura 6.7: Ejemplos de distribución irregular de tomates y varios grados de maduración.

Imágenes

Las imágenes que conforman la colección fueron tomadas por una cámara estéreo ZED articulada sobre la plataforma de un robot que recorría los pasillos del invernadero. Este proceso hace que cada fotografía se haya tomado con un ángulo que influye el enfoque de cada objeto, no siendo fotografías completamente frontales centradas en cada tomate. Además de esto, las imágenes se han tomado con una relación de disparo tal que produce cierta superposición, lo cual nos podría ayudar en un futuro a usar como referencia para adaptar el sistema a secuencias de vídeo.

El conjunto contiene dos listas de imágenes: una con las fotos recortadas y otra a tamaño completo. Para nuestro sistema emplearemos las fotos a tamaño completo, ya que la mayor parte de los modelos no tienen restricción de tamaño de entrada. Las fotos están en formato JPG, con unas dimensiones de 1280 por 720 píxeles, una resolución vertical y horizontal de 96 puntos por pulgada y una profundidad de 24 bit.

Cabe destacar que las fotografías captadas reflejan fielmente la visión que tendría un robot que tuviera que navegar automáticamente un invernadero, esto es, las condiciones de iluminación varían drásticamente en varios sectores 6.6, la cantidad de objetos por imagen no es

Figura 6.8: Ejemplo de una imagen del conjunto anotada.

constante y estos están, en la mayoría de casos, tapados, total o parcialmente, por hojas, lo cual dificulta su identificación 6.7.

Anotaciones

Cada archivo con anotaciones se relaciona con cada imagen, estando en formato xml. Estas anotaciones contienen el nombre del objeto (*tomato*), las esquinas de las *bounding boxes*, en formato Pascal VOC, aparte de datos adicionales que no tenemos en cuenta como si el objeto está segmentado u ocluido.

A partir de esto, se diseñó un módulo que tomaría la estructura del conjunto de datos de carpetas separadas para imágenes, anotaciones y conjuntos de estas para generar ficheros en formato csv por cada subconjunto (entrenamiento, validación y pruebas). De esta forma se simplificaba el hecho de identificar las imágenes de cada conjunto y las anotaciones de cada una, leyéndolas directamente de una tabla.

Como se puede apreciar en la imagen de ejemplo 6.8, la selección de objetos anotados en la imagen no es exhaustiva, ya que muchos tomates podrían ser detectados siendo penalizados por no formar parte de los *ground truths*. Este aspecto lo comentaremos más en detalle posteriormente, ya que consideramos que es un factor importante.

6.2.2 Adaptación del dataset a PyTorch

Para poder trabajar con este conjunto de datos dentro del marco de PyTorch, y posteriormente de PyTorch Lightning, hemos tenido que tener en consideración ciertos aspectos de los datos. Entre estos, registrar las dimensiones de las imágenes, la escala a la que se redimensionan en el caso de darse esta transformación, o el área descrita por cada bounding box, a

Figura 6.9: Error de clasificación de un modelo entrenado con conjuntos con distintas distribuciones.

efectos de saber qué tamaño de bounding boxes es el óptimo para el sistema.

6.2.3 División del dataset

A la hora de realizar una división del conjunto de datos en entrenamiento, validación y test tuvimos en consideración la escasez de datos (449 imágenes), por lo que vimos más conveniente darle más consideración al entrenamiento. De este modo, teniendo como referencia las divisiones de 40/20/40, 50/20/30, 60/20/20, 70/15/15 y 80/10/10, probamos a entrenar uno de los modelos para comprobar si existía algún tipo de sesgo estadístico que produjera resultados distintos con una distribución.

Para comprobar si existía alguna posibilidad de que la distribución elegida estuviera sesgada estadísticamente, se realizó una serie de entrenamientos de una de las redes propuestas con varias distribuciones. En base a los resultados obtenidos 6.9 consideramos probado que la diferencia es desestimable, por lo que decidimos emplear la que dedica un 80% al entrenamiento, un 10% a validación y un 10% a pruebas, con el fin de aprovechar al máximo las pocas imágenes para el aprendizaje de la red.

6.2.4 Incorporación de otras imágenes

Debido al reducido tamaño del dataset, al principio de la etapa de desarrollo del proyecto, contemplamos la posibilidad de incorporar a nuestro conjunto de datos imágenes de otros conjuntos, a modo de dotar de cierta capacidad de abstracción al modelo. Descartamos esta

Figura 6.10: Ejemplo de imágenes transformadas.

opción por varios motivos, ya que no solo iba en contra de la visión detrás del sistema de que el modelo aprenda de forma relativamente específica, sino que el tiempo necesario para recabar el resto de datos para luego adaptarlos a nuestro sistema nos hizo desechar esta idea por completo.

6.2.5 Transformaciones (*Augmentation*)

Debido al limitado número de imágenes de nuestro conjunto de datos, hemos incluido una serie de transformaciones posibles en cada época de entrenamiento para cada imagen. La finalidad de esto es intentar aprovechar al máximo los datos disponibles, haciendo que los modelos aprendan en una variedad de contextos mayor, mitigando el problema del sobreajuste. Como se ha mencionado en el tercer capítulo 3, la librería empleada para esto ha sido Albumentations, y las transformaciones llevadas a cabo son volteo vertical, escalado aleatorio y desenfoque, aparte de normalización para poder convertir las imágenes en tensores.

6.3 Diseño del sistema

En esta sección se describen las funciones diseñadas para adaptar las implementaciones de redes encontradas a nuestro conjunto de datos, así como para entrenarlas y evaluarlas.

6.3.1 Módulos con Lightning

Como se mencionó en el tercer capítulo 3 la librería más importante que empleamos en este proyecto, aparte de PyTorch, es Lightning. Gracias a ella muchos detalles de bajo nivel del uso de redes de aprendizaje profundo se obvian. Esto supone una gran ventaja a la hora de trabajar con estas redes, ya que la comprensión del funcionamiento del sistema en sí es más sencilla. Esto no es solo por su legibilidad, sino porque el número de líneas de código necesarias para implementar un módulo Lightning se reduce notablemente.

Como se puede apreciar en el extracto de código 6.1, para poder entrenar un modelo mediante un módulo Lightning solo hace falta especificar cómo cargar el modelo, establecer los hiperparámetros y el optimizador, y finalmente describir los pasos de entrenamiento y validación.

Por sencillez a la hora de trabajar con los modelos, los pasos de entrenamiento y validación se han simplificado al máximo, registrando el error producido en el paso de entrenamiento y guardando las detecciones en el paso de validación para promediarlas posteriormente, aparte de registrar su error.

6.3.2 Función de pérdida

Debido a que la implementación del modelo de EfficientDet que usamos generaba su propia pérdida y el resto de modelos prescindían de este mecanismo, diseñamos una función de pérdida simple para poder evaluar todos los modelos igualmente. Esta función está basada en el Índice de Jaccard, centrándonos solo en las predicciones de clase y localización mediante cajas. Esta función es la suma dos cálculos, uno de pérdida de clasificación y otro de regresión.

Para el cálculo del error de regresión, tomamos las *bounding boxes* generadas por el modelo y aplicamos el Índice de Jaccard/IoU para hallar las mejores correspondencias con las *bounding boxes* anotadas y reducir ambos conjuntos. Estos dos conjuntos con las mismas dimensiones sirven de argumentos para la función que computa la pérdida de forma completa. Esta función tiene en cuenta el área de superposición, la normalización de la distancia al punto central y la relación de aspecto, implementando la fórmula propuesta en el artículo "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression" [7].

Para el cálculo de la clasificación tomamos la reducción de las clases identificadas y computamos su error con la función de Entropía Cruzada [31] por simplicidad, ya que solo contemplamos una clase. El error final para una predicción es la suma de ambos errores de clasificación y regresión para cada detección.

```python
class RetinaNetLightning(LightningModule):
    def __init__(self,
        num_classes=config.NUM_CLASSES,
        lr=config.LR,
        threshold=0.2, #Threshold de scores de las bounding boxes
        iou_thr=config.IOU_THR): #Threshold de solapamiento de cajas
        super().__init__()
        self.lr = lr
        self.model = retinanet_resnet50_fpn(
            weights = RetinaNet_ResNet50_FPN_Weights.DEFAULT,
            weights_backbone = ResNet50_Weights.IMAGENET1K_V1)
        self.threshold = threshold
        self.iou_thr = iou_thr
        self.loss_fn = compute_loss
        self.mean_ap = MeanAveragePrecision()
        self.val_step_outputs = []
        self.val_step_targets = []

    def forward(self, images : torch.Tensor, targets=None):
        outputs = self.model(images, targets)
        if not self.model.training or targets is None:
            outputs = threshold_fusion(outputs, images_sizes(images),
                    iou_thr=self.iou_thr, skip_box_thr=self.threshold)
        return outputs

    def configure_optimizers(self):
        return torch.optim.AdamW(self.model.parameters(), lr=self.lr)

    def training_step(self, batch, batch_idx):
        images, targets, ids = batch
        loss = self.forward(images, targets)
        self.log('train_class_loss', loss['classification'].detach())
        self.log('train_box_loss', loss['bbox_regression'].detach())
        return loss['classification'] + loss['bbox_regression']

    @torch.no_grad()
    def validation_step(self, batch, batch_idx):
        images, targets, ids = batch
        outputs = self.forward(images, targets)
        loss = self.loss_fn(outputs, targets)
        mean_ap = self.mean_ap(outputs, targets)
        self.log('val_class_loss', loss['class'])
        self.log('val_box_loss', loss['box'])
        self.val_step_outputs.extend(outputs)
        self.val_step_targets.extend(targets)
        return loss['total']

    @torch.no_grad()
    def test_step(self, batch, batch_idx):
        images, targets, ids = batch
        outputs = self.forward(images, targets)
        loss = self.loss_fn(outputs, targets)
        self.log('test_class_loss', loss['class'])
        self.log('test_box_loss', loss['box'])
        return loss['total']

    def on_validation_epoch_end(self):
        val_all_outputs = self.val_step_outputs
        val_all_targets = self.val_step_targets
        mean_ap = self.mean_ap(val_all_outputs, val_all_targets)
        for k in config.KEYS:
            self.log("val_"+k, mean_ap[k], logger=True)
        self.val_step_outputs.clear()
        self.val_step_targets.clear()
```

Listing 6.1: Ejemplo de módulo implementado con Lightning.

6.3.3 Integración de los modelos

Una vez descrito el comportamiento básico de los modelos, el siguiente paso fue adaptar su uso a la librería (PyTorch) *Lightning*. Como se explicó en 3, esto nos evita las partes más tediosas y repetitivas del entrenamiento y evaluación. Esto nos permite simplemente definir el comportamiento del modelo en las etapas de entrenamiento y validación. Con esto, solo faltaba definir los hiperparámetros para el entrenamiento y usar el módulo propio de *Lightning* para entrenar cada modelo, registrando las pérdidas en las etapas de entrenamiento y validación.

6.3.4 Módulo de división del conjunto de datos

Para generar versiones del conjunto de datos con distintas distribuciones para entrenamiento, validación y pruebas se diseñó un módulo que se encargara de esta funcionalidad. Aparte de otras funciones auxiliares este módulo contiene dos funciones principales: una para para separar las anotaciones según las divisiones especificadas(80/10/10, 70/15/15, ...) y otra para convertir los ficheros en formato xml en un *DataFrame*. Con la finalidad de poder probar los modelos entrenados con cada división específica de los conjuntos de datos la función de separación tiene en cuenta que las imágenes de prueba sean comunes a todas las divisiones.

6.4 Umbralización

En esta parte se abordan los mecanismos integrados en el sistema para llevar a cabo un proceso de umbralización.

Como base del proceso de umbralización contemplamos dos umbrales: uno en base a la confianza del modelo para una predicción y otro para considerar dos predicciones como posiblemente la misma.

Con esto, la función de fusión ponderada de cajas citada en el capítulo 3 nos permite integrar ambos umbrales. Así, la función recibe las predicciones generadas por el modelo y ambos valores y en base a estos une las cajas muy solapadas, promediando la confianza de las cajas fusionadas.

De esta forma, con el umbral de solapamiento reducimos el número de predicciones finales y con el umbral de confianza provocamos que la salida solo refleje las predicciones con mejor puntuación.

Este proceso de división de los resultados en base a su índice de confianza tiene una alta variabilidad en sus resultados para los casos de modelos entrenados durante un número reducido de épocas o mediante la técnica de aprendizaje por transferencia. Así, un filtrado de los resultados con un umbral único para todas las predicciones se vuelve trivial ya que la red es capaz de discernir con una alta fiabilidad los casos correctos.

Aparte del enfoque del umbral único tuvimos en consideración integrar algún mecanismo para que los datos de predicción se filtraran de forma dinámica en base a la entrada. La primera opción barajada fue incluir como parte de la red un parámetro entrenable que se ajustase en cada época de entrenamiento, a modo de umbral que el propio modelo fuese cambiando en base a la experiencia. Esta opción fue descartada ya que los resultados no eran satisfactorios, al menos mejores que los producidos por un umbral único.

Otra opción fue la de intentar cribar los resultados en base a un promedio de los tamaños de las cajas de predicción, pero se desechó ya que de esta forma se reducía la capacidad individual de cada red propuesta de detectar mejor objetos dependiendo de su tamaño.

La última de las opciones tenidas en cuenta se centraba en adaptar el umbral dependiendo de características de la imagen como el brillo o ciertos canales. La implementación de esta opción se tornó demasiado costosa para la poca fiabilidad de los resultados, por lo que decidimos no incluirla.

Capítulo 7

Pruebas

En este capítulo se exponen los resultados de los experimentos llevados a cabo durante el diseño del sistema, ofreciendo un análisis de los mismos.

7.1 Inferencias sin entrenamiento

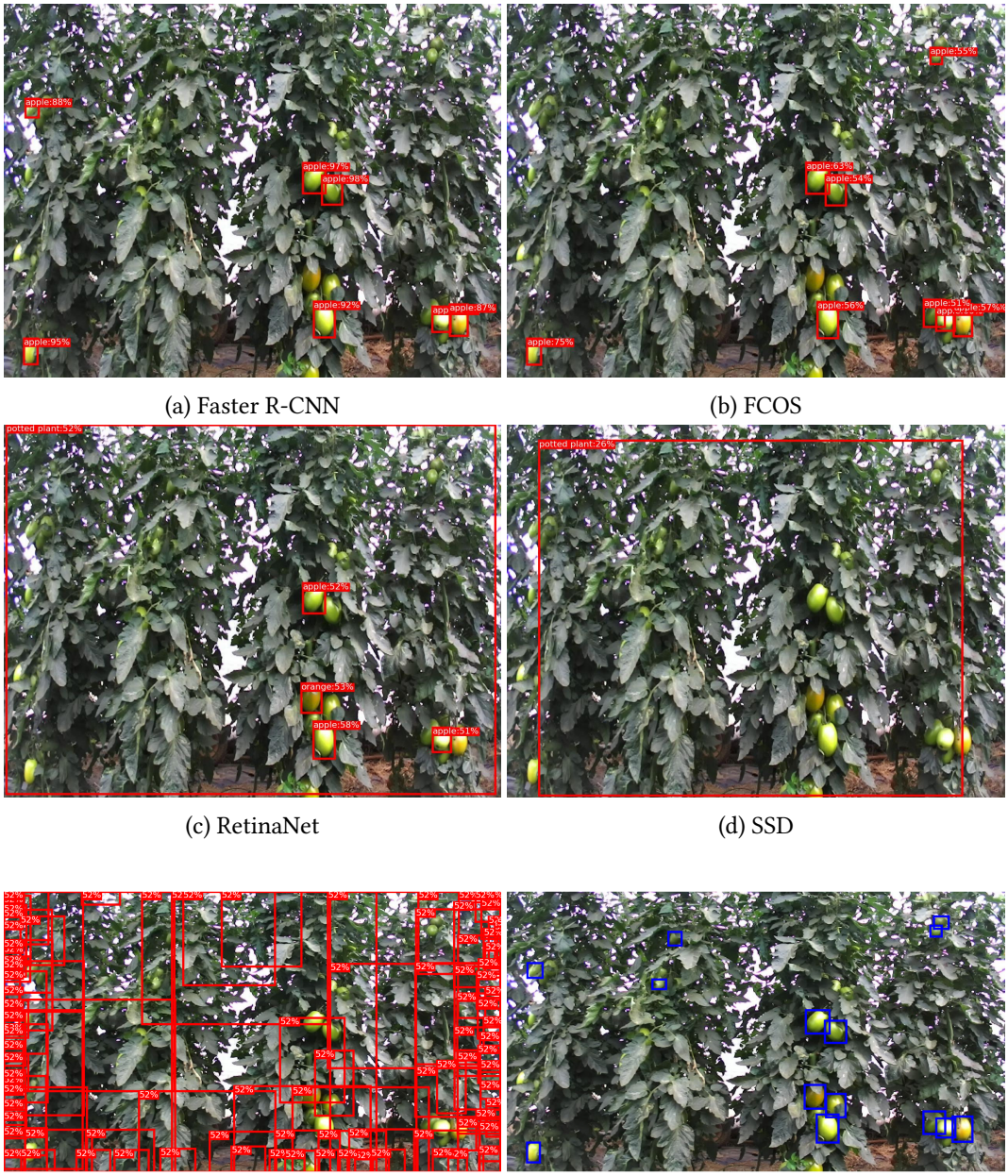
Como paso previo, y a fin de comprobar si los modelos elegidos pueden ser útiles en nuestro contexto, se realizaron pruebas simples en las que cada uno de los modelos propuestos realizaba el proceso de inferencia sobre un conjunto de imágenes de prueba (figura 7.1). En esta prueba no se alteraron el conjunto de etiquetas por defecto que identifica cada modelo, por lo que se exponen para cada objeto identificado. Este comportamiento se altera en pruebas posteriores, ya que la clase "tomate" no existe en los conjuntos de datos con los que estos modelos se han entrenado.

Por razones de legibilidad se han filtrado predicciones con un grado bajo de confianza, ya que queremos comprobar cuáles son las detecciones que cada red identifica mejor para comprobar si se podrían incluir en el sistema.

Como era esperable, las redes preentrenadas no son capaces de identificar los tomates acertadamente, ni en aspecto de clasificación ni de regresión, por lo que el siguiente paso sería entrenarlos con nuestro conjunto de datos.

7.2 Reentrenamiento completo

En esta sección se exponen los experimentos llevados a cabo adoptando como método de entrenamiento un aprendizaje de la completa, aportando gráficas en las que se muestran los parámetros de cada modelo durante el entrenamiento y ejemplos de inferencias.



(a) Faster R-CNN (b) FCOS

(c) RetinaNet (d) SSD

(e) EfficientDet (f) Imagen anotada

Figura 7.1: Inferencias de los modelos preentrenados

7.2.1 Error durante el entrenamiento

Se ha escogido un número de 50 épocas de entrenamiento para los cinco modelos elegidos como compromiso entre un entrenamiento intensivo y la posibilidad de sobreajuste.

Como se mencionó en el capítulo 4, el proceso de entrenamiento se llevó a cabo en la

máquina dedicada con un procesador superior, por lo que la mayor carga computacional ocurre en la GPU, reduciendo las operaciones en la CPU al cálculo del error de predicción y al computar las métricas de evaluación.

En general el proceso de entrenamiento sucede sin problemas, con un error decreciente al ubicar los tomates (figura 7.2), a excepción del modelo SSD. Como es comprensible la pérdida durante la validación es de un orden de magnitud mayor que la calculada durante el entrenamiento, pues esta se computa mediante una fórmula común y simple para todos los modelos. A diferencia de esta, la pérdida durante el entrenamiento es calculada de forma interna por cada red, incluyendo optimizaciones para ajustarla al funcionamiento específico de cada arquitectura.

7.2.2 Evaluación durante entrenamiento

Como se señaló en el capítulo 4, el criterio de evaluación seguido por el sistema es el *benchmark* de COCO, el cual calcula la precisión y el *recall* promediados. Además, expresa los valores de estas métricas para tres tamaños de los objetos, lo cual nos permite apreciar qué redes identifican mejor qué objetos.

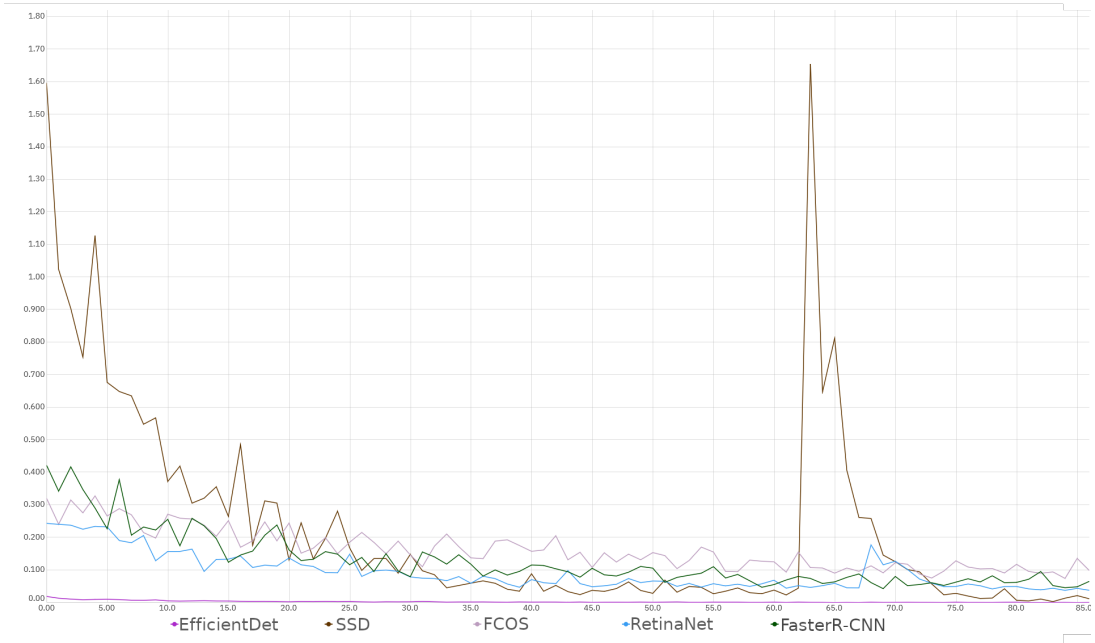
De forma simétrica a la pérdida, se observa que en general a medida que transcurren las épocas de entrenamiento las redes mejoran su precisión y *recall* (figura 7.3).

Evaluación de tiempo y consumo de recursos

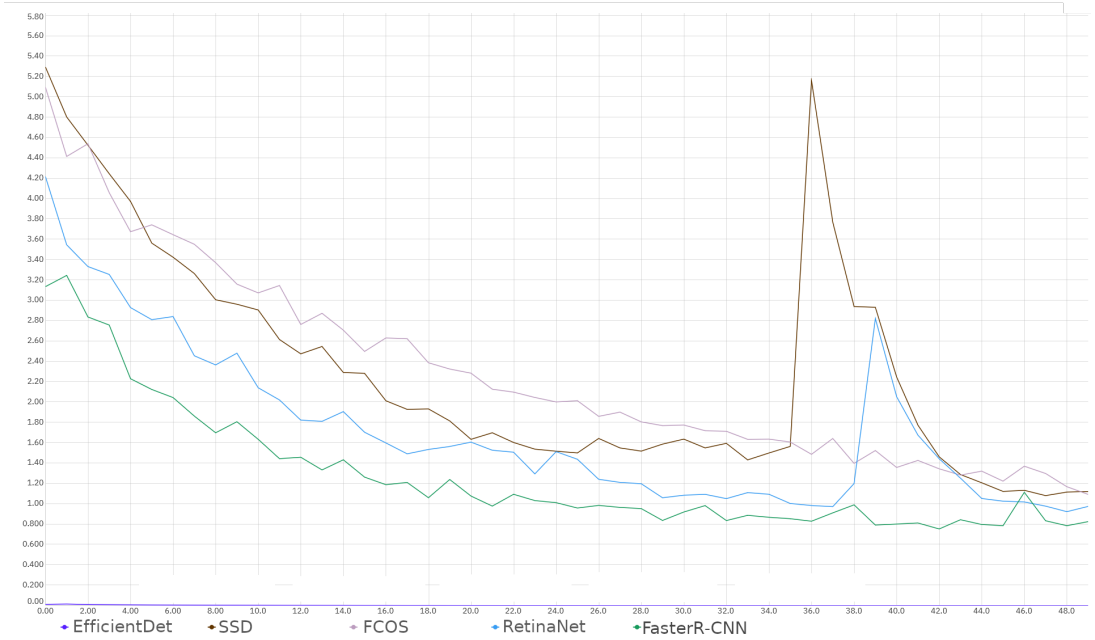
Gracias a la monitorización de las redes llevada a cabo mediante la herramienta Neptune.ai podemos observar la duración de cada entrenamiento, además de su consumo tanto de CPU como de GPU y la memoria total ocupada durante este proceso. Como se puede apreciar en la tabla 7.1 con los valores promediados, la arquitectura de EfficientDet hace honor a su nombre realizando un proceso de entrenamiento en menos tiempo, involucrando menos recursos. El segundo modelo más rápido y con menos gasto de procesador y memoria es SSD, lo cual se debe a que se trata de una arquitectura menos compleja. Los resultados de esta red, consecuentemente son peores, ya que normalmente se toma como columna vertebral de otras redes sobre la que añadir optimizaciones.

7.2.3 Entrenamiento con aprendizaje por transferencia

Como vía alternativa para tratar de optimizar tanto el proceso de entrenamiento del modelo como la capacidad de generalización, llevamos a cabo un entrenamiento de las redes mediante el aprendizaje por transferencia (capítulo 2). De esta forma, se pretendía reducir la carga computacional del entrenamiento, ya que se reducía considerablemente el número

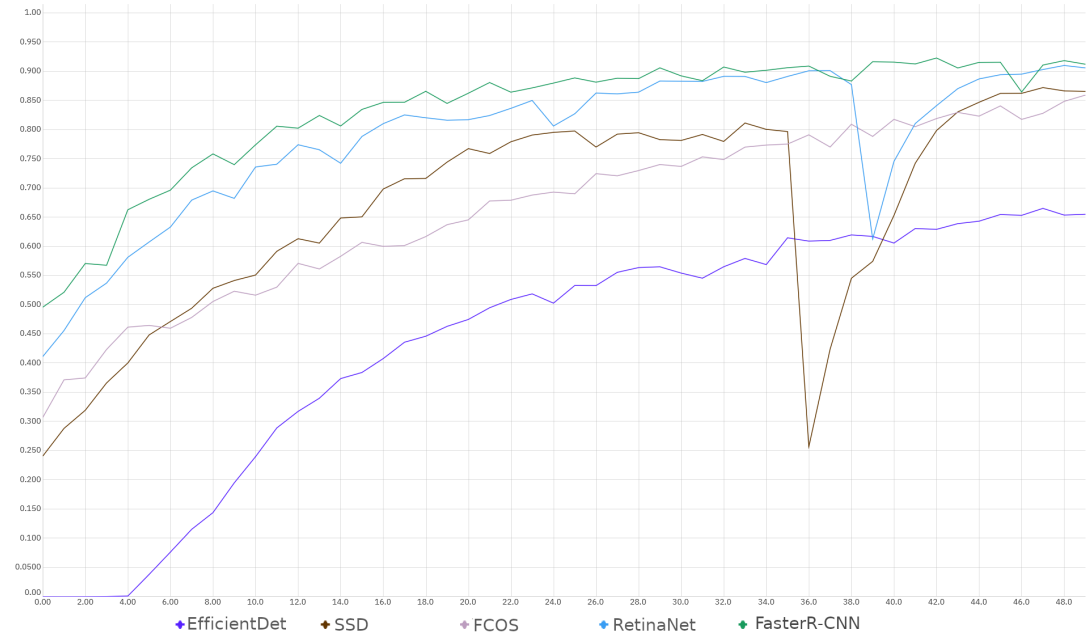


(a) Error de regresión durante el entrenamiento.

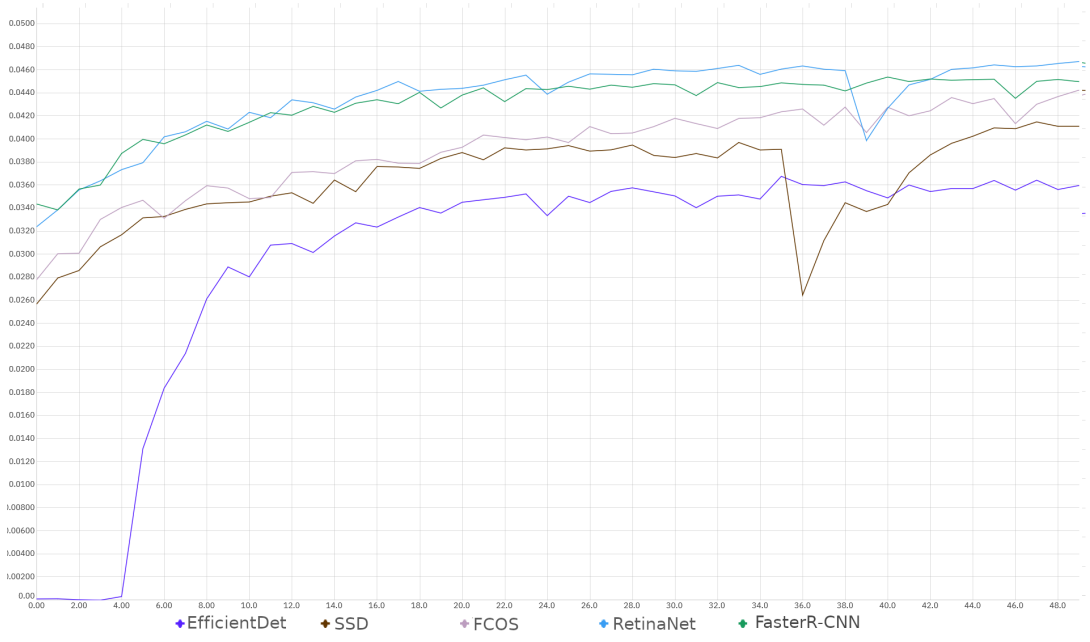


(b) Error de regresión durante la validación.

Figura 7.2: Errores de regresión de los modelos a lo largo del ciclo de entrenamiento.



(a) Precisión media de los modelos.



(b) *Recall* media de los modelos.

Figura 7.3: Métricas de evaluación durante el entrenamiento.

	Duración	CPU	GPU	Memoria
EfficientDet	33 m	6.9 %	62 %	13.2 GB
FasterR-CNN	58 m	5.5 %	96 %	51.4 GB
RetinaNet	1 h	5.0 %	73 %	25.5 GB
FCOS	41 m	4.1 %	11 %	34.4 GB
SSD	23 m	3.8 %	73 %	18.2 GB

Tabla 7.1: Duración y consumo de recursos (CPU, GPU y memoria) en el entrenamiento.

	Tamaño	Parámetros	Entrenables	No entrenables
EfficientDet	467.619 MB	116.0 M	116.0 M	492 K
FasterR-CNN	167.021 MB	41.8 M	15.0 M	26.8 M
RetinaNet	136.060 MB	34.0 M	6.7 M	27.3 M
FCOS	129.078 MB	32.3 M	4.9 M	27.3 M
SSD	142.567 MB	35.6 M	12.7 M	22.9 M

Tabla 7.2: Tamaño y número de parámetros entrenables y no entrenables de cada modelo.

de parámetros entrenables de cada modelo (tabla 7.2), descartando los propios de la columna vertebral.

Si bien los porcentajes de confianza de los modelos con aprendizaje por transferencia tienen más dispersión que los que han sido reentrenados por completo, los primeros son capaces de inferir acertadamente (figura 7.4), produciendo detecciones similares a los segundos. Como es comprensible la predicción de la red se ve disminuida7.5 ya que no adapta su mapa de características al contexto específico del conjunto de entrenamiento, sino que recicla el de su columna vertebral ya entrenada.

7.2.4 Comparación

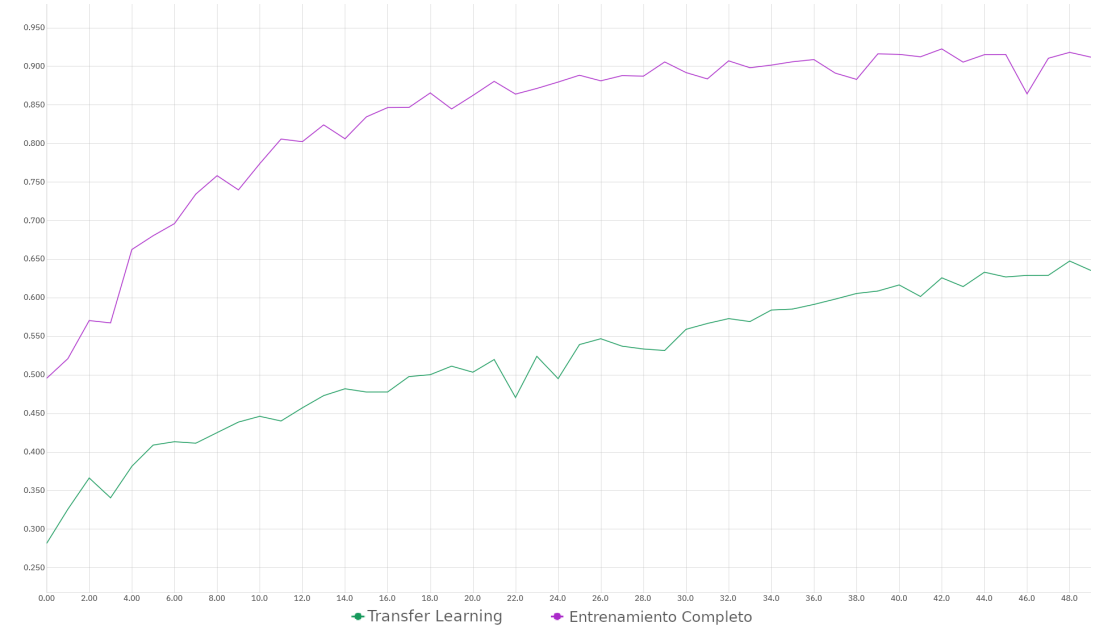
La técnica de *transfer learning* permite reducir el consumo de recursos durante el proceso de entrenamiento, aunque la diferencia entre la duración de ambas aproximaciones es mínima. En general, las redes replican el aprendizaje sin problema, consiguiendo adaptarse al nuevo contexto, no obstante la precisión se ve notablemente afectada. Esto se debe a que entrenando la forma en la que la red detecta características con la columna vertebral el modelo ajusta las predicciones más exhaustivamente.

Figura 7.4: Inferencias del modelo Faster R-CNN con entrenamiento completo (azul) y aprendizaje por transferencia (rojo).

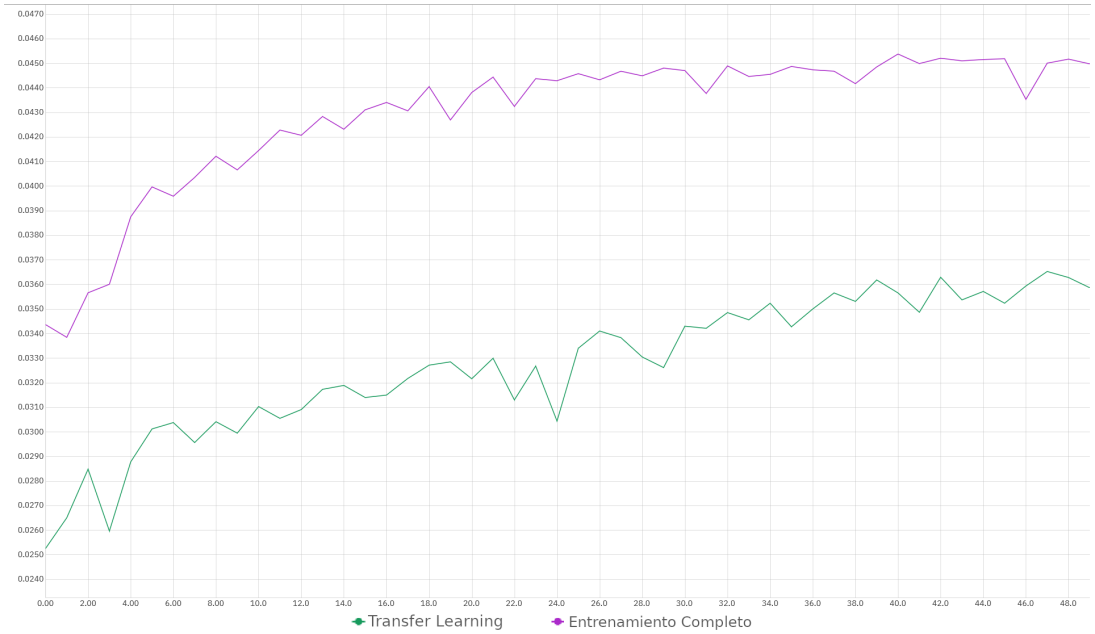
7.3 Umbralización

En esta sección se comentan los experimentos llevados a acabo para ajustar el número de detecciones de las redes, teniendo en cuenta los umbrales ajustables del sistema.

En base al mecanismo de umbralización llevamos a cabo una serie de experimentos con inferencias de los modelos para ver en qué medida cambiaban los resultados del sistema. Como observamos, sin filtrar los resultados, el modelo genera predicciones con porcentajes de confianza muy variables y tamaños dispares (figura 7.6). Si bien las predicciones empleando los dos umbrales de solapamiento y de confianza por separado producen detecciones peores, cuando se combinan ambos estos mejoran considerablemente. Debido a que una vez entrenadas las redes un mayor número de veces, estas producen detecciones con menor variabilidad, estando las más fiables en porcentajes muy altos y las menos en los más bajos. Por esto una variación de ambos umbrales a partir de 0.5 no ofrece mejoras sustanciales.

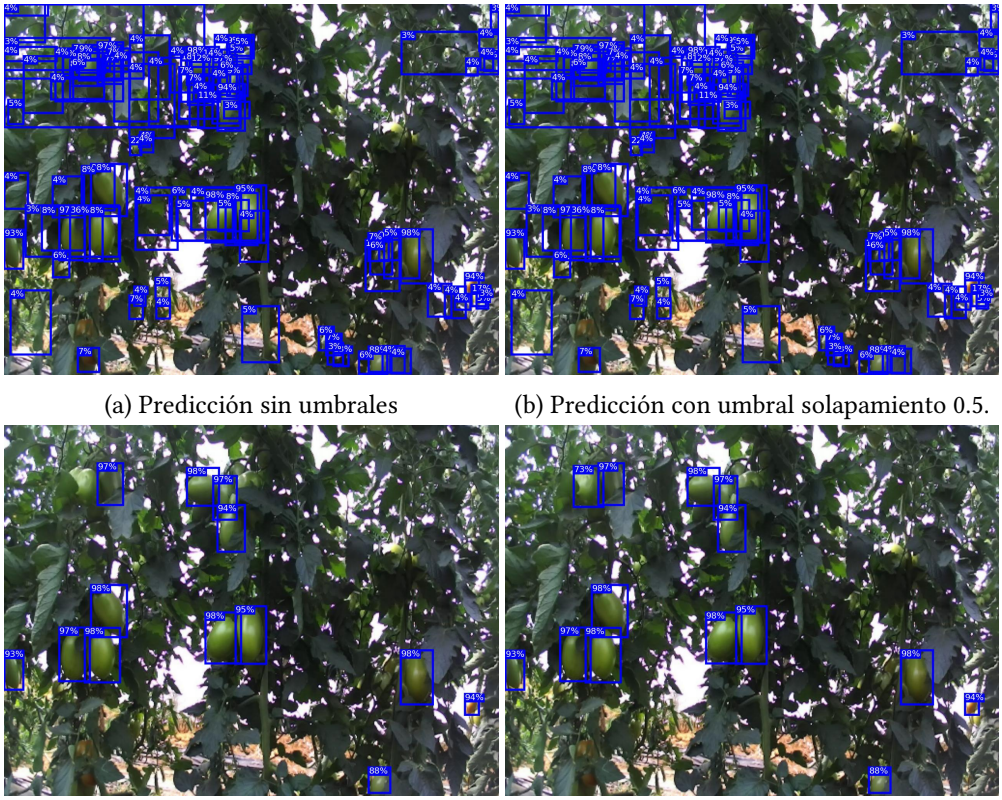


(a) Precisión media del modelo Faster R-CNN.



(b) Recall media del modelo Faster R-CNN.

Figura 7.5: Comparación de métricas con ambos enfoques de entrenamiento



(a) Predicción sin umbrales

(b) Predicción con umbral solapamiento 0.5.

(c) Predicción con umbral de confianza 0.75.

(d) Predicción con ambos umbrales a 0.5.

Figura 7.6: Comparación de resultados con umbrales de confianza y solapamiento.

Capítulo 8

Conclusiones

En este capítulo se comentan las conclusiones extraídas de los resultados del sistema, incluidas las críticas al proyecto en si, y las posibles líneas de trabajo futuro.

8.1 Conclusiones

El principal objetivo de este proyecto es estudiar cómo mejorar la aplicación de redes de neuronales convolucionales (CNNs) de última generación en la tarea de detectar tomates en condiciones reales de operación en un invernadero.

En primer lugar, y a diferencia de los *benchmarks* imperantes en el ámbito de la CNNs, nuestra investigación nos ha confirmado nuestras sospechas de que en esta aplicación hay muy pocos datasets públicos disponibles y todos ellos con muy pocas imágenes etiquetadas. Además, sólo uno de ellos refleja las condiciones normales de operación en un entorno real de trabajo, como son la distribución irregular de objetos, la oclusión por las hojas o la gran variabilidad en las condiciones de luz, textura y perspectiva de la cámara. Aunque de nuevo, el número de datos es reducido y se limita a un único invernadero y una única captura de datos.

En segundo lugar, en este trabajo hemos analizado los modelos de acceso público de las nuevas redes convolucionales para determinar cuáles se podrían adaptar mejor a nuestro problema. Después de varias pruebas, nos hemos decantado finalmente por EfficientDet, SSD, FasterR-CNN, FCOS y RetinaNet. Los cuatro últimos se encuentran en la librería Torchvision, siendo el primero una implementación pública tomada como referencia por la comunidad de PyTorch.

De las pruebas realizadas podemos extraer dos conclusiones principales. La primera es sobre la posibilidad de adaptar modelos de aprendizaje profundo de última generación, comparando su rendimiento y costes de entrenamiento. Como hemos visto en el capítulo anterior, modelos novedosos como EfficientDet, Faster R-CNN o RetinaNet son capaces de registrar re-

sultados satisfactorios, pudiendo adaptar el consumo de recursos según el enfoque de entrenamiento. Así, un entrenamiento completo de las redes propicia mejores detecciones, invirtiendo más carga de procesamiento y alojamiento de memoria, estando estos valores más reducidos cuando se aplica un aprendizaje por transferencia, teniendo un coste en la precisión de las redes.

La segunda conclusión está relacionada con el proceso de umbralización integrado en las redes del sistema. En este caso, hemos observado que un filtrado adaptativo no es necesario para la tarea si aplicamos otras técnicas. De esta forma, un entrenamiento exhaustivo de los modelos combinado con una umbralización estática/única produce resultados lo suficientemente satisfactorios.

8.2 Trabajo futuro

Como líneas de trabajo futuro en base a este proyecto hemos contemplado varias opciones:

- Entrenamiento con vídeos completos en lugar de imágenes individuales, por lo que se podría usar información espacio-temporal: los objetos deberían verse en imágenes consecutivas, pero con diferentes perspectivas y condiciones de iluminación, oclusiones, etc.
- Integración de un mecanismo que permita registrar anotaciones en base a predicciones del modelo que sean correctas y no estén anotadas.
- Generalización del sistema para trabajar en modo *Stream Learning*, es decir, intercalando aprendizaje e inferencia, lo que permitiría aplicar técnicas de auto-etiquetado.
- Ampliar el modelo para incorporar información de usuarios expertos a modo de nuevas etiquetas y/o corrección de las inferencias realizadas. Idealmente, durante el *Stream Learning*.

Apéndices

Apéndice A

Material adicional

En este capítulo se detalla la estructura del repositorio del proyecto, abordando los principales archivos y carpetas con una breve descripción.

A.1 Estructura del repositorio

El repositorio empleado para este proyecto[32] contiene los módulos que implementan las funcionalidades del sistema. No existe interfaz gráfica, por lo que los ficheros se ejeutan directamente desde línea de comandos o desde el intérprete de Python. Los principales archivos son:

- **Modelos**. En esta carpeta residen los archivos que usan el módulo Lightning para modelar las arquitecturas. Dentro hay un fichero por modelo, aparte de uno que usan todos con las funciones comunes de cálculo de error y umbralización.
- **EfficientDet**. En esta carpeta se recogen los ficheros de implementación de la arquitectura y dataset específicos de EfficientDet.
- **Model.py**. Este fichero recoge las funcionalidades básicas de los modelos, tales como la carga de pesos, congelación de capas o inferencia.
- **TomatoDataset.py**. Este fichero contiene el procesamiento de los datos para crear un *DataModule* usado por los modelos. Tiene una división jerárquica de datamodules, datasets y adaptadores.
- **config.py**. Recoge los valores de configuración del sistema que afectan al resto de módulos.
- **train.py**. Módulo diseñado para entrenar un modelo, pudiendo especificar si congelar los pesos y guardarlo. Incluye funciones del módulo de monitorización Neptune.ai.

- **README.md**. Fichero en el que se especifican los comandos necesarios para poder ejecutar cada funcionalidad.

Bibliografía

[1] Z. Zhang, C. Xie, J. Wang, L. Xie, and A. L. Yuille, "Deepvoting: A robust and explainable deep network for semantic part detection under partial occlusion," 2018.

[2] K. Saleh, S. Szenasi, and Z. Vamossy, "Occlusion handling in generic object detection: A review," in *2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, Jan. 2021. [En línea]. Disponible en: http://dx.doi.org/10.1109/SAMI50585.2021.9378657

[3] L. Alzubaidi, J. Bai, A. Al-Sabaawi, J. Santamaría, A. S. Albahri, B. S. N. Al-dabbagh, M. A. Fadhel, M. Manoufali, J. Zhang, A. H. Al-Timemy, Y. Duan, A. Abdullah, L. Farhan, Y. Lu, A. Gupta, F. Albu, A. Abbosh, and Y. Gu, "A survey on deep learning tools dealing with data scarcity: definitions, challenges, solutions, tips, and applications," *Journal of Big Data*, vol. 10, no. 1, p. 46, Apr 2023. [En línea]. Disponible en: https://doi.org/10.1186/s40537-023-00727-2

[4] "Logic and artificial intelligence," https://plato.stanford.edu/entries/logic-ai/, accessed: 2024-02-15.

[5] Varios, "Wikipedia: Convolutional neural networks," 2023, consultado el 15 de febrero de 2024. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Convolutional_neural_network

[6] R. Solovyev, W. Wang, and T. Gabruseva, "Weighted boxes fusion: Ensembling boxes from different object detection models," *Image and Vision Computing*, vol. 107, p. 104117, Mar. 2021. [En línea]. Disponible en: http://dx.doi.org/10.1016/j.imavis.2021.104117

[7] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-iou loss: Faster and better learning for bounding box regression," 2019.

[8] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," 2019.

[9] W. Falcon and T. P. L. team, "Pytorch lightning," 3 2019. [En línea]. Disponible en: https://www.pytorchlightning.ai

[10] N. S. Detlefsen, J. Borovec, J. Schock, A. Harsh, T. Koker, L. D. Liello, D. Stancl, C. Quan, M. Grechkin, and W. Falcon, "Torchmetrics - measuring reproducibility in pytorch," 2 2022. [En línea]. Disponible en: https://www.pytorchlightning.ai

[11] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, 2020. [En línea]. Disponible en: https://www.mdpi.com/2078-2489/11/2/125

[12] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," 2017.

[13] R. Solovyev, "Weighted-boxes-fusion," https://github.com/ZFTurbo/Weighted-Boxes-Fusion, 2019.

[14] N. team, "neptune.ai," 2 2019. [En línea]. Disponible en: https://neptune.ai/

[15] "What is agile development?" https://www.opentext.com/what-is/agile-development, accessed: 2024-02-14.

[16] "Manifesto for agile software development," http://agilemanifesto.org/iso/en/manifesto.html, accessed: 2024-02-14.

[17] "Principles behind the agile manifesto," http://agilemanifesto.org/principles.html, accessed: 2024-02-14.

[18] "MLOps Principles," https://ml-ops.org/content/mlops-principles, accessed: 2024-02-14.

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

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

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*. Springer International Publishing, 2016, p. 21–37. [En línea]. Disponible en: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[22] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," 2016.

[23] Z. Tian, C. Shen, H. Chen, and T. He, "FCOS: Fully Convolutional One-Stage Object Detection," 2019.

[24] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2017.

[25] laboroai, "Laboro tomato: Instance segmentation dataset," 2020, https://github.com/laboroai/LaboroTomato [Last accessed: (01/30/2024)].

[26] up2metric, ""tomatod"," 2020, https://github.com/up2metric/tomatOD [Accessed: (01/30/2024)].

[27] V. Tsironis, S. Bourou, and C. Stentoumis, "Tomatod: Evaluation of object detection algorithms on a new real-world tomato dataset," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLIII-B3-2020, pp. 1077–1084, 2020. [En línea]. Disponible en: https://isprs-archives.copernicus.org/articles/XLIII-B3-2020/1077/2020/

[28] G. Moreira, S. A. Magalhães, T. Padilha, F. N. dos Santos, and M. Cunha, "RpiTomato dataset: Greenhouse tomatoes with different ripeness stages," 2021.

[29] S. A. Magalhães, L. Castro, G. Moreira, F. N. dos Santos, M. Cunha, J. Dias, and A. P. Moreira, "Evaluating the single-shot multibox detector and yolo deep learning models for the detection of tomatoes in a greenhouse," *Sensors*, vol. 21, no. 10, p. 3569, May 2021. [En línea]. Disponible en: http://dx.doi.org/10.3390/s21103569

[30] I. P. Castro, "IMCV - External internship: Detection of tomatoes in a greenhouse," 2021, comunicación personal.

[31] A. Mao, M. Mohri, and Y. Zhong, "Cross-entropy loss functions: Theoretical analysis and applications," 2023.

[32] N. Martínez González, "tfg$_t omates$,$''$, 2022.