Clasificación de microplásticos basada en redes neuronales profundas

Proyecto creado por Borja Zarco Cerezo

Supervisado por:
José Javier Lorenzo Navarro
Modesto F. Castrillón Santana

Las Palmas de Gran Canaria, 10 de Junio de 2020

Agradecimientos

En primer lugar me gustaría agradecer a los tutores y profesores que me han ayudado y guiado a lo largo del desarrollo de este trabajo, pues me han logrado realizar el proyecto de la mejor forma posible. Ese empuje me ha permitido, además, aprender nuevos conceptos y mejorar mis habilidades como informático. Además, querría poner el valor el sobresfuerzo que han hecho para adaptarse a la pandemia del COVID-19 con todas las dificultades que conlleva.

En segundo lugar, quiero agradecer a todos aquellos que me han ayudado a formarme y que me han transmitido no solo muchísimos conocimientos, sino también muchísima ilusión, ganas por aprender y querer mejorar cada día. Desde profesores de bachillerato, profesores de la universidad o de cursos externos, hasta grandes profesionales como los que hay en Squaads. De veras, gracias a estas personas puedo decir que me levanto cada día para trabajar en algo que amo y que me apasiona.

Por ultimo, me gustaría agradecer de corazón a todas aquellas personas que me han rodeado a lo largo de mi vida y que me han hecho ser la persona que soy hoy en día. Doy gracias a que durante estos cuatro años he podido conocer a personas maravillosas y que, por suerte, mantengo la amistad con muchas otras que las considero imprescindibles. Además, quiero dar gracias en especial a mi familia por cuidarme, educarme y apoyarme en todo momento. Se dice que la familia es algo que no se puede elegir, pero si pudiera, no la cambiaría por nada del mundo.

Resumen

El excesivo uso del plástico durante las últimas décadas ha dado lugar a un problema medioambiental muy grave. La monitorización de este tipo de desechos es clave a la hora de solventar este problema.

Debido a la situación geográfica de las Islas Canarias, se llevan a cabo importantes labores de muestreo, conteo y clasificación de los plásticos que llegan a su costa. Estas tareas suelen tener altos costes de tiempo y personal, por lo que es fundamental el desarrollo de herramientas que faciliten dichas labores.

El objetivo de este trabajo es mejorar este proceso mediante una aplicación que analice fotos de estas partículas tomadas con cámaras digitales utilizando redes neuronales.

Abstract

The excessive disposal of plastic nowadays is a very serious environmental hazard. Monitoring this type of waste is essential in order to solve this significant problem.

Due to its geographical location, large amounts of debris are drifted to the north coast of the Canary Islands. Thus, important tasks of sampling, counting and classification of these plastics are carried out. These tasks tend to have high costs in terms of time and personnel, so it is essential to develop tools that ease these tasks.

The aim of this work is to improve the detection and classification by developing an app. In order to achieve this, it will analyse images of this particles taken by digital cameras using neural networks.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Vivimos en un mundo superpoblado. Desde los años 50, el aumento de la población ha sido de forma exponencial. Esto ha provocado que se dispare la manufacturación y consumo de bienes. Desafortunadamente, el tratamiento de desechos y cuidado del ecosistema no se ha visto afectado de la misma manera. Día tras día se tiran a la basura toneladas de materiales que dañan aquel ecosistema al que van a parar. Hasta hace unos años la única medida que se tomaba era amontonar estos elementos en un vertedero o se lanzarlos al mar y esperar a su degradación con el paso del tiempo.

Un gran porcentaje de estos desperdicios lo forman los plásticos de envases, botellas, bolsas y demás objetos de un único uso [15]. Por desgracia, la rapidez con la que se desechan estos elementos no es comparable con el tiempo que tardan en degradarse. El plástico es uno de los elementos que más tiempo permanece entre nosotros, pudiendo llegar a destruir ecosistemas completos. Se trata de un material que no se biodegrada, únicamente se desintegra en porciones de menor tamaño. Hasta en forma de fragmentos de pequeño tamaño, denominados microplásticos, resulta perjudicial. Estos trozos, con unas dimensiones inferiores a los 5mm, pueden ser ingeridos por diversos organismos, pudiendo provocar su intoxicación y muerte.

No obstante, este no es el único problema que atesora este componente. Según un estudio de la Universidad de Hawaii [42], su degradación provoca la emisión de gases de efecto invernadero. La exposición prolongada a la luz solar hace que se liberen gases, como metano y etileno, que agravan este fenómeno.

El control de los plásticos es fundamental para el cuidado del planeta. Una mejor gestión de los mismos reduciría de forma significativa la contaminación del mundo en el que vivimos. Para ello es esencial el desarrollo de herramientas que permitan su monitorización. Si se tiene un mayor número de datos sobre este problema, se podrá idear una mejor solución de forma más rápida.

Este proyecto tiene como objetivo elaborar una mejora sobre un sistema de monitorización de microplásticos para el Grupo de Ecofisiología de Organismos Marinos (EOMAR) de la Universidad de Las Palmas de Gran Canaria (ULPGC). La finalidad de este trabajo es el de emplear una arquitectura de redes neuronales dentro del campo de la visión por computador

para la detección y clasificación de microplásticos en un tiempo menor al que se obtiene actualmente. El resultado sería, por tanto, una aplicación que en principio permita agilizar el trabajo de los investigadores. Además, al tratarse de un proyecto con espíritu *open source* [19], se permitiría el uso de dicha aplicación a cualquier interesado en realizar la tarea.

De esta manera se pretende aportar un pequeño grano de arena en la lucha contra esta terrible situación. El bienestar del planeta se trata de una cuestión primordial. Toda ayuda, por pequeña que resulte, permite el cuidado de un aspecto de gran relevancia como es la salud de nuestro hogar.

1.1. Estado actual

Como se ha mencionado anteriormente, este trabajo parte de uno previo publicado en IEEE que consiste en la automatización de la detección y clasificación de microplásticos [27]. En dicho trabajo, se hace un estudio previo de la eficiencia del proceso manual.

La detección y clasificación manual consiste en tres pasos fundamentales. En primer lugar, se ha de proceder a la preparación de la muestra, generalmente en un folio blanco. Posteriormente, se pasa a la cuenta y clasificación de los diferentes elementos en una muestra mediante una exploración visual por una persona experta en este tipo de elementos. Por último, se retira la muestra analizada y se limpia el entorno para evitar errores en un siguiente conteo y clasificación. Además, en el mencionado estudio previo se realizó una medición de los tiempos que se invierten en analizar cada muestra.

Sample	Preparation	Transp. Scan.	Color Scan.	Count. & classifying	Remove sample	Total
1	0:00:20	-	-	0:22:23	0:01:11	0:23:34
2	0:00:20	-	-	0:22:23	0:01:11	0:23:34
3	0:00:20	-	-	0:22:23	0:01:11	0:23:34
4	0:00:39	-	-	0:24:33	0:00:42	0:25:15
5	0:00:31	-	-	0:16:40	0:01:07	0:17:47
6	0:00:41	-	-	0:24:00	0:01:26	0:25:26
7	0:00:47	-	-	0:19:14	0:01:25	0:20:39
8	0:00:25	-	-	0:22:27	0:01:28	0:24:20
9	0:00:44	-	-	0:17:46	0:02:43	0:21:13
10	0:00:27	-	-	0:16:16	0:02:21	0:19:04
11	0:00:23	-	-	0:18:47	0:01:00	0:20:10
12	0:00:41	-	-	0:17:55	0:02:40	0:21:16
Average	0:00:32	-	-	0:20:24	0:01:32	0:22:09

Ilustración 1.1: Tabla de tiempos del proceso manual obtenida en el estudio. Fuente: [27]

Como se puede observar en la figura 1.1, la metodología manual supone un enorme gasto de tiempo para los expertos. Estos tiempos se consiguen reducir de forma significativa (algo menos de la mitad). Para la automatización de esta metodología se ha de utilizar un escáner de alta resolución, concretamente el modelo *Epson Perfection V800*. El nuevo procedimiento implica dos escaneados: uno transparente y otro de color. Seguidamente se pasan las digitalizaciones por un modelo de inteligencia artificial previamente entrenada para obtener el

conteo y la clasificación de los elementos en las imágenes. Los tiempos con esta metodología se muestran en la figura 1.2.

Sample	Preparation	Transp. Scan.	Color Scan.	Count. & classifying	Remove sample	Total
1	0:02:22	0:04:26	0:01:45	0:00:28	0:00:21	0:09:22
2	0:03:10	0:06:22	0:00:21	0:01:08	0:00:30	0:11:31
3	0:02:51	0:04:51	0:00:25	0:00:24	0:00:40	0:09:11
4	0:02:33	0:06:18	0:00:25	0:00:34	0:00:50	0:10:40
5	0:02:05	0:06:10	0:00:22	0:00:33	0:00:54	0:10:04
6	0:02:25	0:05:32	0:02:21	0:00:34	0:00:55	0:11:47
7	0:03:49	0:04:28	0:00:22	0:00:39	0:00:38	0:09:56
8	0:03:19	0:04:20	0:00:42	0:00:33	0:00:57	0:09:51
9	0:01:50	0:04:12	0:00:20	0:00:31	0:01:02	0:07:55
10	0:03:38	0:04:04	0:00:21	0:00:26	0:00:32	0:09:01
11	0:03:08	0:04:50	0:00:22	0:00:35	0:00:35	0:09:30
12	0:03:12	0:04:13	0:00:21	0:00:39	0:01:09	0:09:34
Average	0:02:52	0:04:59	0:00:41	0:00:35	0:00:45	0:09:52

Ilustración 1.2: Tabla de tiempos del proceso automatizado obtenida en el estudio. Fuente: [27]

A pesar de la considerable reducción de los tiempos con respecto al proceso manual, esta nueva versión sigue siendo algo lenta. Además, supone una inversión al ser dependiente de elementos *hardware*, como en el caso del escáner. Es por ello que se ha considerado necesario desarrollar una tercera versión, que sea más rápida, no bloqueante e independiente de herramientas específicas para poder realizar la tarea en cuestión. Consideramos que, con las arquitecturas de redes neuronales actuales, se puede realizar una detección bastante fiable en un tiempo menor y que esté disponible a cualquier individuo u organización que desee realizar estas tareas.

1.2. Objetivos

El objetivo de este trabajo fin de grado consiste en evaluar diferentes aproximaciones basadas en redes neuronales profundas para el recuento y clasificación de microplásticos (partículas con un tamaño entre 1 y 5 mm) a partir de fotografías de muestras extraídas de las playas.

Como resultado se obtendrá un prototipo básico de una aplicación que permita realizar el recuento y clasificación de los microplásticos partiendo de imágenes obtenidas mediante cámaras fotográficas o móviles. Este objetivo se puede dividir, a su vez, en los siguientes:

- **Detección de microplásticos:** Dado que una de las actividades que realiza el departamento es el de contar el número de ejemplares tomados en una muestra, habrá que desarrollar un sistema que en primer lugar sea capaz de distinguir entre el fondo y un ejemplar.
- **Clasificación de cada ejemplar:** La otra tarea que se ha de satisfacer es la elaboración de un sistema que, no solo sea capaz de detectar microplásticos, sino también clasificarlos en unos tipos previamente establecidos.
- **Marcado de microplásticos:** Para que el sistema cumpla completamente con los objetivos planteados, ha de generar una imagen idéntica a la que se ha tomado en la que aparezcan marcadas todas las partículas con una especie de rectángulo en el que se incluya una etiqueta con la categoría a la que pertenece el plástico detectado.

1.3. Aportaciones

Este nuevo sistema de detección y clasificación constituiría un avance en la labor de monitorización de residuos plásticos en el océano. Asimismo, permitirá aumentar la eficiencia de las labores, al automatizarlas y poderlas llevar a cabo en tiempos más que aceptables.

Actualmente el proceso es bastante tedioso para los científicos, pues les consume bastante tiempo ya que hay tareas que, si bien están automatizadas, los tiempos que tardan en completarse son bastante grandes. A esto hay que añadirle el alto coste que supone la adquisición y el mantenimiento del equipo necesario para la automatización de dichas tareas.

Así, este trabajo de fin de grado contribuye en los siguientes aspectos:

- Reducir los tiempos de detección y clasificación de microplásticos.
- Aumentar la precisión de detección y clasificación de microplásticos.
- Producir una herramienta que no implique la realización de grandes inversiones en equipo.
- Construir las bases para la monitorización de residuos a nivel global.
- Fomentar la colaboración entre departamentos para reducir la contaminación.

1.4. Competencias cubiertas

En este apartado se expondrán las competencias que han sido cubiertas a lo largo de la realización de este trabajo, así como su correspondiente justificación.

1.4.1. Competencias TFG

- **TFG01**: *Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.*

1.4.2. Competencias específicas

- **CP01**: *Capacidad para tener un conocimiento profundo de los principios fundamentales y modelos de la computación y saberlos aplicar para interpretar, seleccionar, valorar, modelar, y crear nuevos conceptos, teorías, usos y desarrollos tecnológicos relacionados con la informática.*
- **CP04**: *Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.*

Capítulo 2

Estado del arte

En esta sección se va a exponer el estado actual de la tecnología para la detección de objetos a partir de fotos. El campo de la visión por computador es bastante amplio, por lo que en esta ocasión solo vamos a centrarnos en las técnicas más relevantes de aprendizaje profundo usando redes neuronales aplicadas al ámbito de la visión por computador. Este capítulo es un reflejo de todo lo aprendido durante la primera parte del trabajo.

Para realizar el estudio se han comparado las arquitecturas de redes neuronales para la detección de objetos más populares, analizando características como la estructura interna, la facilidad de implementación y entrenamiento, los resultados, tiempos de detección, etc.

2.1. Introducción al aprendizaje profundo

Antes de entrar en materia, es necesario explicar el estado actual del mundo de la visión por computador, también conocida como visión artificial, y por qué se ha elegido el aprendizaje profundo como herramienta principal para resolver este problema.

Como ya se ha mencionado anteriormente, el campo de la visión por computador es bastante amplio y cuenta con multitud de estrategias y herramientas para resolver problemas como el que se plantea en este trabajo.

Dentro de las herramientas más empleadas hoy en día se encuentran el aprendizaje profundo y las redes neuronales. A pesar de su reciente popularidad y auge, estas técnicas fueron desarrolladas hace décadas, alrededor de los ochenta, pero no fueron implementadas hasta hace unos pocos años. Ejemplos pioneros de estos sistemas son el modelo de aprendizaje mediante backpropagation por Rumelhart [43] o el modelo de red convoluciuonal para la detección de números elaborado por LeCun [23].

Las redes neuronales son estructuras altamente complejas que son fruto de múltiples estudios que analizan el funcionamiento del cerebro humano, así como el desarrollo e implementación de modelos matemáticos que sean capaces de emular el comportamiento de

dicho órgano. Para ello fue necesaria la implicación tanto de psicólogos, como matemáticos e informáticos a lo largo de los años.

Sin embargo, estos modelos teóricos requieren una gran capacidad de cómputo y grandes cantidades de datos para funcionar de manera óptima. Por desgracia, dada la alta complejidad de estos sistemas, les fue imposible aplicarlas a problemas reales. En la actualidad la situación ha cambiado bastante. La tecnología ha avanzado lo suficiente como para permitir el uso de estos modelos inteligentes. Ha sido precisamente el aumento de la potencia de los ordenadores y el incremento de la disponibilidad de grandes cantidades de datos lo que ha permitido el desarrollo de estas técnicas tan potentes.

2.1.1. Red neuronal artificial

Dada la alta complejidad de estos algoritmos, se considera necesaria una pequeña introducción al funcionamiento y estructura de estos sistemas. Si bien son bastante conocidos y estudiados, todavía resulta difícil la comprensión de las redes neuronales para aquellos no entendidos en la materia.

2.1.1.1. Estructura

Como bien se ha mencionado anteriormente, las redes neuronales intentan emular el funcionamiento del cerebro humano, un órgano altamente complejo compuesto por células llamadas neuronas. Las neuronas reciben estímulos de otras a través de unas prolongaciones denominadas dendritas, y las reenvían a través del axón.Este modelo está replicado a la perfección en una red neuronal artificial.

Normalmente estos sistemas se dividen en numerosas capas sucesivas. Cada capa contiene una serie de unidades independientes e interconectadas. Estas unidades reciben una cierta información y la transforman aplicando una función particular, generando una salida que envían a otras unidades. Como se puede apreciar, su comportamiento es muy parecido a su inspiración biológica, es por ello por lo que se conoce a estas unidades como *neuronas*.

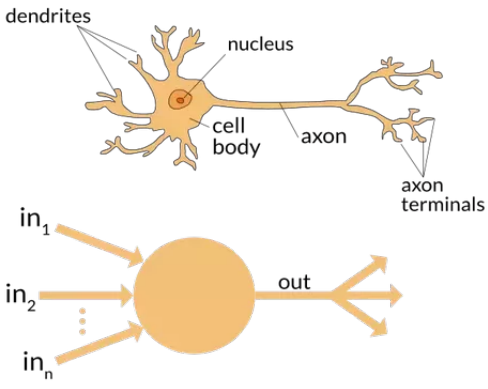

Ilustración 2.1: Comparación de una neurona artificial y biológica. Fuente:[47]

El conjunto de las salidas generadas por todas las neuronas de una capa conforma la salida de la misma. Esta nueva información generada constituye la entrada de la siguiente capa. De esta manera, a medida que la información va avanzando por las diferentes capas intermedias, conocidas como *Hidden Layers* (Capas Ocultas), las neuronas generan una señal en función de determinadas características presentes de forma intrínseca en la información.

El resultado es que en cada capa se va transformando la información para que la salida de la última capa sea una respuesta acorde a la información proporcionada a la red. Esta salida de la última capa sería la respuesta de la red. Es por ello que la última capa se denomina *Output Layer*, o Capa de Salida. Por contra, la primera capa de la red se conoce como *Input Layer* (Capa de Entrada) al ser la que recibe la información de entrada.

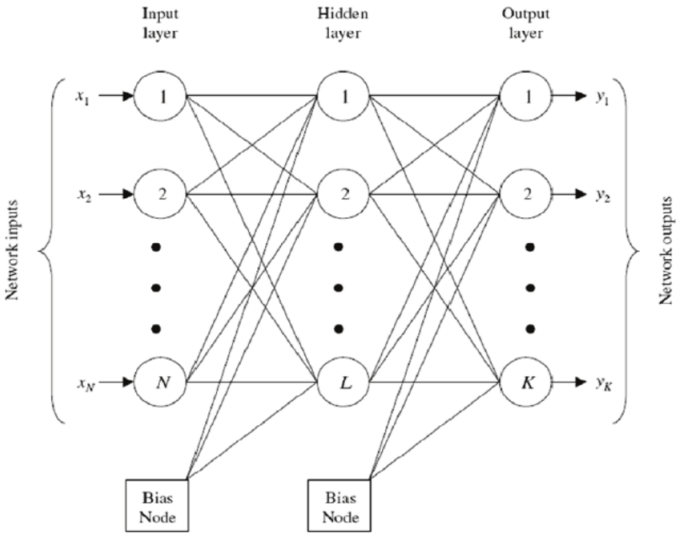

Ilustración 2.2: Esquema básico de una red neuronal artificial. Fuente:[22]

2.1.1.2. Funcionamiento

Como se ha podido comprobar, las redes neuronales logran una aproximación bastante acertada al modelo biológico en cuanto a su estructura. En lo referente a su funcionamiento, también posee características muy similares. En principio, una red neuronal no está preparada para generar valores deseados, ha de pasar primero por una etapa de *aprendizaje* o *entrenamiento*.

El entrenamiento de una red neuronal es un proceso que varía según el tipo de red que estemos tratando. No obstante, la idea fundamental consiste en la aportación de datos a la red para que intente generar una salida deseada en un entorno controlado. Esta salida se suele conocer como *predicción*, ya que la red trata de predecir un comportamiento o valor en función de la entrada, sin conocer con exactitud cómo debería de ser dicho resultado. Una vez la red genera dicha respuesta, se analiza para conocer si se trata de una reacción esperada dentro de ese entorno controlado o si ha errado.

Una vez se conoce el error (en caso de acierto el error sería nulo), se retroalimenta a la red con dicho error para que ajuste parámetros internos. Estos parámetros se conocen como *pesos* y son propios de cada neurona. En otras palabras, en función del error cometido en la predicción cada neurona ajusta sus pesos para generar una salida diferente y que, de esta manera, la salida de la red en su conjunto se aproxime a la respuesta esperada.

Cuando este proceso se ha realizado con todos los datos de un cierto conjunto, conocido como *conjunto de entrenamiento*, se entiende que se ha completado una *epoch* (época o iteración). El entrenamiento de la red consiste en la repetición de estas *epochs* un cierto número de veces o hasta que se considere que el ajuste produce una mejora tan insignificante en las salidas de la red que no merece la pena continuar con el entrenamiento.

2.1.2. Redes neuronales convolucionales

También es conveniente, antes de entrar en la evaluación del estado del arte, una explicación del tipo de red neuronal que comúnmente se emplea en la detección de objetos en imágenes. Hasta ahora, la explicación y diagramas mostrados eran de redes neuronales convencionales, conocidas como redes *fully connected* [5].

Estas redes son bastante comunes y se usan para el análisis de datos más simples, como números o texto. Como se puede observar en la figura 2.2, poseen un número de capas en las que las neuronas están interconectadas entre ellas. Además, las capas poseen una estructura y funcionamiento idéntico, únicamente varían sus pesos.

Sin embargo existen otro tipo de redes que son mucho más efectivas para el análisis de imágenes. Se tratan de las *convolutional neural networks*, o redes neuronales convolucionales [46]. Este tipo de redes poseen una estructura algo diferente a las redes presentadas hasta ahora.

Se puede decir que las redes neuronales convolucionales se dividen en dos grandes módulos, cada uno con un objetivo específico.

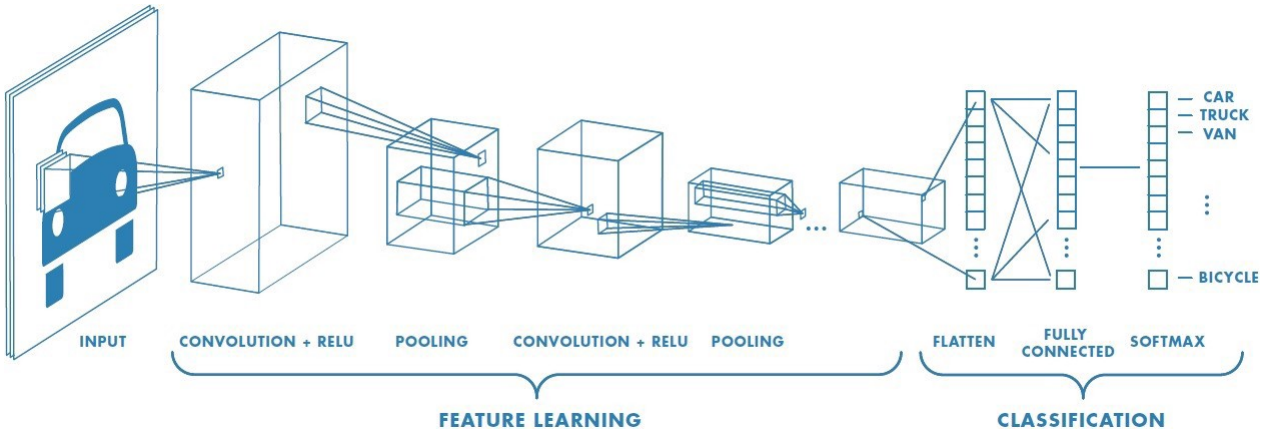

Ilustración 2.3: Estructura básica de una red neuronal artificial. Fuente: [46]

El primer módulo recibe una imagen y tiene como objetivo analizar las características de la misma. Esto se conoce como *feature learning*. Para lograrlo se emplean dos tipos de capa: la capa convolucional y la capa *pooling*.

La capa convolucional sería la encargada de extraer dichas características. A diferencia de las redes *fully connected*, esta capa no estaría formada por neuronas. Las unidades que la forman se denominan *kernels* o filtros. Estos filtros analizan la imagen píxel a píxel y extraen diferentes características de la misma, como sombras, colores o bordes.

Por otro lado, la capa de *pooling* reduce el tamaño de la imagen aplicando distintas técnicas de compresión. De esta manera, se reduce la capacidad computacional requerida para que los filtros de una capa convolucional extraigan características de la misma. Además, al comprimirse la imagen, se acentúan sus características predominantes, permitiendo a las capas sucesivas extraer dichas características con mayor facilidad. Es por ello que una capa convolucional típicamente va seguida de una de *pooling*.

La salida de la red convolucional sería un tensor formado por matrices que contienen un número de características de la imagen. Cada matriz se obtiene en cada capa convolutiva. Este tensor se denomina *Feature Map* (Mapa de Características). Este resultado comunmente se pasa como entrada a un *clasificador*. Se trata de una red neuronal *fully connected* convencional. Es la encargada de generar una salida en base a las características obtenidas en el primer módulo, como se muestra en la figura [46].

2.2. Fast R-CNN

Fast R-CNN [14] (*Fast Region-based Convolutional Network*) es el primer paso para lograr una detección de objetos en tiempo real, o al menos en tiempos reducidos. Se trata de una red convolucional que, como su nombre indica, hace detecciones de imágenes basada en regiones. Esto quiere decir que realiza detecciones, no en la imagen entera, sino en determinadas regiones de la misma. Esta revolución surgió a partir del concepto de *atención*, bastante similar a la realidad. El objetivo es el de centrar la *atención* de la red convolucional en determinadas zonas de la imagen, para que solo trate de clasificar imágenes en dichas zonas.

Para lograrlo, esta red toma como entrada la imagen completa y una serie de regiones propuestas. La imagen pasa por la red convolucional y se obtiene su mapa de características correspondiente. A su vez, las regiones propuestas se pasan por una *Region of Interest pooling layer* (RoI) para obtener las zonas del mapa que se corresponden con cada región de interés propuesta.

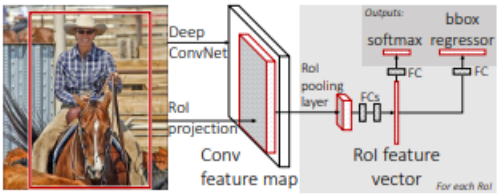

Ilustración 2.4: Arquitectura de Fast R-CNN facilitada en su *paper* [41]

A continuación, se pasan por el clasificador *fully connected* las zonas obtenidas en el paso anterior. Esta red genera dos salidas, la primera salida devuelve, en forma de probabilidad, la certeza que tiene de haber detectado un objeto de alguna clase en la región propuesta. La otra salida indica la posición en la que se encontraría el objeto detectado de cada clase, a modo de 4 puntos. Estos cuatro puntos conforman el *bounding-box*, es decir, delimitan una región en la que estaría localizado dicho objeto. En la imagen 2.4 se puede observar la arquitectura de esta red.

2.3. Faster R-CNN

La arquitectura Faster R-CNN [41] fue una arquitectura publicada en 2015 que pretende ir más allá y mejorar la velocidad de detección de su predecesora, Fast R-CNN. Para lograrlo, introduce una red neuronal entre las capas convolucionales y la red *fully connected*.

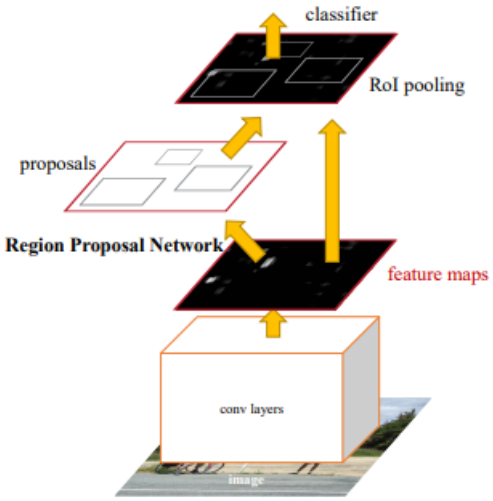

Ilustración 2.5: Estructura de Faster R-CNN facilitada en su *paper* [41]

Esta red se trata de una *Region Proposal Network* (Red de Proposición de Regiones). Es decir, una red neuronal convolucional cuyo objetivo es, no solo el de generear el mapa de características de una imagen, sino el de proponer regiones en las que es más probable que se encuentren objetos a detectar. Esta red toma como entrada una imagen y genera como salida su mapa de características correspondiente, así como las posibles regiones en las que hay más probabilidad de que se detecten objetos.

De esta manera las regiones que se toman como entrada en la capa *RoI pooling* ya no son fijas, sino que las determina una red. El clasificador sigue tomando como entrada el mismo mapa de características que genera la red convolucional.

Por tanto, la mejora de esta arquitectura frente a su predecesora se basa fundamentalmente en lo bien que se predigan las regiones propuestas. De esta manera, no solo reduce el número de falsas detecciones, sino que también decrementa el tiempo necesario para la detección, ya que no se propone un numero fijo de zonas de tamaño invariable, sino que estas son generadas acorde a cada imagen.

2.4. R-FCN

Los creadores de esta arquitectura argumentan que la introducción de la capa *RoI pooling* intermedia reduce el potencial de Faster R-CNN. Indican que, si bien esto aumenta la precisión de la red en cuanto a las detecciones, esta configuración reduce la velocidad de mismas.

Esta capa se introduce porque la detección de los objetos es mejor si se hace independiente de la posición en la que estén. Sin embargo, como también se necesita obtener el *bounding-box* (caja delimitadora), tienen que ser dependientes a la posición. La arquitectura *Region-based Fully Convolutional Network* [8] trata de solucionar el problema.

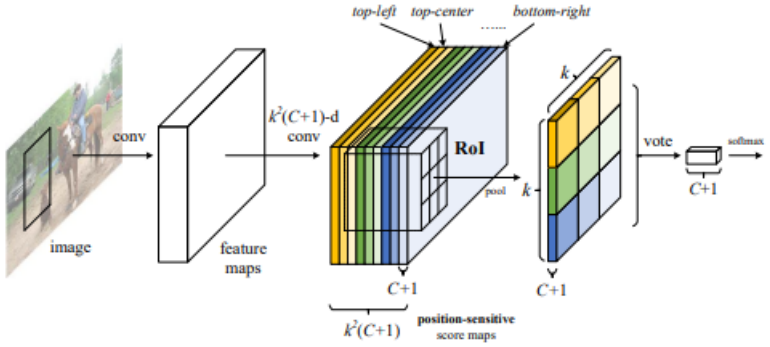

Ilustración 2.6: Estructura de R-FCN facilitada en su *paper* [8]

Para lograrlo, se usa una red totalmente convolucional y se construye un conjunto de *position-sensitive score maps* (mapas de puntuación sensibles a la posición). Estos mapas se obtienen usando un grupo de capas convolucionales especializadas. Cada uno de estos mapas codifica información relativa a la posición del objeto.

De esta manera, se consigue una red que se entrena de principio a fin con la misma información y de la que se obtienen detecciones independientes a la posición, pero aún así se obtienen las cajas delimitadoras de los objetos detectados.

2.5. RetinaNet

Esta otra arquitectura se conoce por el nombre de RetinaNet [24]. El equipo de investigación de Facebook descubrió un grave problema en los detectores del tipo *one-stage*, es decir, aquellos que no se dividen en dos módulos (un detector convolucional y un detector fully connected). Este problema se trata del extremo desequilibrio entre las clases situadas en el fondo de una imagen y las que se encuentran en la parte delantera, conocido como *foreground-background imbalance problem* [32]. La figura 2.7 permite la visualización de esta cuestión. Por este motivo creen los investigadores que este tipo de modelos están un escalón por debajo de los detectores *two-stage*.

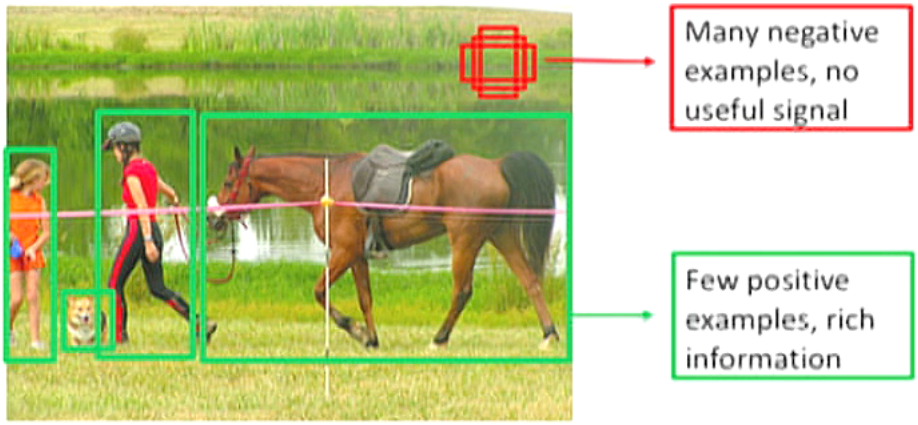

Ilustración 2.7: Representación del problema *foreground-background imbalance*. Fuente: [51]

No obstante, creen tener la solución a este problema utilizando una técnica denomiada *focal loss*. En otras palabras, el valor de la entropía de la red cambia dependiendo de la detección. Si se trata de una del tipo *background*, tendrá un menor peso que en el caso de aquellas realizadas en la parte delantera de la imagen. De esta forma, se reduce la importancia de las detecciones en el fondo y, así, se entrena a la red para centrarse en la parte frontal de la imagen, donde suelen localizarse los elementos a examinar.

2.6. YOLO

Esta arquitectura ataca la detección de objetos desde otro punto de vista. Se trata de una única red neuronal convolucional que predice las cajas delimitadoras y las probabilidades de detección directamente de la imagen completa en una única evaluación. Esto último es lo que utiliza para reducir el tiempo que necesita esta red para realizar las detecciones. De hecho, su nombre *You Only Look Once* [38] (Solo Miras Una Vez), viene dado por la capacidad que tiene esa red de asemejarse al comportamiento humano, al realizar las detecciones con un solo *vistazo* a la imagen.

YOLO cuenta con varias versiones, se trata de un modelo que ha ido mejorando con el tiempo. En cada versión de YOLO se añaden nuevas características que la mantienen como una de las arquitecturas más relevantes dentro de su ámbito. En la fecha del comienzo de este trabajo, la última versión era la tercera. Sin embargo, se considera necesario explicar cada una de las versiones para comprender completamente el funcionamiento de esta red.

2.6.1. YOLOv1

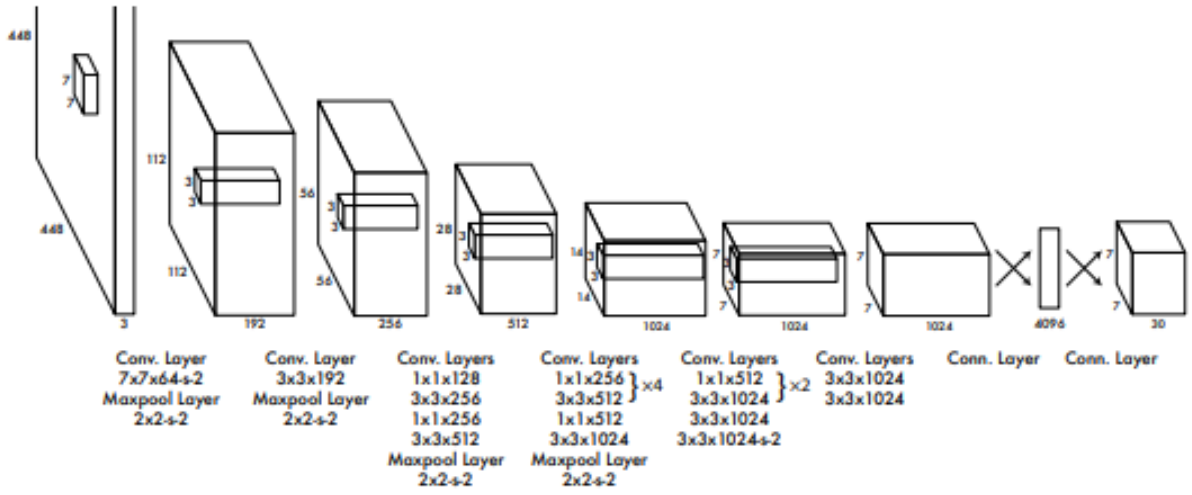

Ilustración 2.8: Estructura de YOLO facilitada en su *paper* [38]

Para lograr esas detecciones en tiempos bastante reducidos, YOLO divide la imagen en un número de celdas. En cada celda se intenta predecir un solo objeto, concretamente, se intenta detectar si en esa celda se encuentra el "centro del objeto". Una vez obtenido ese punto, se intenta predecir un número fijo de regiones que delimiten al objeto. Este proceso se repetiría para cada celda de la imagen.

Por tanto, para cada celda, la red generaría un vector que indica la posición de cada posible caja delimitadora, junto con la probabilidad de que dicha caja sea la correcta. Además, genera

un listado con las probabilidades de que el objeto detectado pertenezca a cada una de las posibles clases. Esto viene bien reflejado en la ilustración 2.9 y que aparece en su *paper*.

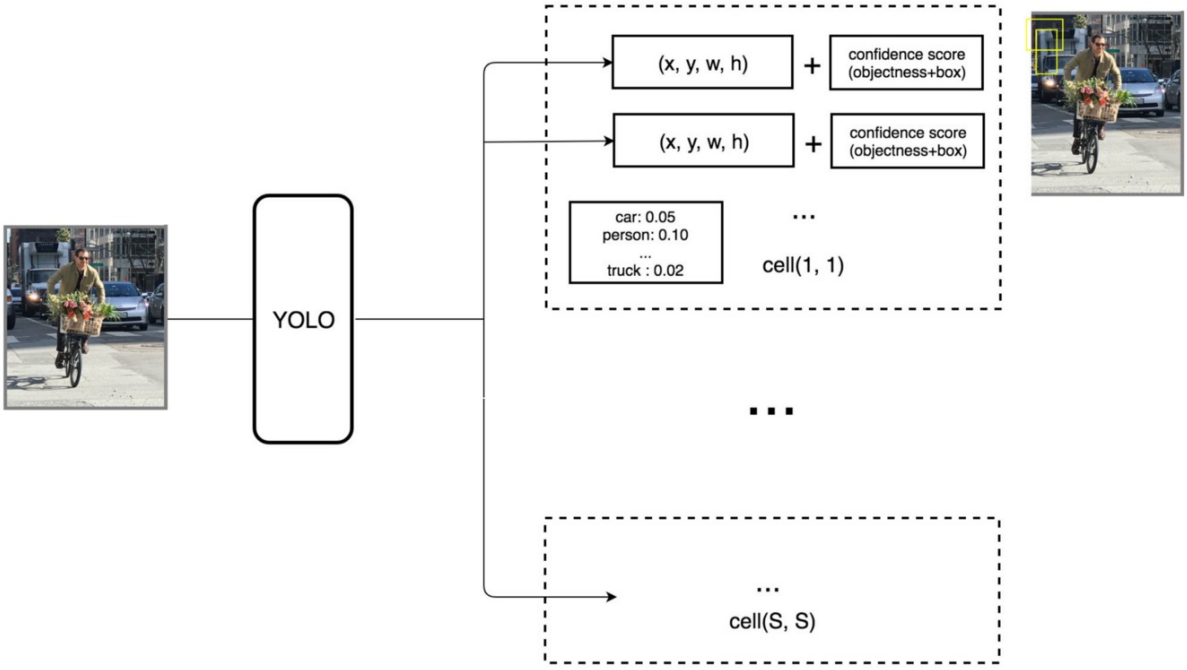

Ilustración 2.9: Salida de YOLO facilitada en su *paper* [38]

A pesar de que la detección de un único objeto por celda le otorga a YOLO su principal ventaja, su rapidez de detección, también supone su principal inconveniente. La primera versión de YOLO tiene bastantes problemas en la detección de objetos cuando estos están superpuestos o muy cercanos.

2.6.2. YOLOv2

Para solventar el problema de la superposición de objetos, la segunda versión de esta arquitectura, conocida como YOLOv2 o YOLO9000 [39], se centra en perfeccionar la precisión de esta red, sin que esto suponga una pérdida en su característica rapidez de detección.

El incremento en la precisión la consigue introduciendo nuevas técnicas y cambioss a la arquitectura. Dentro de estas mejoras destacan el uso de un clasificador de alta resolución, aumentando la resolución requerida de la imagen de entrada de $256 * 256$ a $448 * 448$; el uso de cajas delimitadoras predeterminadas en el entrenamiento, que impulsa significativamente el aprendizaje de la red, sobre todo en las etapas iniciales; y la predicción directa de la localización, a partir de las cajas delimitadoras propuestas.

Como se puede observar en la ilustración 2.10, gracias a los cambios mencionados anteriormente, junto con otros, se obtiene un refinamiento significativo de la precisión del sistema.

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Ilustración 2.10: Comparación de la precisión entre YOLO y YOLOv2 facilitada en su *paper* [39]

Asimismo, la mejora de precisión viene acompañada de un incremento en la velocidad de detección. Esto se logra mediante un cambio en la estructura principal de la red por una arquitectura conocida como *darknet* [37] que requiere un menor número de operaciones para realizar las detecciones.

Como resultado, la nueva versión de YOLO genera una mejor detección y reduce los tiempos de su predecesora. Además, la salida de la red varía ligeramente. Ya no se tiene un vector con las probabilidades de que un objeto sea de una determinada clase para todos las cajas delimitadoras de una celda. Ahora se obtiene ese mismo vector para cada caja delimitadora permitiendo, por tanto, la detección de varios objetos dentro de una misma celda. La figura 2.11 permite la visualización de esta nueva configuración.

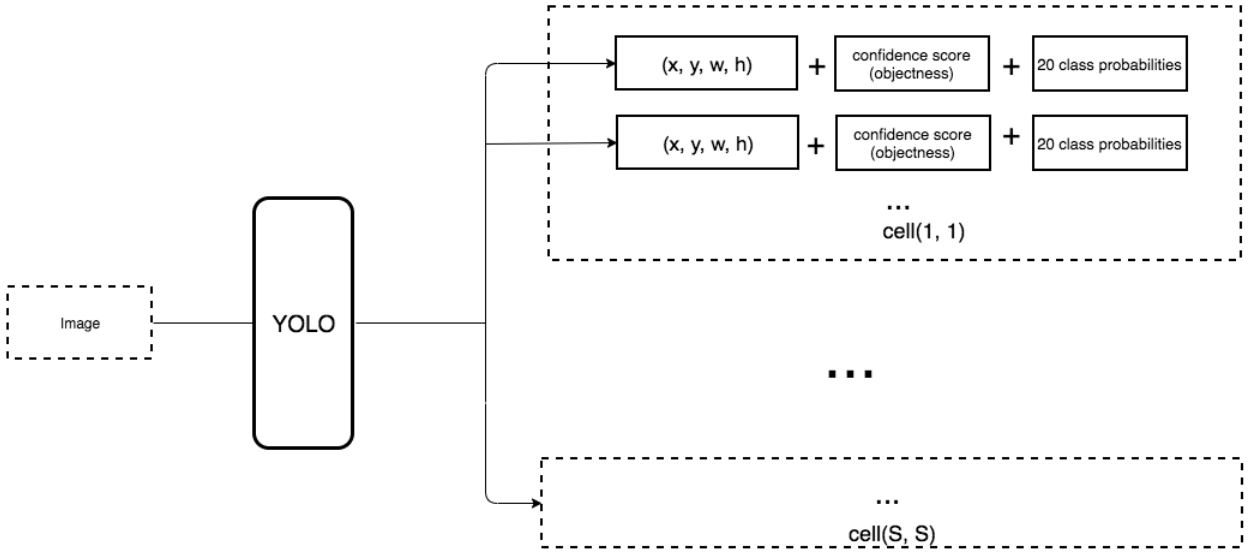

Ilustración 2.11: Salida de YOLOv2 facilitada en su *paper* [39]

2.6.3. YOLOv3

En su tercera versión [40], YOLO se convierte en arquitectura que, a cambio de la reducción de su característica velocidad, gana bastante robustez. Esta red deja de sufrir a la hora de detectar objetos pequeños. Además se añaden características que están presentes en la mayoría de sus competidores, como *skip connections* o *upsampling*, que mejoran el rendimiento general de la red.

Una de sus diferencias principales con respecto a su antecesora es el cambio de la estructura. Anteriormente, YOLO usaba una *darknet* de 19 capas a las que le añadía 11 más. En la tercera versión, se utiliza una *darknet* de 53 capas a las que añade otras 53, llegando a un asombroso total de 106 capas convolucionales. Esta es la principal razón de la pérdida de velocidad de YOLO.

No obstante, esto permite una singular estrategia: la detección a tres escalas. Como se puede observar en una impresionante ilustración de su arquitectura 2.12, a medida que la imagen avanza por la red, se va reduciendo el tamaño de las celdas y consecuentemente aumenta su número.

Es precisamente esto lo que permite a esta versión de YOLO detectar objetos de pequeño tamaño. A mayor número de celdas, más objetos puede detectar (recordemos que solo se permitía un objeto por celda). Por tanto, cuantos más objetos puede detectar, más preciso es. Esto permite a YOLO ponerse de nuevo al mismo nivel que el resto de arquitecturas del momento.

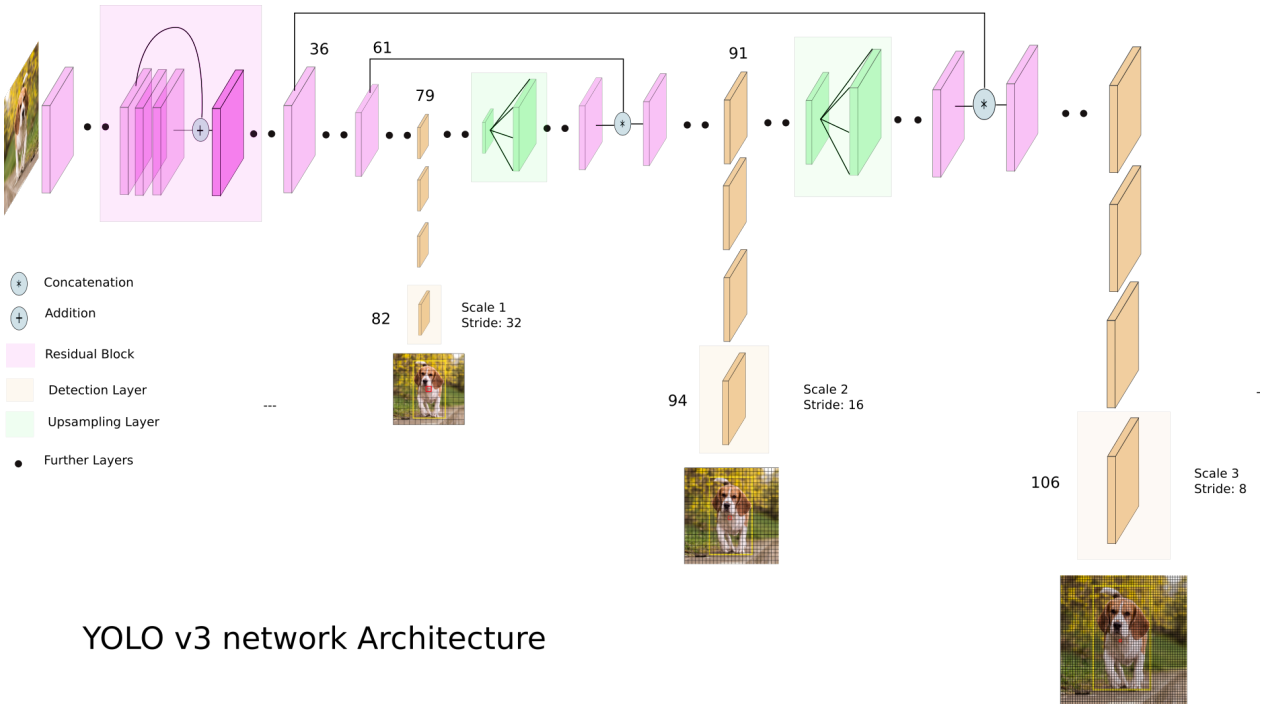

Ilustración 2.12: Arquitectura de YOLOv3 facilitada en [20]

2.7. SSD

Esta ultima red es muy similar a YOLO. La arquitectura *Single Shot Multibox Detector* contiene en su nombre sus tres principales características. En primer lugar, se realiza la localización y clasificación de objetos en un único ”intento”. Esto le permite a la red tener unos tiempos de procesamiento más que correctos.

Por otro lado, se trata de un modelo que aplica el concepto *multibox* presentado por Szegedy [50]. Este concepto no es otro que el de emplear en el entrenamiento un conjunto de cajas delimitadoras que trate de ajustarse al tamaño y posición reales de los diferentes tipos de objetos a detectar. De esta forma se mejora significativamente la precisión de las cajas delimitadoras resultantes tras el proceso de aprendizaje de la red.

Finalmente, se trata de un detector. Es decir, se trata de una red que no se limita a la detección de los diferentes elementos en una imagen, sino que también los clasifica.

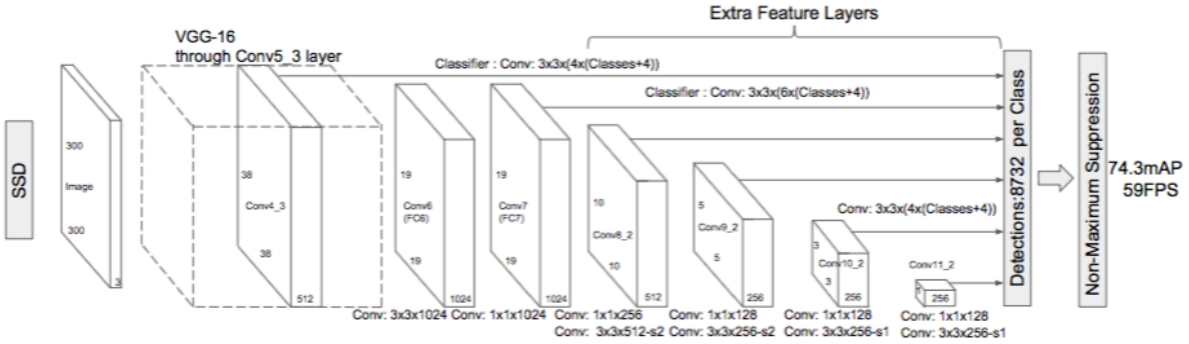

Ilustración 2.13: Arquitectura de SSD. Fuente: [26]

La principal diferencia que presenta esta red con respecto a YOLO es su arquitectura. En lugar de *darknet*, se utiliza la estructura de VGG-16 [48], con una ligera modificación. En SSD, se sustituye las últimas capas *fully connected* por una serie de capas convolucionales auxiliares, lo que permite la extracción de características a múltiples escalas. En la figura 2.13, se puede observar un esquema de esta arquitectura.

Capítulo 3

Metodología de trabajo

Como ya se ha mencionado anteriormente, una buena planificación y selección de las herramientas puede tener un impacto significativo en el desarrollo de un proyecto. En esta sección se van a exponer los distintos instrumentos que se usaron para realizar este trabajo, así como la filosofía que se empleó para planificar y distribuir las tareas del proyecto. Se considera que una breve explicación de cada técnica o aplicación facilita la lectura y el entendimiento del resto de secciones del trabajo.

Además se va a defender el uso de estas metodologías para la consecución de los objetivos propuestos al inicio. El objetivo no es únicamente el de exponerlas, sino el de poner en valor el acierto que supone haber confiado en ellas. Esta es una manera de agradecer a los creadores de estas herramientas. La ausencia de las mismas hubiera dificultado de sobremanera este trabajo.

3.1. Planificación del trabajo

Este trabajo fue orientado desde un primer momento siguiendo SCRUM [9]. Se trata de una metodología de desarrollo ágil [21], principalmente de *software*.

Parte de la idea de que el desarrollo de cualquier aplicación debe ser un proceso iterativo incremental. Es decir, el proyecto se ha de dividir en etapas de una corta duración en el que se realizan un número de tareas. Estas etapas se conocen como *sprints* y generalmente tienen una duración de unas dos semanas. Al finalizar una etapa, el equipo de desarrollo se reúne y se presenta el estado del proyecto a los clientes. En esta reunión se comentan las mejoras que se han incluido durante dicho *sprint*, se evalúa el estado del mismo y se establecen los objetivos a cumplir en la siguiente etapa.

Esta continua revisión del proyecto permite detectar y corregir errores en el trabajo de una forma más rápida. Así, la hoja de ruta del mismo se vuelve tolerante a cualquier cambio o imprevisto. Por consecuencia, hay una mayor probabilidad de que el producto resultante

tenga un menor número de errores o que se adapte a unas nuevas circunstancias con más facilidad.

3.1.1. One Week Sprint

Dentro de SCRUM existen multitud de vertientes o pensamientos. Uno de ellos se trata del *One Week Sprint* [16]. Esta idea consiste en reducir al mínimo la duración de los *sprints*. Concretamente se reducirían a la mitad, ya que serían, como bien indica el nombre de esta vertiente, etapas de una semana. Consecuentemente, al igual que disminuye la duración de un *sprint*, también se reduce el número de tareas a realizar durante ese tiempo.

En un principio se puede pensar que esto puede llevar a que el desarrollo sea menos productivo. Al reducirse las tareas se entiende que la productividad decae, que el proyecto aparentemente no avanza y que las reuniones, que aumentan su asiduidad, supondrán una "pérdida de tiempo". Sin embargo, la realidad es todo lo contrario, siempre y cuando el número y dificultad de las tareas se ajuste fielmente a las estimaciones de desarrollo.

Reducir el tiempo del sprint permite al equipo de trabajo ser aún más tolerante a improvistos, cambiar el rumbo del proyecto de una manera más sencilla y el *feedback* es constante.

Esta fue la metodología empleada para realizar este trabajo. Cada semana se mantenía una reunión con los tutores y otros alumnos que estaban realizando trabajos similares y se presentaba el estado del proyecto. A continuación, se ideaban soluciones ante cualquier problemática imprevista durante esa semana y se establecían las tareas que se deberían completar antes de la siguiente reunión. Como es lógico, recibir el consejo del profesorado casi de forma continua facilitaba la corrección de errores y el entendimiento de los diferentes problemas a atacar en el proyecto.

3.2. Herramientas utilizadas

En este apartado se explicarán con detalle cada una de las herramientas empleadas, tanto su configuración básica, como su funcionamiento. En el trascurso de este texto, hay ocasiones en las que se va a hacer referencia a ciertas características de algunas de estas herramientas. Por tanto, se cree necesario que el lector ha de conocer dichos detalles para una comprensión de la lectura.

3.2.1. Python

Python [13] es lenguaje de programación principalmente utilizado en este proyecto. Se trata de un lenguaje bastante popular hoy en día por su tremenda sencillez. Esta facilidad para entender el lenguaje es comparable a su simpleza en cuanto a su instalación. Python se puede ejecutar en cualquier máquina y puede interactuar con otras aplicaciones o el sistema operativo del ordenador de manera sencilla.

Dada su alta versatilidad y popularidad, se ha convertido en el principal lenguaje en temas de inteligencia artificial. Las principales bibliotecas relativas a este ámbito están preparadas para Python. Además, cuenta con su propio gestor de paquetes, denominado Pip [11]. Gracias a este gestor, la instalación de las dependencias de un proyecto se reduce a un simple comando.

Por otro lado, cuenta con su propio entorno virtual [12]. En otras palabras, permite la instalación de las dependencias únicamente en el directorio en el que se ubica un proyecto. Esto evita que el funcionamiento del resto de los proyectos alojados en dicha máquina se vean afectados por otros. Por ejemplo, si dos proyectos usan una versión concreta de una librería, la ausencia de un entorno virtual hace que solo pueda instalarse una única versión, ya que en una máquina no es posible tener dos versiones de la misma librería para python.

3.2.1.1. Tensorflow Keras

Este proyecto se apoya en múltiples dependencias. Sin embargo, hay que hacer una especial mención para una de ellas, Tensorflow Keras [1]. En realidad, más que una librería, se trata de una API de alto nivel que establece un marco de trabajo para la construcción y entrenamiento de modelos de aprendizaje profundo. En otras palabras, simplifica la construcción y modularización de modelos altamente complejos.

Por un lado, Tensorflow se trata de una biblioteca que ofrece un gran número de herramientas para el desarrollo de algoritmos de alta complejidad para el aprendizaje automático. Ofrece a los desarrolladores un enorme ecosistema en el que trabajar para la creación de todo tipo de soluciones de gran complejidad, desde grafos o tratamiento de tensores, hasta modelos de redes neuronales.

Por otro lado se encuentra Keras [7], otra biblioteca que tiene como objetivo simplificar la creación de modelos de inteligencia artificial. No obstante, la manera de construir estas arquitecturas en Keras es bastante más simple que en Tensorflow, ya que abstrae al desarrollador de bastantes detalles de la configuración de las mismas. Por todo esto se podría decir que Keras constituye un marco de trabajo de alto nivel que permite a la persona que la utiliza despreocuparse de ciertos detalles a la hora de construir estos modelos.

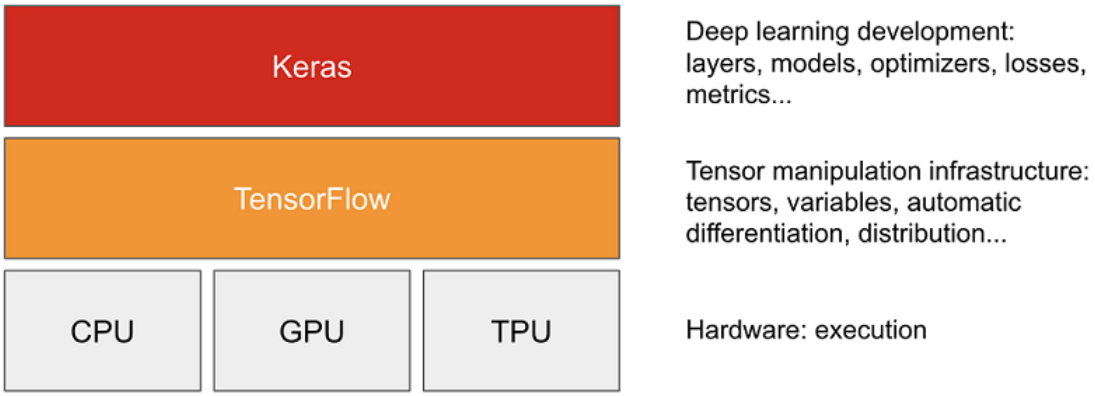

Ilustración 3.1: Visualización de la estructura de Tensorflow Keras. Fuente:[6]

En 2017, Tensorflow comenzó a dar soporte a Keras, constituyendo Tensorflow Keras. Esta plataforma, como se mencionó antes, permite al usuario utilizar las funciones de Keras con la potencia de Tensorflow. Por tanto, se simplifica de manera considerable el diseño e implementación de modelos de aprendizaje profundo. De esta manera, Keras constituiría una capa de alto nivel que se sitúa por encima de las funciones más rudimentarias de Tensorflow, como indica la figura 3.1.

3.2.2. Faktun Batch Downloader

Se trata de una extensión [2] para el navegador Google Chrome que permite la descarga masiva de todas las imágenes que se encuentran en una página web concreta. Esto facilita el aprovisionamiento de imágenes para crear un *dataset* con una cantidad más que aceptable.

Un ejemplo de uso sería la de la búsqueda de una cierta entidad, como ”gato” en Google Imágenes. A continuación, se inicia la extensión y se selecciona la opción de ”Descargar imágenes de la pestaña actual”, como se muestra en la ilustración 3.2. Una vez seleccionada, la aplicación escanea la web y extrae todas las imágenes que se encuentra. Al finalizar este proceso, se abre automáticamente un selector donde finalmente se establece, de entre todas las imágenes detectadas, las que se desean obtener y se procede a su descarga.

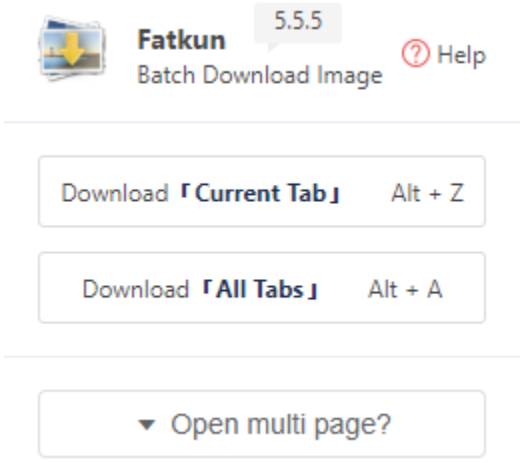

Ilustración 3.2: Parte de la interfaz de Faktun Batch Downloader. Fuente:[2]

3.2.3. Microsoft Visual Object Tagging Tool

Como bien indica su nombre, la herramienta *Visual Object Tagging Tool* [28], también conocida como *VoTT*, tiene como objetivo facilitar el etiquetado de imágenes. Se trata de una aplicación escrita en *React* [10] y *Electron* [34]. Esto permite una fácil instalación y uso, ya que se puede ejecutar como aplicación de escritorio o como web.

Como se mencionará a continuación, una gran labor de este trabajo consiste en crear un conjunto de imágenes etiquetadas para el entrenamiento de una red neuronal. Si bien esta tarea resulta monótona y tediosa, se trata de una de las partes fundamentales del trabajo. Un mal etiquetado de las imágenes puede suponer que la red funcione de forma incorrecta. Esta fue, por tanto, otra aplicación imprescindible en el trabajo. Con su simple interfaz permite la realización del proceso de preparación de la información de una manera sencilla y en una menor duración.

Esta aplicación divide el etiquetado en tres simples pasos. En primer lugar, se seleccionan el directorio en el que se encuentran las imágenes a etiquetar y el directorio de salida, se establecen las posibles clases con las que se va a etiquetar y el formato de salida que desea.

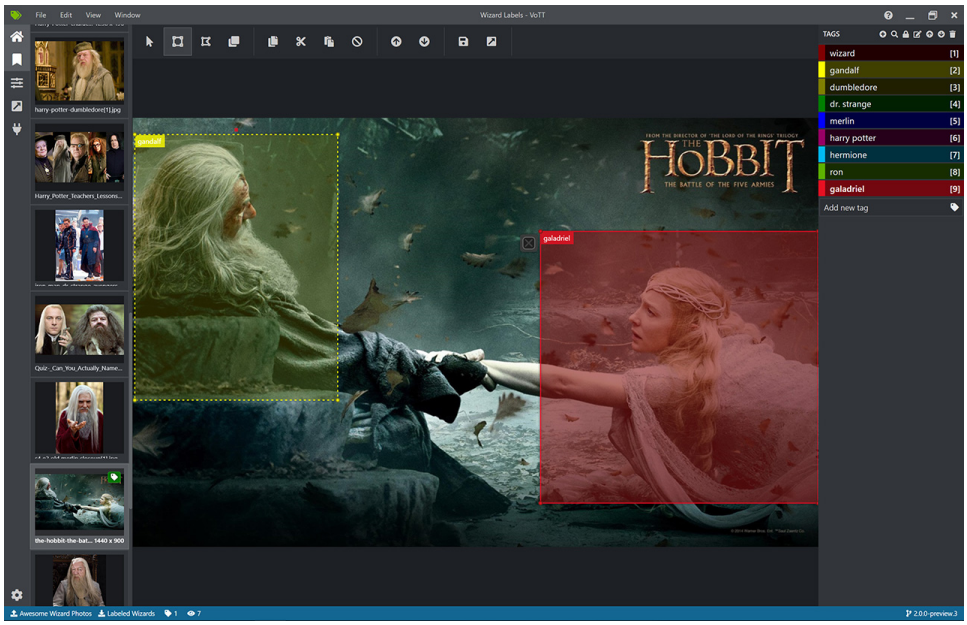

Ilustración 3.3: Interfaz de etiquetado de VoTT. Fuente:[28]

A continuación, se puede navegar por cada imagen e ir marcando en cada una de ellas las diferentes zonas en las que hay un objeto a detectar. Asimismo, se permite al usuario definir de qué clase es dicho objeto marcado, como se muestra en la figura 3.3. Una vez se pasa por cierto proceso de aprendizaje, el etiquetado de las imágenes se vuelve bastante sencillo. Esta aplicación dispone de una serie de combinaciones de teclado que, una vez conocidas, agiliza el etiquetado de forma considerable. Finalmente, con un simple comando se genera el conjunto de imágenes etiquetadas en el formato y directorios previamente establecidos en el primer paso.

3.2.4. Train Your Own YOLO

Train Your Own YOLO [3] es una aplicación que permite el entrenamiento de una red YOLOv3 de forma sencilla. Automatiza la realización de ciertas tareas de instalación y confi-

guración que resultan triviales. De esta manera, permite al usuario entrenar esta arquitectura en poco tiempo.

A su vez, hace uso de una API [29] desarrollada sobre una implementación de YOLO en Tensorflow Keras [35]. De esta forma, actuaría como una mera interfaz. A pesar de esta alta dependencia, se deseaba invertir el menor tiempo posible en la instalación de la red y su posterior entrenamiento siempre y cuando los resultados sean favorables. Es por ello que se realiza un entrenamiento de prueba preliminar, como se menciona en la sección 4.3 antes de comenzar con el trabajo.

El proceso de instalación de este sistema es bastante sencillo. Cuenta con su propio entorno virtual y un fichero en el que se especifican los requisitos a instalar. Adenás, cuenta con un sistema de organización por carpetas. Cada una de ellas tiene un propósito específico para cada proceso de entrenamiento y uso de una red. A continuación se expone de forma breve cada una de las carpetas más relevantes del proyecto, indicando su contenido y el objetivo que cumplen.

- **Image Annotation:** Dentro de este directorio se encuentra un script en Python que convierte el formato en el que VoTT ha generado las imágenes a un formato requerido por YOLOv3.
- **Training:** En esta carpeta hay dos ejecutables. El primero automatiza la descarga de unos pesos preentrenados con el dataset ImageNet 1000 [44]. El segundo inicia el proceso de entrenamiento de la red. Al finalizar, guarda en un directorio determinado los pesos obtenidos.
- **Inference:** En este directorio se encuentran el último script. Se encarga de automatizar la detección de todas las imágenes que se sitúan en una determinada ubicación.
- **Data:** Dentro de este directorio se hallan dos subcarpetas bastante relevantes.
    - **Model Weights:** Aquí se encuentran los pesos que se obtienen del modelo entrenado, entre otras configuraciones del mismo.
    - **Source Images:** En esta carpeta se localizan las imágenes de entrenamiento de la red, así como aquellas sobre las que se desea que el modelo realice sus predicciones una vez entrenado y el resultado de dichas predicciones. A su vez, cada tipo de imagen se ubica en un directorio correspondiente.

Capítulo 4

Desarrollo

Como ya se ha mencionado en el apartado anterior, la realización de este trabajo fue un proceso iterativo incremental. Por tanto, en este capítulo se va a exponer los diferentes pasos que se fueron dando a lo largo del proyecto, así como los objetivos que se cumplieron. Al tratarse de un proceso dirigido por la consecución de metas, la exposición se hará de manera cronológica para facilitar la comprensión del mismo.

Antes de entrar en el desarrollo como tal, hay que hacer una mención a la situación anómala bajo la que se realizó este trabajo. El confinamiento debido a la pandemia del COVID-19 supuso un cambio de dirección en el rumbo del trabajo que se comentará a continuación. En otras palabras, este imprevisto ha supuesto la realización de algunas tareas para seguir con la progresión del trabajo, que bajo condiciones normales, no hubieran sido necesarias.

4.1. Selección de una arquitectura

Uno de los aspectos más importantes para la resolución de un problema consiste en la selección de la herramienta que más se ajuste a sus condiciones. Esto puede cambiar el devenir de cualquier trabajo y afecta de manera significativa al desarrollo del mismo. Es por ello que, a pesar de ser el primer paso a dar, se trata de una decisión crítica. Un gran porcentaje del éxito en todo proyecto es el de usar las herramientas correctas.

Por ello se realizó un estudio minucioso del estado de la tecnología. Este estudio se ve reflejado en el capítulo 2. Todas las arquitecturas encontradas están mencionadas en la sección correspondiente. Hay que recordar que para acortar el estudio, solo se incluyeron las redes más populares utilizadas para la detección de objetos en imágenes de manera veloz y eficaz.

Una vez se encontraron las principales arquitecturas que se ajustaran a los requisitos iniciales, se precisaba hacer una comparación para seleccionar una entre ellas. No obstante, la realización de unos *benchmarks* (medidas de rendimiento) propios y posterior análisis iba a resultar un proceso tedioso. Por ello se partió de estudios previos como el que nos ofrece este

artículo [18], en el que se realizan varias comparaciones entre las principales arquitecturas. Estas comparaciones se centran en la velocidad y precisión de estas redes.

4.1.1. Estudio de la precisión

Para el estudio de la precisión primero se ha de definir las medidas que se van a utilizar para evaluar las distintas redes en este aspecto. Por tanto se ha precisado de un apartado previo a la propia comparación para facilitar la lectura de este escrito. La precisión es uno de los aspectos que más peso tienen a la hora de medir la eficacia de una red, por lo que es necesario establecer de forma clara los marcos bajo los que se van a evaluar las redes. Esto servirá incluso para facilitar la lectura de la evaluación del propio sistema que se va a desarrollar en este trabajo, puesto que se volverán a mencionar algunos de estos conceptos.

4.1.1.1. Conceptos relativos a la precisión

En primer lugar, se ha de hacer mención a la manera elegida para determinar la precisión de una red. Esta métrica se denomina *Mean Average Precision* (mAP) [17], o simplemente *Average Precision* (AP).

Para realmente conocer como se obtiene este valor, primero se han de definir los conceptos de *precision* (precisión) y *recall* (sensibilidad). El primero de ellos mide el porcentaje de detecciones correctas y el segundo representa la calidad de estas detecciones, es decir, si se han detectado elementos que no corresponden a los que se debería de detectar. Para cuantificar la precisión y sensibilidad de una red se utilizan, a su vez otras métricas. Estas métricas son las siguientes:

- ***True Positive (TP)***: Los verdaderos positivos son aquellas detecciones que se realizan de forma correcta, es decir que se detecta y clasifica un elementos de la forma que debería.
- ***False Positive (FP)***: Los falsos positivos se corresponden con el número de elementos que detecta la red, pero que no debería de detectar.
- ***True Negative (TN)***: Cuando la red no detecta un elemento que no debería de detectar, se denomina un verdadero negativo y es un comportamiento esperado por parte de la red.
- ***False Negative (FN)***: Los falsos negativos se denominan a aquellos elementos que no ha identificado pero que se espera que lo haga.

Para el cálculo de la precisión, por tanto, se ha de dividir el número de verdaderos positivos, entre la suma de esa cifra y los falsos positivos. Por otro lado, la sensibilidad se corresponde con el cociente de los verdaderos positivos y la suma de ellos mismos y los falsos negativos. Sus definiciones mátemáticas serían, por tanto, de la siguiente manera:

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

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

Asimismo, hemos de conocer otra medida que se tiene bastante en cuenta. Se trata de la *Intersection over Union* (IoU), o intersección sobre la unión. Más que una medida que define la precisión del sistema de forma directa, tiene influencia sobre la misma de manera indirecta. Este valor determina si una detección se considera correcta. Para calcular su valor se ha de obtener el área del rectángulo que se forma al solapar la caja delimitadora que genera la red y la región rectangular que ocupa el objeto real en la imagen. A este valor hay que dividirle el área que la unión de las áreas de las dos regiones, la generada y la real.

$$IoU = \frac{Areadeinterseccion}{Areadeunion} \tag{4.3}$$

Como se ha mencionado antes, la IoU es una medida que incide de forma indirecta en la precisión del sistema. Esto se debe a que una detección se considera correcta siempre y cuando el valor de su IoU supere un cierto umbral.

Por tanto, para calcular la precisión media se ha de hacer un balance entre la precisión y la sensibilidad del sistema en el que se ha fijado un cierto umbral para la IoU. Por ejemplo, si se ha fijado en el 75 %, se indicaría como AP_{75}. La representación matemática de este balance correspondería con la siguiente ecuación:

$$AP = \int_0^1 p(r)dr,\ r \equiv recall \wedge p \equiv precision \tag{4.4}$$

En otras palabras, la precisión media se corresponde con el área que encierra la función que relaciona un determinado valor de sensibilidad con un valor de precisión. Se trata, además, de una métrica muy popular en este ámbito que se usa con frecuencia para cuantificar lo bueno que es un sistema de detección. Es por este reconocimiento por lo que la comparación se realiza usando esta métrica.

4.1.2. Comparación de la precisión

Una vez se ha definido la medida que se va a emplear para realizar la comparativa, hay que mencionar que estos valores se obtuvieron usando un *image dataset* (conjunto de imágenes) muy popular y reconocido en la comunidad, denominado COCO [25]. En otras palabras, cada arquitectura se entrenó con este conjunto y, posteriormente, se realizó una serie de detecciones sobre estas fotografías para hallar los valores representados en la tabla. Como se puede observar en la ilustración 4.1, en cuanto a la precisión, destaca la arquitectura *RetinaNet*, seguida por *Faster R-CNN*, *SSD* y *YOLO*, en las que se obtienen valores algo menores a la primera red, pero que no son tan dispares entre ellos.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [7]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Ilustración 4.1: Comparación de la precisión de distintas arquitecturas usando el dataset COCO [25]. Fuente: [40]

4.1.3. Estudio de la velocidad

En cuanto a la velocidad de detección, la diferencia es significativa. Como se puede observar en la ilustración 4.2, la diferencia entre la velocidad de detección de YOLO es asombrosamente abismal frente al resto de sus competidores. Se trata de una diferencia tan grande que incluso la segunda arquitectura más rápida de las comparadas prácticamente dobla a YOLO en tiempos de detección.

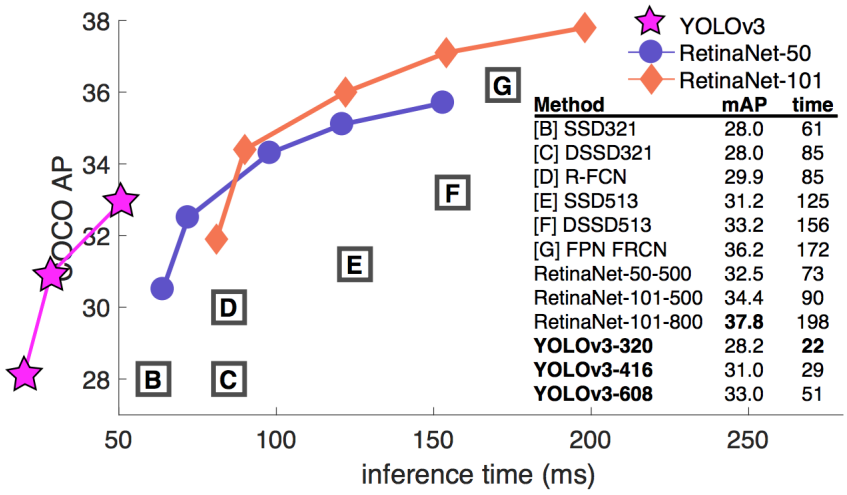

Ilustración 4.2: Comparación de la velocidad de detección de distintas arquitecturas usando el dataset COCO [25]. Fuente: [40]

4.1.4. Decisión final

Una vez hemos realizado la comparación en cuanto a la velocidad y la precisión de los distintos modelos, llega el momento de decidir cuál de ellos vamos a usar para atacar nuestro

problema. Para facilitar la decisión se descartaron todos los modelos de detección salvo tres: SSD, RetinaNet y YOLO.

La arquitectura SSD habría sido un modelo altamente interesante para este problema, de no ser por las dificultades que presenta a la hora de detectar de objetos de pequeño tamaño. Dado que el objetivo de este proyecto es la identificación de objetos de dimensiones reducidas, finalmente se descartó esta red. Por otro lado, pese a la gran precisión de RetinaNet, el enorme tiempo de detección que necesita fue uno de los principales motivos por los que se descartó.

Su apabullante velocidad de detección ha sido determinante para que se escogiera YOLO como herramienta para desarrollar este trabajo. Al tratarse de un problema en el que las detecciones son, *a priori* sencillas, no se ha considerado tan necesario el uso de un modelo como RetinaNet. Es decir, no es necesario el sacrificio de los tiempos de detección, a cambio de un ligero incremento en la precisión. También hay que mencionar que, al ser una de las arquitecturas más populares actualmente, existen un mayor número de facilidades para la instalación, entrenamiento y uso de YOLO.

4.2. Instalación de YOLO

En un principio se pretendía optar por la instalación de la arquitectura original de YOLO. Sin embargo, la instalación de *darknet* era algo compleja y tediosa. Además, la configuración de esta red para el entrenamiento con imágenes requería invertir un tiempo del que no se disponía. Por fortuna, se encontró una aplicación que servía de interfaz y reducía esa configuración a unos cuantos comandos, Train Your Own YOLO.

Por tanto, para el entrenamiento y posterior despliegue de YOLO de manera sencilla se ha utilizado esta aplicación. Su instalación fue bastante simple, tal y como se explica paso a paso en este artículo [31]. Tanto la configuración como el entrenamiento se realiza en un tiempo ínfimo y en unos simples pasos. Tan solo hay que ejecutar una serie de comandos para tener la arquitectura lista.

No obstante, el proceso de instalación no fue tan asequible como se creía. A pesar de que en las instrucciones de uso de esta aplicación se indica que puede usarse en Windows, se encontraron múltiples conflictos con el sistema operativo. Se trató de solucionar dichos errores, incluso probando el revolucionario *Windows Subsystem for Linux* (WSL), pero no dio resultado.

Dado que no se deseaba invertir demasiado tiempo en corregir problemas relativos a la incompatibilidad de esta aplicación con un sistema operativo, se optó por la instalación de Ubuntu 18.04 en una partición de la computadora usada para realizar el trabajo. Si bien se podría haber utilizado una máquina virtual, se optó por la instalación de un sistema operativo en una partición para usar todos los recursos *hardware* disponibles. De esta manera el entrenamiento de la red se podría realizar en un menor tiempo, así como las detecciones.

4.3. Entrenamiento de prueba

Para comprobar el correcto funcionamiento del sistema, se realizó un entrenamiento de prueba siguiendo al pie de la letra las instrucciones del artículo [31] y con un conjunto de datos simple. En otras palabras, se preparó a YOLO para la detección de perros, gatos y *fidget spinners*. Este último se trata de un popular juguete que presenta una forma bastante singular. En la ilustración 4.3 se muestran ejemplos de imágenes utilizadas en el entrenamiento.

Ilustración 4.3: Ejemplos de imágenes usadas en el entrenamiento

En el entrenamiento inicial, además de comprobar el correcto funcionamiento de esta red tras su instalación, se deseaba probar la precisión de la misma. Por ello se incluyen dos entidades muy similares visualmente, perro y gato, junto con otra que no tiene nada que ver. Además, se consideró que sería interesante ver cómo YOLO se comporta detectando una entidad que no está presente en el conjunto de imágenes con el que se obtienen los pesos iniciales. Esta prueba se quería hacer para demostrar que YOLO no tenía problemas en detectar entidades totalmente nuevas, como sería el caso de los microplásticos.

Según venía recogido en la documentación de la aplicación, para entrenar el modelo bastaba con un conjunto de entrenamiento con un mínimo de 100 imágenes por cada clase que se quiera detectar. Para la obtención de estas imágenes se usó una extensión de Google Chrome denominada *Faktun Batch Downloader*. Una vez se tenían alrededor de 100 imágenes de cada entidad, se pasó al etiquetado de las mismas usando *Microsoft Visual Object Tagging Tool* (VOTT).

A continuación se pasó al entrenamiento de la red, que está dividida en dos etapas. La primera consiste en un ajuste de la red convolucional que consta de 50 épocas. Se trata de un proceso lento y tedioso que tarda algo más de una hora en finalizar. La segunda parte consiste en el *fine tuning* del clasificador, cuya duración resulta variable. En otras palabras, no tiene un tiempo fijo ya que finaliza cuando la red detecta que no está logrando una mejora significativa. Es decir, cuando la mejora que se produce entre época y época está por debajo de un cierto umbral que se ha especificado en uno de los hiperparámetros de la red, el proceso de refinamiento termina.

Una vez concluido, se realizaron varias detecciones de prueba para comprobar la eficacia de la red. Como se muestra en la ilustración 4.4, la arquitectura consigue resultados bastante favorables a pesar de solo contar con 100 muestras por cada clase en el conjunto de entrenamiento. Dada la consistencia de dichas detecciones a partir de un entrenamiento con tan pocas imágenes, tanto YOLO, como el programa utilizado se consolidaron como las herramientas a utilizar en este trabajo de forma definitiva.

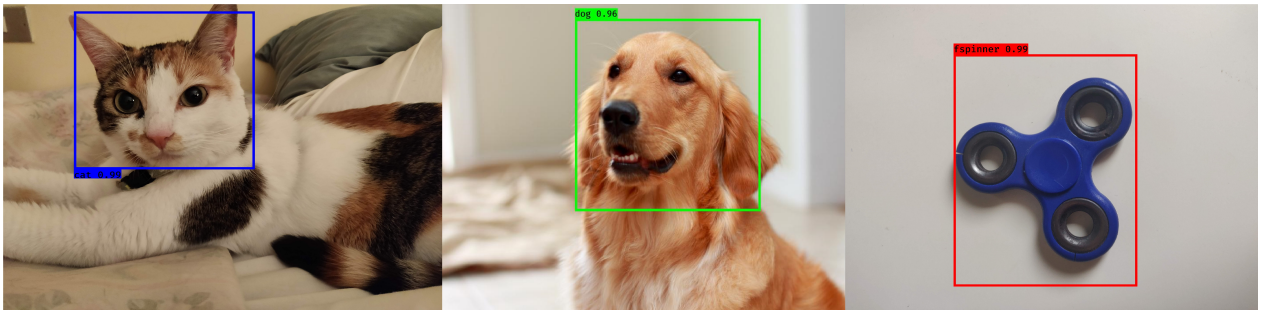

Ilustración 4.4: Ejemplos de imágenes pasadas por el detector entrenado. Fuente: [53]

4.4. Problemática con el COVID-19

El siguiente paso estaba previsto que se realizara con imágenes de microplásticos. Sin embargo, como ya se ha mencionado anteriormente en varias ocasiones, el desarrollo de este trabajo fue afectado por la pandemia. Además, el momento en el que se inició la cuarentena obligatoria no pudo ser más inoportuna. Este imprevisto impidió acceder al conjunto de imágenes de los microplásticos, por lo que hubo que elaborar una estrategia alternativa.

Por suerte, se conocían las dimensiones aproximadas de dichos microplásticos y de las fotografías necesarias para entrenar. Por ello, se podía simular, al menos, la detección de elementos que resultaran de un tamaño similar a la de dichos microplásticos, de unos $200 * 200$ píxeles.

Afortunadamente, las caras de perros y gatos con las que preparó del detector tenían unas dimensiones similares a las de los mencionados microplásticos. Por ello, se escogió reproducir la detección de microplásticos empleando imágenes con caras de perros y gatos. Así sería innecesario realizar un nuevo entrenamiento con clases diferentes, permitiéndonos ahorrar bastante tiempo.

4.5. Estudio del tamaño de las imágenes

Este proyecto está enfocado a la detección de imágenes tomadas con cualquier teléfono móvil o cámara digital. A pesar de que el tamaño de las imágenes cambia dependiendo del dispositivo con el que toman, actualmente estas dimensiones rondan los $4000 * 4000$ píxeles. Por tanto, era necesario evaluar si la red efectivamente conseguía detectar una forma de un

tamaño de hasta 20 veces menor que la imagen completa, ya que como se ha mencionado anteriormente, los elementos son de unos $200 * 200$ píxeles.

Por ello se generó de forma artificial una imagen de $5000 * 5000$ en cuyo centro había una cara de un gato. Esta cara se tomó de la imagen que se muestra en la figura 4.5. Previamente se pasó esta fotografía por el detector para comprobar que la detección se realizaba sin problemas, pues arrojó un valor superior al 95 %.

Ilustración 4.5: Imagen original de un gato. Fuente: [52]

Cuando se obtuvo esta imagen artificial, ilustrada en la figura 4.6, fue recortándose de forma automática. Con cada reducción se iba generando una nueva foto de menor tamaño. La diferencia entre el tamaño de un recorte y el siguiente era de 200 píxeles de alto y ancho. Este proceso se repitió hasta obtener una imagen de 400 píxeles de alto y ancho. Por tanto, se generaron un total de 24 imágenes.

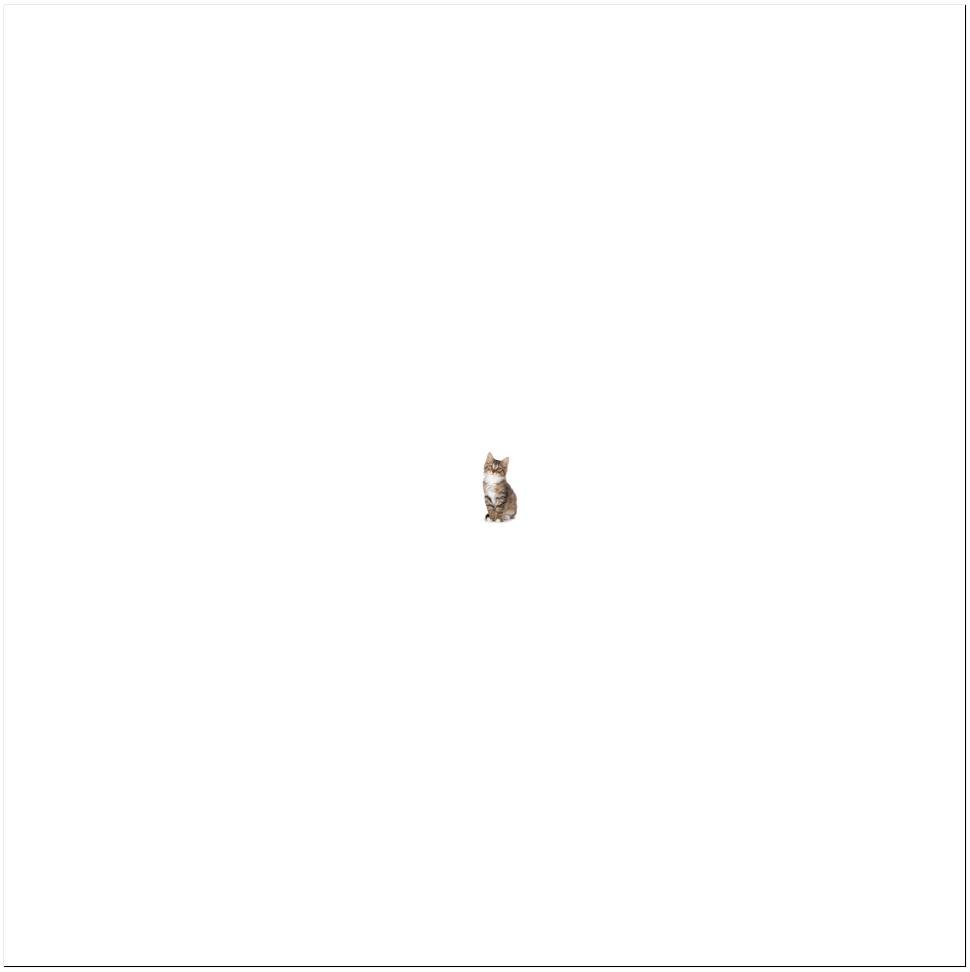

Ilustración 4.6: Imagen generada de un gato de 5000 píxeles

Una vez fueron generadas todas estas imágenes, se trató que el clasificador detectara la presencia de la cara de un gato en cada una de ellas. Al finalizar el proceso de detección, se evaluaron los resultados para comprobar cómo se comporta el sistema. Estos valores obtenidos se observan en la tabla 4.1.

Ancho (píxeles)	Alto (píxeles)	Confianza
400	400	9.83
600	600	6.42
800	800	3.05
1000	1000	8.97
1200	1200	8.40
1400	1400	7.72
1600	1600	7.59
1800	1800	6.01
2200	2200	3.74
2400	2400	4.69
2600	2600	5.13
2800	2800	4.39
3000	3000	4.23
3200	3200	3.54
3400	3400	2.56
4000	4000	2.62
4200	4200	2.90
4400	4400	3.04
4600	4600	2.81
5000	5000	3.19

Cuadro 4.1: Imagen generada de un gato

Como se aprecia en la figura 4.1, a partir de los 500 píxeles la detección promedia de la imagen empeora significativamente. Esto encaja con la hipótesis planteada antes del estudio. Previamente a esta comprobación, se descubrió que la implementación en Keras espera recibir una imagen de $416 * 416$ píxeles. Si el tamaño de la misma es mayor que el esperado, la red la comprime de forma automática, reduciendo su calidad. La reducción del valor de la confianza en las detecciones surge como consecuencia de este empeoramiento en la resolución.

4.6. Ajuste de la entrada a la red

Ante la problemática del tamaño de las imágenes había dos posibles soluciones, o bien se optaba por la modificación de la configuración de la red para permitir una entrada de mayor tamaño o se preprocesaba la imagen para adaptarla a las dimensiones requeridas por la red. A continuación se van a exponer las dos posibilidades, valorando los aspectos positivos y negativos de las mismas.

4.6.1. Aumento del tamaño de entrada

Se consideró modificar los hiperparámetros de la red para que pudiera aceptar un tamaño de imagen de unos $4000 * 4000$ píxeles. Este es un tamaño bastante común para las fotos tomadas por los diferentes tipos de cámaras hoy en día. Por tanto, se permitiría la detección de todas las entidades de una imagen de forma sencilla, como promete la documentación de la red. Sin embargo, esta medida trae consigo diversas problemáticas o consideraciones.

Por un lado, a pesar de ser una resolución común, eso no significa que todas las imágenes que vayan a proveer a la red van a ser de ese tamaño. En otras palabras, si bien se espera que la mayoría de las representaciones tengan aproximadamente las dimensiones mencionadas, puede haber casos en que las fotografías sean de menor o mayor tamaño. Esta fluctuación en las dimensiones de entrada supone que la red puede perder cierta eficiencia y precisión, ya que tendrá que escalar las imágenes de entrada de forma automática. Dado que uno de los objetivos de este proyecto es que cualquier grupo de investigación o individuo pueda usar este clasificador con cualquier tipo de imagen, se trata de un aspecto negativo con cierta importancia.

Por otro, si bien en este punto del trabajo todavía no se tenía acceso a las imágenes, se conocía la escasez de las mismas. En el caso de adaptar la red para esperar imágenes completas, sería casi imposible entrenarla debido a la falta de datos. Se podría intentar realizar un cierto *data augmentation*, o aumento de los datos, es decir, generar datos de forma artificial a partir de los reales. No obstante, dada la tremenda carencia de los mismos, resultaría un conjunto de entrenamiento casi artificial.

Estos dos puntos provocaron descartar el aumento del tamaño de entrada como opción real. Ante la situación del problema y los datos proporcionados, este camino resultaba inviable.

4.6.2. Adaptación de la imagen a la entrada de la red

Esta otra opción, a pesar de reducir la velocidad de las detecciones, aumenta de forma considerable la consistencia del sistema. La adaptación se haría tratando de mantener lo máximo posible el aspecto original de la imagen. Este proceso consiste en dividir la imagen en tamaños iguales a los esperados por la red, es decir, obtener recortes de la imagen original de unos 416 píxeles de ancho y alto.

Dado que las imágenes de entrenamiento son de unos 4000 píxeles tanto de ancho como de alto, se obtendrían unas 100 porciones a partir de la original. De esta manera, se podría conseguir un conjunto de datos de un tamaño sobradamente necesario a pesar de contar con pocas imágenes originalmente. Recordemos que para entrenar el modelo basta con unas 100 muestras de cada clase. Así, apenas se necesitaría una fotografía completa por cada clase para poder entrenar esta red.

Asimismo, se estandarizaría el uso del clasificador. Cualquiera podría utilizar este sistema, sin importar el dispositivo con el que ha tomado la imagen o si ha modificado la misma. La

precisión del detector resultaría invariable al tamaño de la imagen que se ha proporcionado.

Por todo esto, a pesar de que este proceso supone un incremento en los tiempos de detección, la robustez que supone la adaptación de la imagen a la red, al igual que la generación de un conjunto de entrenamiento resulta determinante para que este sea el método escogido.

4.7. Pre-procesamiento: adaptación de la imagen a la red

El proceso de adaptación elegido sufrió ciertas variaciones a medida que se implementaba en el proyecto y se apreciaron ciertas problemáticas. A continuación se discurrirá sobre las dos versiones que se implementaron, así como las dificultades que se afrontaron en esta etapa.

4.7.1. Mosaico simple

La primera versión de este sistema consistiría en una división simple de la imagen original en cuadrículas, como se muestra en la ilustración 4.7. Cada celda de dicho mosaico tendría un tamaño aproximado al requerido por la red. De esta manera, la detección de los elementos en la imagen completa consistiría en la suma de las sucesivas detecciones de cada una de las cuadrículas generadas a partir de la misma.

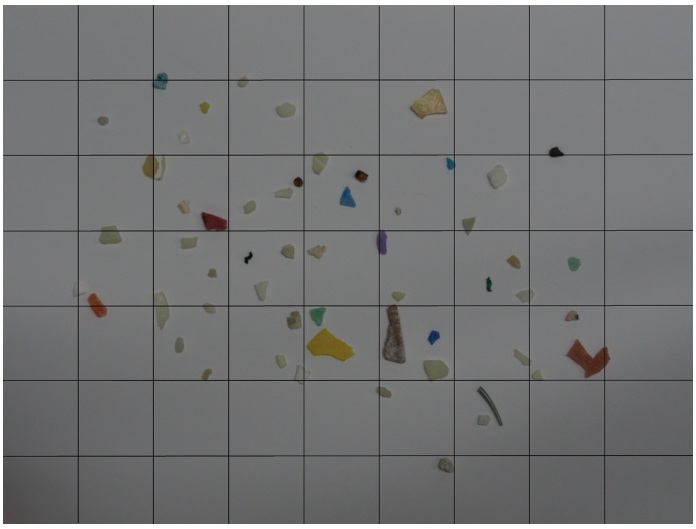

Ilustración 4.7: Visualización del mosaico sobre una imagen de microplásticos

El tamaño escogido sería de unos 512 píxeles de alto y de ancho. De esta manera se proporcionarían a la red imágenes con unas dimensiones algo superiores a lo establecido, pero no lo suficiente como para que esto suponga una pérdida en el rendimiento de YOLO. Además, se generarían menos imágenes. De esta forma, se ahorrarían algunas detecciones reduciendo, así, el impacto en el tiempo de detección que supone dividir la foto original.

A pesar de la aparente idoneidad de esta metodología, surge una gran problemática a tener en cuenta. Esto es, que un objeto a detectar se encuentre en medio de la división que podría resultar en dos casuísticas: la detección múltiple de un mismo objeto o la incapacidad de detección de dicho elemento ya que no se tiene una "visión" completa del mismo.

Como se muestra en la ilustración 4.8, la partícula azul señalada con un círculo rojo cae justo en medio de la división entre dos celdas. Así, se generan dos imágenes en las que en cada una hay una porción de dicha partícula azul. Esto, como es lógico, puede llevar a la detección duplicada de la misma partícula o a que la red no llegue a detectarla ya que no se muestra una forma completa de la misma.

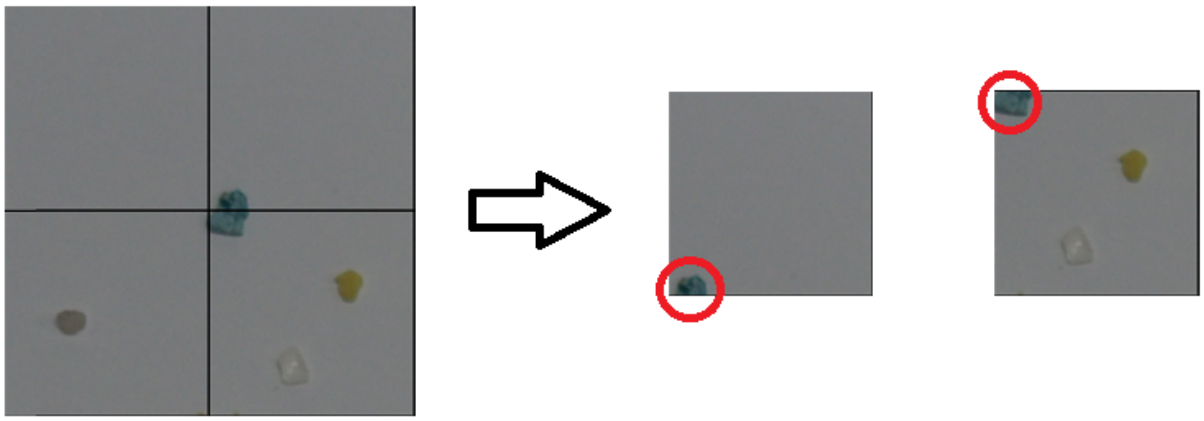

Ilustración 4.8: Visualización de la problemática del mosaico

4.7.2. Mosaico con solape

Para solucionar este problema, se optó por la elaboración de un sistema de solapamiento. Es decir, que las celdas contiguas del mosaico se superpongan, por lo que habrá regiones que provengan de la misma zona de la imagen original. Para ello, se dividió la fotografía en cuadrículas más pequeñas. Seguidamente, estas cuadrículas se combinan para generar un fragmento de un cierto tamaño, en este caso, 512 píxeles. Como se muestra en la ilustración 4.9, al realizar el solape, se generan dos imágenes que tienen una zona en la que se "superponen". Asimismo, se puede observar que en la segunda imagen aparece la partícula prácticamente en su totalidad, algo que en el sistema sin solape no se producía.

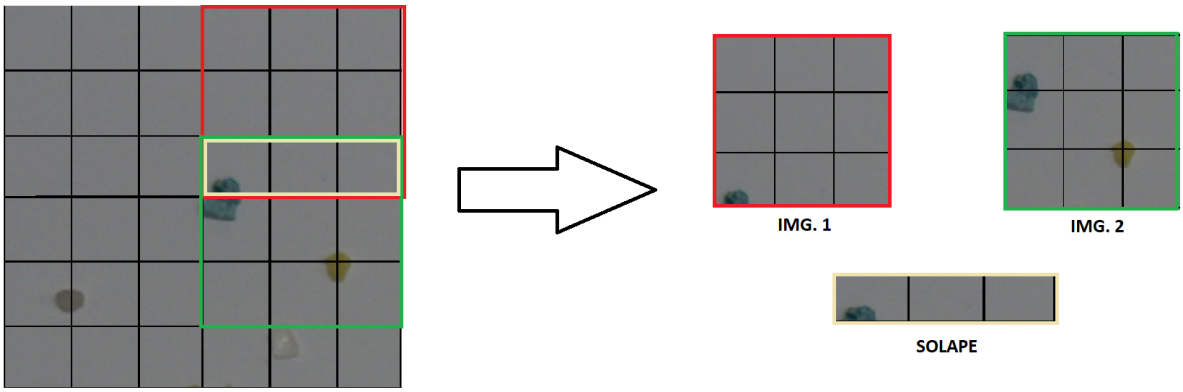

Ilustración 4.9: Visualización del resultado de realizar solape

Obviamente, esto trae consigo la generación de un mayor número de imágenes, que producirá un incremento en los tiempo de detección de la imagen completa. Sin embargo, es un coste que se debe asumir ya que sin esta solución, la utilidad del sistema se vería reducido drásticamente.

El algoritmo que se emplea para conseguir esta labor viene implementado en el código que se muestra en la ilustración 1. Como se puede observar, se va iterando la imagen pasada por parámetros y se van obteniendo sus correspondientes mosaicos. Para ello, se obtienen en primer lugar sus dimensiones. Una vez se obtienen dichas medidas, se dividen la imagen entre un valor determinado. Este valor representa el tamaño del solape entre las imágenes. En la arquitectura del pre-procesamiento que se ha elaborado, valor del solape correspondería con un tercio de 512 píxeles, es decir, unos 170 píxeles.

```
1     def cut_image(image, overlap):
2         cells = []
3         grid_width, grid_height = image.size
4
5         for h in range(1, int(grid_height/overlap) - 1):
6             for w in range(1, int(grid_width/overlap) - 1):
7                 left = overlap * (w - 1)
8                 right = left + (overlap * 3)
9                 top = overlap * (h - 1)
10                bot = top + (overlap * 3)
11
12                crop_img = image.crop((left, top, right, bot))
13
14                cells.append(crop_img)
15
16        reutrn cells
```

Listing 1: Código que genera una matriz de imágenes recortadas a partir de una original dada

4.8. Post-procesamiento: recomposición de la imagen

Este sistema de descomposición de la imagen en fragmentos de la misma nos obliga a hacer un cierto post-procesado. Al detector se le proporcionan cada una de las divisiones de la imagen original para que las evalúe por separado. Por tanto, debemos crear un sistema que unifique todas esas detecciones para obtener una clasificación en base a la imagen original. En otras palabras, hemos de elaborar un sistema encargado de recoger todas las detecciones por separado y plasmarlas en la imagen original. A continuación se detallarán todas las circunstancias que se tuvieron que tener en cuenta para completar esta minuciosa tarea.

Hay que recordar que, como ya se mencionó anteriormente, tras la detección de cada imagen la red nos devuelve otra en la que aparecen marcados, por medio de formas rectangulares, los elementos detectados en la misma. Esta imagen tiene el mismo nombre que la imagen proporcionada. Además, nos proporciona un único fichero CSV en el que vienen anotadas todas las detecciones realizadas. En este archivo se indica para cada detección, el nombre de la imagen en la que se realizó esta predicción, la posición y el tamaño de la caja que delimita ese elemento detectado, la clase a la que pertenece este objeto y la confianza de la red expresada en número decimal de 0 a 1.

4.8.1. Eliminación de duplicados

El sistema de solape permite asegurar que todos los elementos de la imagen original estén presentes de forma completa o prácticamente completa en al menos una de las imágenes que se van a proveer al detector. Sin embargo, esto también provoca que un mismo objeto aparezca en múltiples imágenes si se localiza en una zona de solapamiento, como se muestra en la ilustración 4.10.

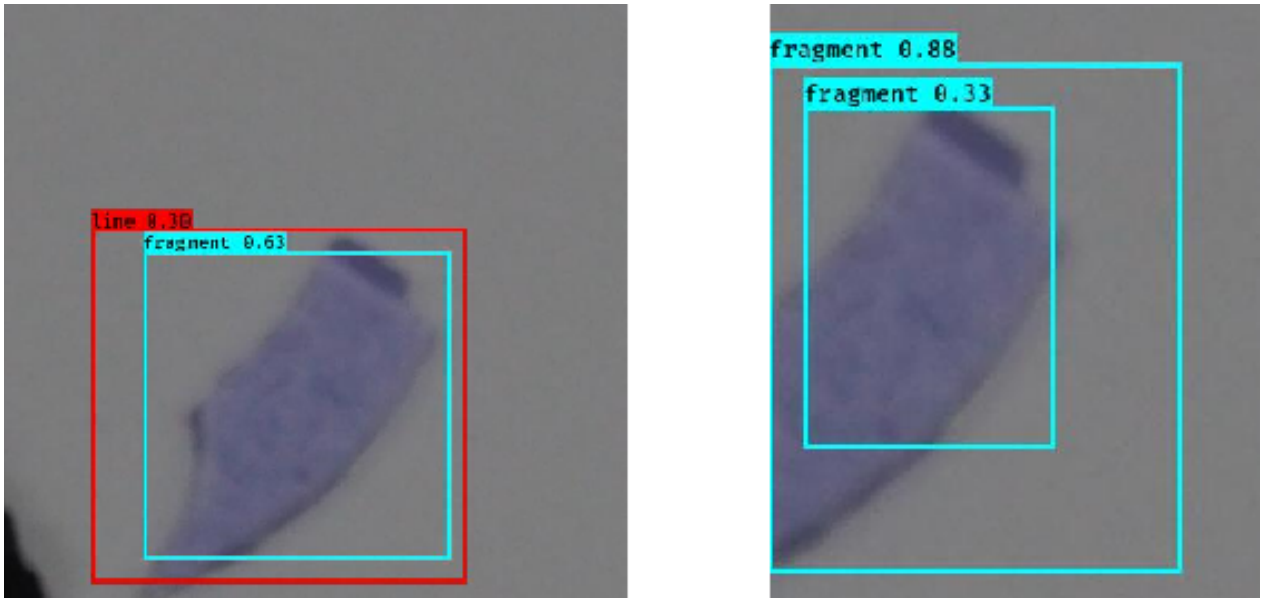

Ilustración 4.10: Visualización de que un mismo fragmento se detecta múltiples veces

Por tanto, al recombinar todas las fracciones de imágenes es imprescindible que se eliminen aquellas detecciones repetidas. Solo se ha de conservar una única detección por cada objeto. De lo contrario se estaría corrompiendo el proceso de detección.

Para resolver este problema se elaboró un sistema de detección de solapes. Este sistema genera de manera provisional todos las cajas delimitadoras de una imagen y los guarda en una lista provisional. A continuación, recorre esa lista y va comparando cada rectángulo de la lista con el resto. En cada comparación se calcula la intersección sobre la unión entre los dos rectángulos, en otras palabras, se calcula el porcentaje del área en la que se solapan los dos rectángulos. Pero para hallarlo, primero se ha de calcular el área de intersección de los dos rectángulos, así como el área de cada uno por separado.

Para realizar esta operación es conveniente simplificar la representación de los rectángulos en cuatro valores, lado izquierdo (l_n), lado derecho (r_n), parte superior (t_n) y parte inferior (b_n). Además, conocemos que el área de intersección entre estas dos figuras es un rectángulo en si mismo. Por tanto, hemos de hallar el ancho y el alto para calcular su área. En la ilustración 4.11 se puede visualizar el planteamiento del problema.

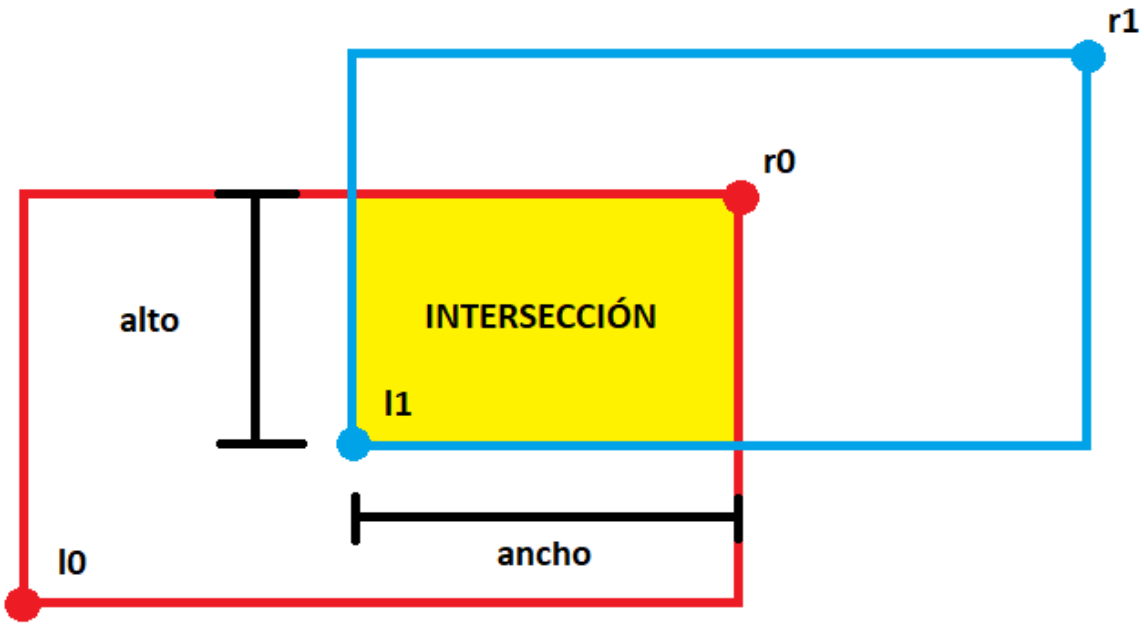

Ilustración 4.11: Visualización del problema del área de intersección

Para hallar el ancho, hemos de encontrar el lateral derecho del rectángulo que se encuentre más hacia la izquierda, es decir, el que tenga un menor valor y restarle lateral izquierdo que se encuentre más a la derecha, es decir, que tenga un valor mayor en el eje X. Por tanto:

$$ancho_{intersect} = min(r_0, r_1) - max(l_0, l_1) \tag{4.5}$$

En el caso de hallar la altura, el procedimiento sería bastante similar. Habría que tomar la parte superior de los dos rectángulos que tenga un menor valor y restarle la parte inferior que tenga un mayor valor, quedando la ecuación de la siguiente manera:

$$alto_{intersect} = min(t_0, t_1) - max(b_0, b_1) \tag{4.6}$$

Una vez tenemos el alto y el ancho, los multiplicamos para obtener el área. Habría que tener en cuenta que si los rectángulos no se solapan, el área resultante puede ser negativa o menor que 0, por lo que habría que añadir cierta estructura de control que nos indicara ese caso.

$$Area_{intersect} = ancho_{intersect} * ancho_{intersect} \tag{4.7}$$

Una vez obtenida el área de intersección, para hallar el porcentaje de dicho área, hemos de dividir el área obtenida entre el área del primer rectángulo (A_0) más el área del segundo (A_1) menos el área del solape ($A_{intersect}$). De esta forma, el cálculo del porcentaje quedaría de la siguiente manera:

$$P_{intersect} = \frac{A_{intersect}}{A_0 + A_1 - A_{intersect}} \tag{4.8}$$

En el caso de que le porcentaje calculado exceda un cierto umbral, establecido en un 10 %, se entiende que se trata de una detección duplicada. Para establecer qué detección se ha de descartar, se escogería aquella con un mayor factor de confianza por parte de la red. Este sistema tan estricto se ha podido idear debido a las características del problema. No se espera que en las fotos que se tomen haya muestras que se superpongan, sino que estarán dispuestas por toda la foto con una cierta separación.

4.8.2. Cálculo de la posición de las cajas delimitadoras

Este supuso el otro gran reto a abordar a la hora de reconstruir la imagen original. Había que reposicionar todos los cuadros delimitadores en la imagen completa. Las medidas que se indican en el CSV son relativas a cada fragmento, por lo que había que encontrar una forma para encontrar la posición absoluta de cada uno.

Por fortuna, se podía controlar el nombre que iba a tener cada fragmento cuando se recortara de la imagen original. Además, estos nombres no eran modificados por el detector y aparecían junto a cada detección en el fichero CSV. Por tanto, se ideó un sistema en el que se incluía en el nombre de cada fragmento, la posición de la imagen original de la que se había obtenido. Además, se incluía el nombre exacto de la imagen original de la que se había tomado cada porción. De esta manera, tras el paso por la red, se podía conocer de qué imagen procedía cada detección, así como su posición.

Por tanto, para cada una de las cajas delimitadoras, únicamente había que iterar sobre el fichero CSV. En cada fila del mismo, se obtenían del nombre del fichero (columna *image*), la posición de la imagen original de la que se extrajo el recorte (*h_offset* y *w_offset*) y la posición de la caja delimitadora relativa al fragmento (*xmin*, *xmax*, *ymin*, *ymax)*. Como se muestra en el extracto 2, estas posiciones se sumaban para obtener la nueva posición en la que se ha de ubicar la caja delimitadora.

```
1     def generate_rectangle(csv_row):
2         filename = csv_row["image"].split(".")[0]
3         [_, h_offset, w_offset] = filename.split("_")
4         xmin = int(w_offset) + int(csv_row["xmin"])
5         xmax = int(w_offset) + int(csv_row["xmax"])
6
7         ymin = int(h_offset) + int(csv_row["ymin"])
8         ymax = int(h_offset) + int(csv_row["ymax"])
9
10        return Rectangle(xmin, ymin, xmax, ymax, int(csv_row["label"]), float(csv_row["confidence"]))
```

Listing 2: Código que genera un rectángulo a partir de una fila de CSV

4.9. Entrenamiento con el nuevo sistema

A pesar de que las ilustraciones mostradas hasta este momento son de fotos de microplásticos, en realidad todavía no se tenía acceso a dichas imágenes. Por tanto, para probar este sistema había que simular dichas imágenes. Aprovechando que el detector había sido entrenado para la detección de caras de perros y gatos, se reprodujeron estos elementos utilizando la imagen de la cara de un gato y la imagen de la cara de un perro. Estas simularían dos clases distintas de microplásticos.

Su selección se determinó en base al hecho de que la arquitectura entrenada podía detectarlas sin problema. No se deseaba que la imagen escogida para simular un microplástico interfiriera en la prueba. Por tanto, una imagen en la que se obtuviera una detección con bajo valor de confianza no podía ser utilizada para el ensayo. De haberla utilizado no se podría conocer si algunos fallos en las detecciones se debían a la propia imagen o a un fallo en el proceso. Por tanto, las imágenes escogidas, que aparecen en la ilustración 4.12, tenían que tener un valor en la confianza mayor que el 95 %. En este caso, tienen un increíble valor del 98 %.

Ilustración 4.12: Imágenes empleadas para simular los microplásticos. Fuentes: [52], [36]

Una vez seleccionadas las representaciones de dos clases distintas de "microplásticos", faltaría generar una imagen que simule una foto tomada de una muestra de los mismos. Normalmente una muestra se se sitúa sobre un folio en blanco, por ello podemos asumir que el fondo de esta imagen ficticia ha de ser blanco. Además, conocemos las dimensiones y posicionamiento de los elementos por la imagen. Para ello se elaboró un simple algoritmo que depositara en zonas aleatorias de un fondo blanco de $5000 * 5000$ píxeles las imágenes mostradas en la ilustración 4.13.

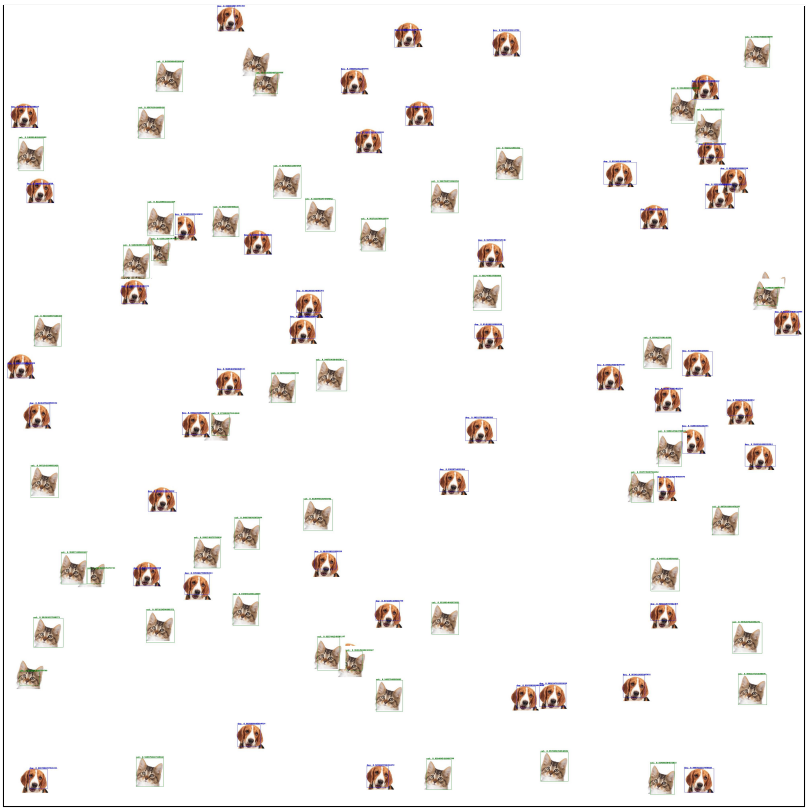

Ilustración 4.13: Imagen que simula una foto de una muestra de microplásticos

Como se puede observar en la figura 4.13, el resultado parece representar con bastante exactitud la realidad. No obstante, hay que mencionar que no se deseó controlar de manera deliberada la superposición de dos elementos. A pesar de que esto no es fiel a la realidad,

se consideró que era innecesario dada la abundancia de objetos en la imagen. Además, el control de este hecho incrementaría la complejidad del algoritmo y llevaría bastante tiempo. Por tanto, se tomó la decisión de que al realizar la evaluación de los resultados simplemente se descartarían estos casos. El resultado de la detección, como se muestra en la ilustración 4.13, es bastante bueno. Se consiguen detectar la totalidad de las *muestras*.

Tras el éxito obtenido en estas pruebas preliminares, se pasó a la evaluación completa del sistema con imágenes reales, como se expone en la sección 5. Gracias a esta evaluación se pudieron realizar los ajustes pertinentes hasta conseguir un resultado admisible.

4.10. Despliegue del modelo

Una vez se obtuvo una configuración que arrojara valores aceptables, se pasó a la elaboración de un sistema que facilite el uso de esta aplicación por otros usuarios. En un principio se tenía la idea de desplegar este modelo en un servidor web que funcione junto con un panel, también web. De esta manera, el detector puede ser usado por cualquier persona en cualquier región del mundo.

Sin embargo, se temía que la elaboración de este sistema impidiera la entrega de este proyecto en los plazos deseados. Por ello se comenzó por la elaboración de un prototipo simple que funcionara mediante comandos en una consola. Aún así, este prototipo ha de estar preparado para su uso por personas que no tienen un conocimiento informático elevado. Por ello se quiso simplificar la interfaz de esta aplicación a un único comando.

En otras palabras, con la ejecución de un único comando, el sistema debería de obtener una imagen o un conjunto de imágenes, procesarlas y generar, por cada una de ellas, dos ficheros: un archivo CSV en el que se anoten todas la información relativa a cada detección realizada en la imagen y otra fotografía en la que se indique con una caja delimitadora y una etiqueta cada una de dichas detecciones, como se muestra en la ilustración 5.3. Además, estas cajas seguirían un código de colores que determinaría el tipo de elemento que se ha detectado.

Para mantener este proceso lo mas sencillo posible, se otorgarían al usuario dos carpetas: una carpeta *in* y una carpeta *out*. El usuario ha de colocar todas las imágenes sobre las que quiera realizar una detección en la carpeta *in*. Una vez iniciado el programa, este se encargara de dividir cada una de las imágenes en fragmentos, así como guardarlos en una carpeta que crea temporalmente, llamada *tmp*. Esta carpeta se borraría al concluir el proceso.

A continuación, la aplicación pasa todos los fragmentos por el detector y obtiene en un fichero CSV las detecciones preliminares, que también se guardan en la carpeta *tmp*. Para la correcta organización interna del programa, se ha de mencionar que dentro de la carpeta temporal crea un subdirectorio para cada imagen. Esto facilita el proceso de post-procesado que seguiría a la etapa de detección. En este último paso se reconfiguran las detecciones para hacerlas relativas a la imagen original y se guardan todos los cambios en la carpeta *out*. De esta manera el usuario obtiene en el directorio *out* todo lo esperado.

A pesar de actuar casi como un *script* pues no tiene comandos, este prototipo cuenta con un manual de usuario donde se describe su instalación y uso de forma detallada para que cualquiera pueda realizar las detecciones. Este manual se encuentra junto con el código en un repositorio [54] público y accesible para todo el mundo.

Capítulo 5

Evaluación

Este capitulo se corresponde con la evaluación del sistema con imágenes de prueba. La valoración del modelo se realizó en dos ocasiones. En primer lugar, se efectuó un examen para comprobar que el rendimiento y precisión mostradas con las imágenes artificiales no sufría muchas pérdidas al usar imágenes reales. Es decir, se quiso confirmar que las hipótesis planteadas a lo largo del desarrollo eran correctas, cuando no se tenía acceso a las fotografías y se tuvieron que simular de forma artificial.

A continuación, tuvo lugar una segunda evaluación tras hacer una serie de correcciones acordes a los resultados arrojados en un primer lugar. No es recomendable realizar demasiadas modificaciones a un modelo para no caer en un sobreajuste a los datos de entrenamiento. Por ello, únicamente se quiso hacer un segundo examen para solucionar aquellos aspectos que durante el desarrollo del trabajo no se contemplaron o se contemplaron de forma errónea dada la ausencia de estas fotos.

5.1. Naturaleza de los datos

Antes de entrar en la mera evaluación, se considera necesaria una introducción previa de las características de los datos para entender, de esta forma, los resultados y decisiones que se toman en el último ajuste.

Con respecto a los aspectos generales de estas fotografías, se ha de destacar su tamaño y procedencia. Como ya se mencionó anteriormente, son imágenes de unos 4000 píxeles de ancho y 3500 de alto. Además, son imágenes de alta resolución. Para cada muestra se tomaron fotos con dos modelos diferentes de cámaras digitales: la Olympus E-M10MarkII [33] y la Sony [49]. Además, entre las dos cámaras se aprecian ciertas diferencias con respecto a la iluminación. Esto fue a propósito, para comprobar que la arquitectura no presentaba variaciones significativas pese a cambios en las características de las mismas, como la cámara digital o la iluminación.

El objetivo inicial de este trabajo era la detección de 5 clases distintas de partículas:

fragment, *pellet*, *line*, *tar* y *organic*. Su traducción al español sería fragmento, bolitas, lineas o hilos, alquitrán y orgánico, respectivamente. Estos elementos se recogen de una parcela de arena de $50cm * 50cm$ y se pasan por un proceso de limpieza y selección para eliminar los elementos que no se desean examinar. Una vez se filtran todos los elementos no deseados, se disponen con cierta separación a lo largo de un folio en blanco tamaño DIN A4 y se toma la foto de dicho folio con los elementos separados. Un ejemplo de estas fotos puede verse en la figura 5.1.

Ilustración 5.1: Ejemplos de imágenes tomadas sobre la misma muestra con las cámaras Olympus *(dcha)* y Sony *(izda)*

Los elementos de cada una de las clases presenta características propia. A continuación se expondrán dichos aspectos, pues, sin su especificación, no se puede conocer realmente la naturaleza de los mismos.

- **Línea o Hilo:** se corresponde con los pequeños segmentos de nylon procedentes de las redes de pesca o los sedales. Se tratan, por tanto, de elementos largos pero finos que suelen tener un color verdoso o azulado.
- **Bolitas:** son aquellas partículas de microplásticos que presentan mayor erosión. Es por ello que su forma es comúnmente redonda y por lo general tiene un color *beige* o marrón claro.
- **Alquitrán:** no se trata de un microplástico, pero si se trata de un residuo que los investigadores desean tener en cuenta. Presenta una forma irregular pero un color negro propio de este elemento químico.
- **orgánico:** esta clase la forman aquellos elementos de tienen procedencia orgánica, como huesos, algas o pequeñas conchas. No son objeto de estudio pero interesa tenerlas presentes como elementos que no se desean detectar y que se descartarán en un futuro.
- **Fragmentos:** las porciones que se desprenden de grandes elementos de plástico debido a la erosión de los mismos entran dentro de esta categoría. No presentan un color ni forma característica ya que son bastantes irregulares en estos aspectos. Sin embargo,

se puede decir que esa falta de algún elemento característico es, en efecto, algo que los define.

Por último, se ha de decir que cada imagen contiene elementos únicamente de una clase. En el nombre de la imagen se indica la clase a la que pertenecen todos los elementos de la misma. Estos elementos han sido clasificados previamente por un experto por lo que facilita enormemente el proceso de etiquetado y evaluación.

5.2. Evaluación preliminar

Hay que mencionar que se trata de un conjunto reducido de unas 17 imágenes, aproximadamente 3 imágenes por cada una de las clases a detectar. Recordemos que las imágenes que se van a proveer al detector son porciones de 512 píxeles de tamaño. Por ello, las fotografías utilizadas para el entrenamiento han de ser fragmentos de cada imagen. Concretamente se han de generar 100 fragmentos de cada clase para realizar el proceso de aprendizaje de la red.

5.2.1. Entrenamiento y evaluación de 2 clases

Aparentemente el sistema debería de estar preparado para funcionar correctamente, no obstante se prefirió realizar un entrenamiento preliminar con dos clases: fragmento y hilo. Esto se debe a que, por cada entrenamiento, se han de etiquetar de forma manual cada una de las imágenes del conjunto. Se trata de un proceso tedioso que requiere una enorme inversión de tiempo. Por ello, se decidió realizar un primer entrenamiento con solo dos clases a modo de comprobar que el sistema funcionaba en casos reales.

Para la generación de imágenes se elaboró un pequeño *script* en python que segmenta la imagen y guarda los recortes en una carpeta determinada. Posteriormente, se seleccionan de forma aleatoria 110 imágenes de cada clase y se guardan en el directorio pertinente. Se selecciona un 10 % por encima de la cantidad requerida para realizar el entrenamiento ya que se observó que en múltiples ocasiones no aparecen objetos a detectar en un recorte, como se puede observar en la ilustración 4.7. De esta manera se consigue un conjunto de entrenamiento heterogéneo con la cantidad de imágenes requerida y en el que no ha habido intervención humana.

Además, en esta ocasión el etiquetado de las imágenes se trató de forma minuciosa. Había que evitar etiquetar objetos que se encuentren cortados, como se muestra en la ilustración 5.2. La forma de estos objetos no refleja la realidad. Además de que en estos casos no se ve la forma completa del mismo, uno de los lados va a ser perfectamente recto debido al corte. Esto puede provocar que la red entienda erróneamente que los objetos de una determinada clase han de ser de tener un lado recto, cuando en la realidad no es así.

Ilustración 5.2: Ejemplo de imágenes que no se etiquetaron deliberadamente

Una vez fueron etiquetadas todas y cada una de las imágenes, se realizó el entrenamiento de la red. Como se muestra en la ilustración 5.3, el resultado de este entrenamiento es significativamente satisfactorio. Todos los elementos de la imagen se detectan correctamente y con una confianza bastante alta. Si bien hay algunos falsos positivos, se trata de unos casos mínimos.

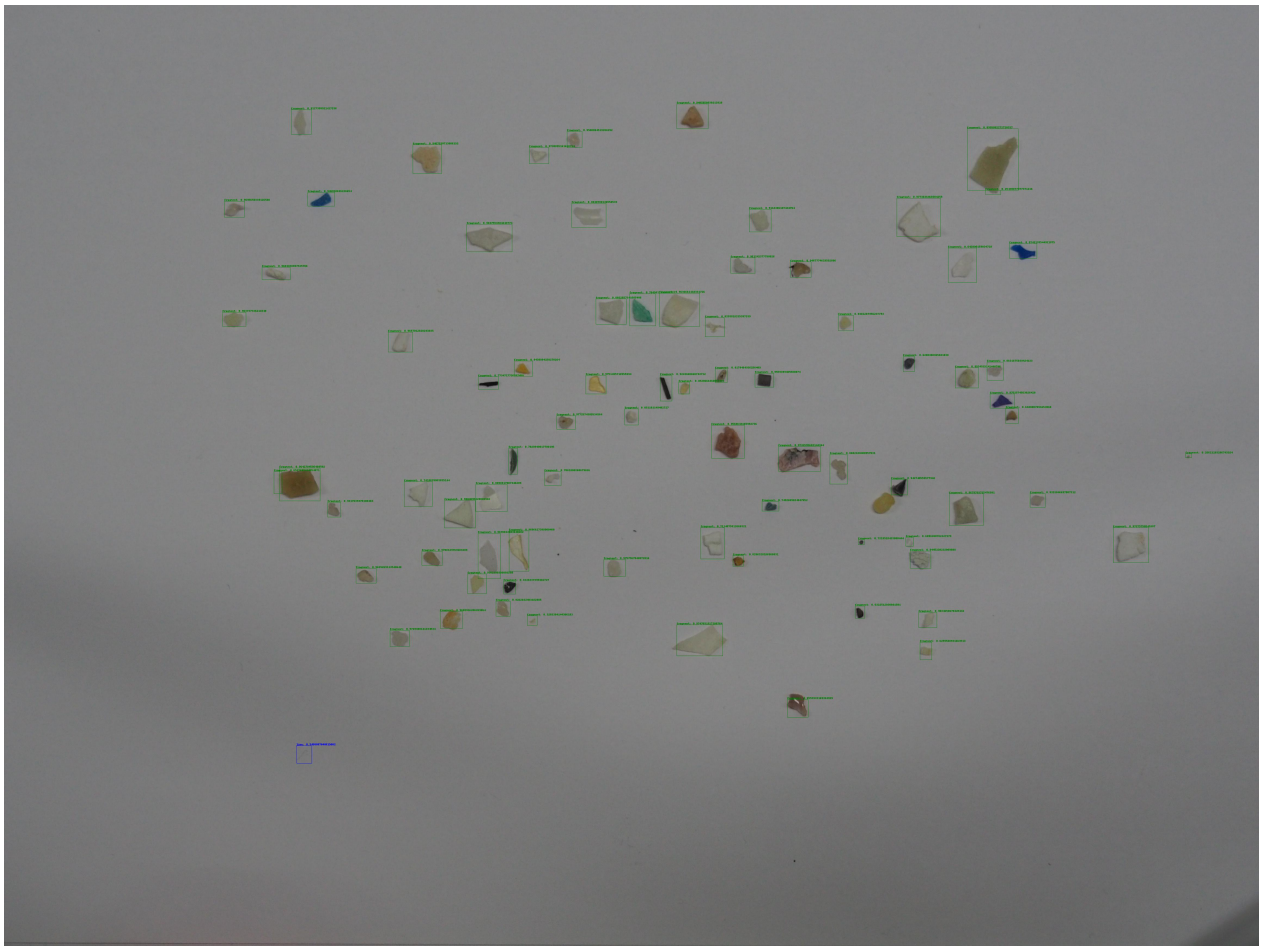

Ilustración 5.3: Resultado de la detección de una imagen de elementos de tipo fragmento

5.2.2. Entrenamiento y evaluación con 5 clases

Al comprobarse la eficacia del sistema incluso con imágenes de microplásticos, se procedió al entrenamiento de la totalidad de las clases. Para ello se volvió a generar un nuevo conjunto de entrenamiento en el que se incluían las 5 clases.

Una vez más, se volvió a generar de forma aleatoria un conjunto de entrenamiento de 110 porciones recortadas de las imágenes seleccionadas para este proceso. Posteriormente se pasó al etiquetado manual de esas 550 fotos, procurando etiquetar únicamente los elementos que estuvieran completamente representados.

A continuación, tuvo lugar el entrenamiento de todas estas imágenes. Debido a que el conjunto de entrenamiento poseía un mayor número de elementos, este proceso fue más extenso de lo normal. Una vez finalizado el entrenamiento, se precedió a la evaluación de los resultados. Para ello se pasó por la arquitectura las imágenes que se habían reservado para esta ocasión. Una vez finalizada la detección, se realizó un pequeño examen visual para comprobar la calidad de las predicciones.

Tras este somero análisis preliminar, se pudo percibir que la detección no había ido como se presuponía. Hay que recordar que se conoce que todos los objetos que se encuentran en una imagen pertenecen a una única clase, que viene indicada en el nombre la misma. En la ilustración 5.4 se puede observar como aparecen etiquetadas múltiples clases. Esto se puede observar ya que las cajas delimitadoras de la red adoptan un cierto color dependiendo de la clase que detecte. En esta ocasión, todas deberían de ser de color verde, correspondiente a la clase fragmento, sin embargo aparecen colores del resto de clases.

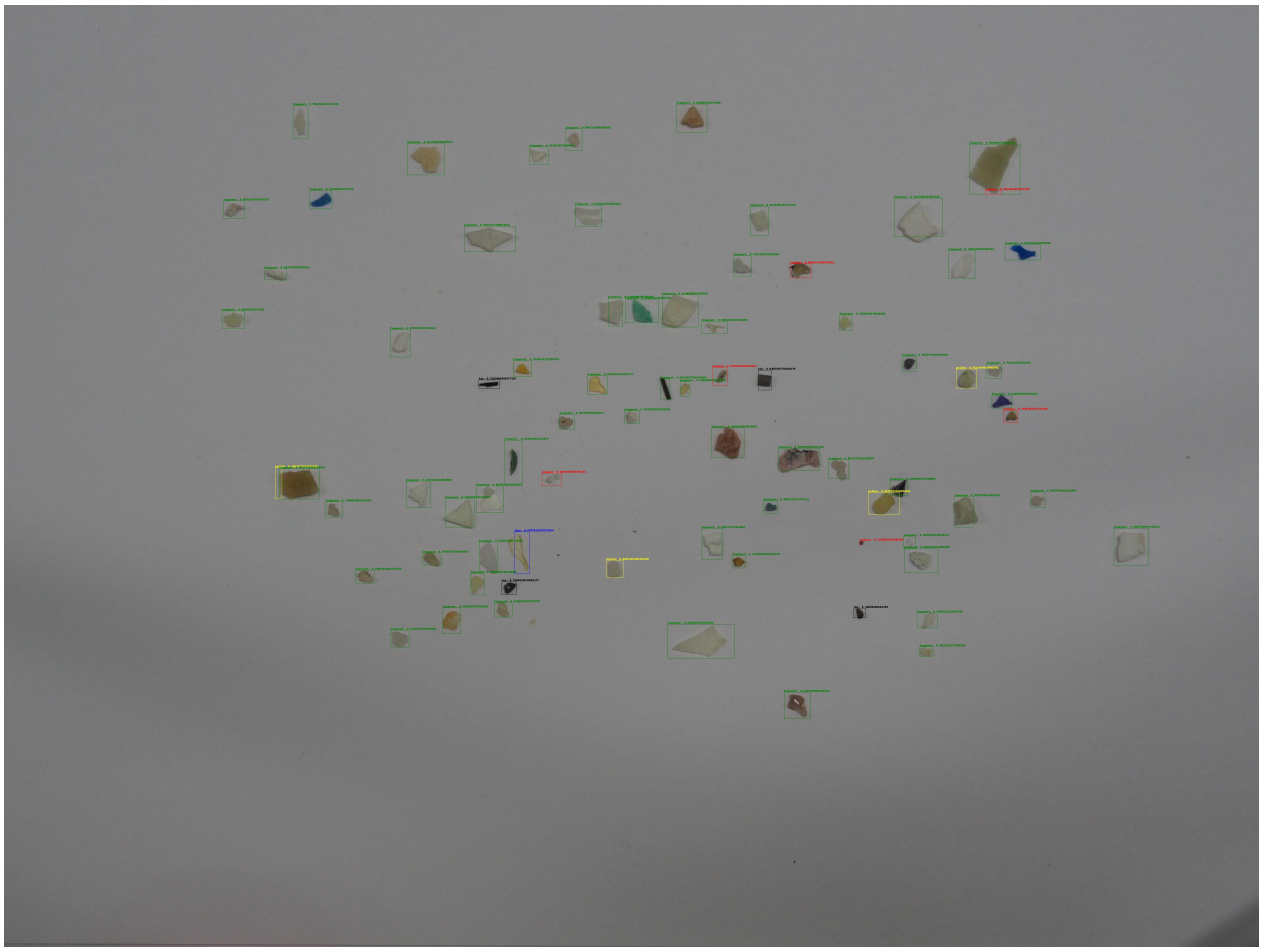

Ilustración 5.4: Resultado de la detección de una imagen de fragmento con todas las clases

Dado que el análisis visual arrojó múltiples resultados desfavorables, era necesario realizar un segundo análisis más estricto. Para ello se elaboró una pequeña aplicación que desgranara las detecciones de cada clase. En la figura 5.1 se pueden observar el resultado de este análisis más preciso. A partir de este análisis se pueden sacar diversas conclusiones, que comentaremos más adelante.

Para la realización de este análisis se extrajeron varias métricas a partir de los datos que nos devuelve el detector en un CSV. Estas medidas son relativas a cada una de las imágenes, por lo que cada fila de la tabla 5.1 se corresponde con una imagen. Además se decidió tomar métricas de cada clase para cada imagen. Estas métricas son las siguientes:

- **Total (T):** Número de elementos de un cierto tipo que se encuentran en la imagen.
- **Porcentaje (%):** Porcentaje de elementos que se encuentran en la imagen que son de una cierta categoría.
- **Mean confidence (mC):** Valor medio de la precisión que arroja la red para la detección de todos los elementos de un cierto tipo.

Además, se obtiene el número de elementos que consigue detectar YOLO, así como el

número de elementos que debería de detectar la red. De esta manera podemos conocer cuantos falsos positivos y falsos negativos arroja el sistema. Como recordatorio, ambos se obtienen al restar el número de elementos que detecta la red menos el número de elementos que realmente aparecen en la imagen. Si el valor obtenido es positivo, el resultado se corresponde con el número de falsos positivos, pues la red detecta más elementos de los que debería. En cambio, si el número es negativo, se trata de falsos negativos, ya que faltan elementos por detectar.

En el estudio de la precisión 4.1.1.1 se hace referencia a estos valores, sin embargo hay que tener en cuenta que se trata de medidas un tanto inexactas en este caso, ya que el número de elementos que detecta la red puede verse afectado por las detecciones múltiples. Es decir, si el número de detecciones sobrantes es igual al número de objetos que no detecta la red puede provocar que se compensen y que, por tanto, el número total de detecciones sea exactamente igual al número de elementos encontrados por la red. Por tanto, se podría estar asumiendo que no se produce error alguno cuando no es así en la realidad.

En este punto del trabajo se desea realizar una evaluación general del sistema para encontrar los errores más graves. Buscar una leve mejora en la precisión del mismo no era el objetivo de esta evaluación, ya que los valores que arrojaba la red en la clasificación eran más que aceptables. Además, recordemos que esta evaluación está complementada con una exploración visual de las detecciones. De esta manera, solo se va usar para medir la precisión en cuanto a las detecciones y poder apreciar posibles valores anómalos con mayor facilidad.

Asimismo, el empleo de esta medida para cuantificar el acierto en la clasificación resultaría improcedente. Esto se debe al hecho de que las pequeñas imprecisiones ya mencionadas pueden verse magnificadas si tratamos de medir con exactitud la correcta clasificación de cada elemento. A continuación se van a exponer las observaciones que se obtuvieron para cada clase en base al análisis visual y al análisis de los datos que se muestran en la figura 5.1.

	fragment			line			organic			pellet			tar			Total	
Clase	**T**	**%**	**mC**	**T**	**%**	**mC**	**T**	**%**	**mC**	**T**	**%**	**mC**	**T**	**%**	**mC**	**YOLO**	**Real**
fragment	61	0,8	0,74	1	0,01	0,67	6	0,07	0,49	4	0,05	0,7	4	0,05	0,68	76	74
tar	1	0,01	0,31	0	0	0	6	0,11	0,56	0	0	0	47	0,87	0,86	54	52
line	0	0	0	74	1	0,86	0	0	0	0	0	0	0	0	0	74	63
line	1	0,01	0,76	77	0,98	0,85	0	0	0	0	0	0	0	0	0	78	58
pellet	3	0,05	0,84	0	0	0	0	0	0	51	0,92	0,87	1	0,01	0,27	55	55
organic	7	0,18	0,56	2	0,05	0,88	20	0,54	0,81	2	0,05	0,43	6	0,16	0,71	37	33

Cuadro 5.1: Resultados de los análisis para el entrenamiento con todas las imágenes

5.2.3. Observaciones fragmento

La clase fragmento es bastante heterogénea, la forma y color de los elementos de este tipo resulta muy variada. Es por ello que es comprensible que el clasificador confunda un elemento fragmento con otro de otra clase. Consecuentemente, YOLO no aparenta tienen problema en detectar todos los elementos en la imagen. Sin embargo si que se producen ciertos fallos en la clasificación. Un 20 % de las mismas son erróneas, siendo la mayoría de estas atribuciones a la clase orgánico.

A pesar de ello, consideramos que un acierto del 80 % en cuanto a la clasificación se trata de un valor más que aceptable. Además, solo se aprecian 2 falsos positivos en lo referente a la detección, por lo que consideramos que en lo referente a la clase bolita, la arquitectura actúa de manera notable.

5.2.4. Observaciones hilo

En un primer momento podemos considerar que los resultados sobre la clase hilo son perfectos. Se obtiene un porcentaje excelente en cuanto a las clasificaciones, del 100 %. Este valor es tan alto que se llevó a cabo una segunda detección gracias a que para esta clase se pudo reservar más de una imagen para el test. Esta segunda detección no dejó lugar a dudas, con un porcentaje del 98 % en cuanto a la clasificación.

En cambio, la detección deja mucho que desear. En ambos casos, el número de elementos que la red encuentra está muy por encima del número de objetos que hay realmente en la imagen. Esta diferencia llega a ser del 34 % en uno de los casos. Esto significa que se están dando falsos positivos ya que la red detecta elementos en lugares donde no hay ninguno.

No obstante, ya antes de realizar la detección se presagiaban estos pésimos resultados. Al examinar las imágenes con elementos de esta clase se percibió que había objetos que eran mucho más grandes de lo esperado. Esto provoca que un mismo elemento llegue a ocupar múltiples celdas, pero que sus detecciones no se solapen, por lo que el sistema puede llegar a atribuir hasta cuatro cajas delimitadoras a un mismo elemento.

Tras encontrarlo, este error fue comunicado a investigadores expertos en la materia. Ese gran tamaño resultaba bastante extraño dado que no coincidía con la idea de un microplástico. Su respuesta ante este dilema fue favorable: coincidían en que los microplásticos no tienen tales dimensiones. Por tanto, se optó por asumir este problema y que fueran los usuarios de la futura aplicación los que filtraran esos falsos positivos de forma manual.

5.2.5. Observaciones bolita

Como se puede observar en la ilustración 5.1, la clase bolita es la que muestra mejores resultados tanto para la detección como para la clasificación. Esta clase posee una forma redondeada característica y unos colores ciertamente llamativos, por lo que consideramos que su detección puede resultar bastante sencilla. No es sorprendente, por tanto, que se

detecten exactamente todos los elementos de la imagen. En otras palabras, no presenta ni falsos positivos ni falsos negativos.

Por otro lado, la clasificación tampoco se queda atrás: presenta una asombrosa precisión del 92 %. Poco más se puede decir en lo referente a esta clase. Si bien la detección puede ser mejor, entendemos que en ocasiones la red puede confundir la forma de un bolita con un fragmento ya que hay algunas muestras de fragmento que presentan características muy similares a un pellet.

5.2.6. Observaciones alquitrán

Si bien esta clase no presenta resultados tan asombrosos como la clase bolita, sus métricas son destacables. Se trata, además, de otra de las categorías que se consideran "fáciles de detectar", ya que todos los elementos de la misma son de color negro. Además, salvo algunas excepciones en la clase orgánico, ninguna otra presenta elementos de dicho color.

Como era de esperar, la detección es sobresaliente. Apenas cuenta con 2 falsos positivos, es decir, con dos detecciones por encima del número de elementos que debería identificar en la imagen. En lo referente a la clasificación, esta también es bastante notable. La imagen con elementos de esta clase presenta una tasa de acierto del 87 % de entre los elementos detectados.

Una vez más, estamos ante una clase en la que YOLO cumple con los buenos resultados que se esperan debido a la facilidad que se tiene en la detección. No se tienen más observaciones para esta clase.

5.2.7. Observaciones orgánico

Esta es, de lejos, la que peores resultados atesora. Se trata de una categoría en la que sus elementos tienen formas bastante complejas y son de gran tamaño. Sin embargo, se considera que la falta de un patrón o un rasgo característico compartido por todos los elementos de este tipo es lo que provocan estos malos resultados.

Aparentemente la detección no es del todo desastrosa, cuenta con 4 detecciones por encima de lo que se debería esperar. Sin embargo, esto empeora bastante en la clasificación. Debido a las características de los elementos antes mencionadas, apenas se obtiene algo más del 50 % de detecciones correctas.

La clase orgánico resulta bastante confusa para la red. Además, al realizar una exploración visual de las imágenes con elementos de este tipo, destacó también las grandes dimensiones de los mismos, como ocurría en el caso de los elementos de clase hilo. Esto nos hace pensar que la detección puede ser incluso peor de lo que refleja la tabla 5.1, ya que se pueden haber hecho múltiples detecciones a un mismo objeto mientras que otros no han sido detectados por parte de la red.

Una vez más se notificó de inmediato al equipo de investigación para idear una solución ante este problema. Su respuesta ante este dilema fue el de descartar esta categoría entre las que ha de detectar YOLO. En realidad a ellos no les interesaba tomar mediciones de este tipo de elementos. En un principio fueron añadidas al conjunto de datos para evitar que el detector que se desarrollara los tuviera en cuenta, no obstante, en la realidad los biólogos eliminan este tipo de partículas en un proceso previo a la detección y el conteo, como se explica en el artículo [27].

5.2.8. Observaciones generales

Por lo general el valor medio de la confianza de la red era suficientemente alta como para considerar que los aciertos eran seguros. Por otro lado, se observó que en algunas de las clasificaciones erróneas, el valor medio de la confianza era algo bajo. Por tanto se decidió que en la siguiente versión se iba a establecer un umbral que determine si una detección por parte de la red merece tenerse en cuenta. Teniendo en cuenta el alto valor en el caso de los aciertos, se decidió establecer este umbral en un 50 %. En otras palabras, todas aquellas detecciones y clasificaciones que tengan un valor de confianza menor al 50 % no se iban a tener en cuenta. De esta manera se intentan corregir estas detecciones espurias.

5.3. Entrenamiento definitivo

Por lo general, tras este entrenamiento se descubrió que la red funciona bastante bien salvo para dos categorías: hilo y orgánico. Para ambos casos, idear una solución tendría un alto coste en el proyecto, por lo que se decidió asumir esos errores.

Por un lado, en el caso de los elementos de tipo hilo que fueran demasiado grandes, se estableció que el usuario de la aplicación se compromete que corregir a mano las detecciones extras, ya que desde un principio el sistema fue ideado para la detección de elementos de pequeño tamaño. Por otro lado, en el caso de la clase orgánico, se decidió que se iba a descartar esta clase dentro de las posibles.

Esta segunda decisión supone que la red ha de "*olvidarse*" de esta categoría y centrarse en la detección de las otras cuatro. Por ello, se suprimió del conjunto de entrenamiento del apartado anterior las imágenes relativas a la clase orgánico y se volvió a entrenar YOLO con el nuevo conjunto, preparando a la red únicamente para la clasificación de objetos de cuatro tipos distintos.

Asimismo, aprovechando que en este punto del trabajo se permitía una mayor libertad de movimiento a la población, se pudo acceder al laboratorio donde se encontraban más muestras de microplásticos. Esto permitió la elaboración un nuevo conjunto de imágenes tomadas con dos cámaras distintas para evaluar de forma efectiva la arquitectura. Una vez entrenado el modelo, se pasaron las nuevas fotografías por el detector. El resultado de estas predicciones viene reflejada en las tablas 5.2 y 5.3.

Como se puede observar en dichas tablas, la precisión general de la red mejora de forma significativa. La eliminación de la clase orgánico y la inclusión de un umbral evitan las detecciones espurias que, en muchas ocasiones, contaban con un valor de confianza bajo. También se ha de mencionar que se aprecian ligeras diferencias entre los resultados de la misma imagen tomada con una cámara diferente. No obstante, las conclusiones generales son que esta última versión es bastante robusta: se puede apreciar una reducción de falsos positivos en la detección, así como un aumento en las clasificaciones correctas.

	fragment			**line**			**pellet**			**tar**			**Total**	
Clase	**T**	**%**	**mC**	**T**	**%**	**mC**	**T**	**%**	**mC**	**T**	**%**	**mC**	**YOLO**	**Real**
fragment	27	0,77	0,91	2	0,06	0,6	2	0,06	0,91	4	0,11	0,78	35	33
fragment	23	0,82	0,91	1	0,04	0,89	0	0	0	4	0,14	0,81	28	26
line	1	0,03	0,91	29	0,96	0,88	0	0	0	0	0	0	30	11
pellet	3	0,1	0,92	1	0,03	0,73	27	0,87	0,92	0	0	0	31	30
tar	0	0	0	0	0	0	0	0	0	24	1	0,92	24	23

Cuadro 5.2: Resultados tras el entrenamiento final en la detección de imágenes tomadas con una cámara Sony

	fragment			**line**			**pellet**			**tar**			**Total**	
Clase	**T**	**%**	**mC**	**T**	**%**	**mC**	**T**	**%**	**mC**	**T**	**%**	**mC**	**YOLO**	**Real**
fragment	32	0,88	0,93	0	0	0	2	0,05	0,73	2	0,05	0,089	36	33
fragment	25	0,89	0,84	2	0,07	0,68	0	0	0	1	0,03	0,55	28	26
line	1	0,04	0,65	24	0,96	0,89	0	0	0	0	0	0	25	11
pellet	2	0,06	0,72	0	0	0	28	0,93	0,83	0	0	0	30	30
tar	0	0	0	0	0	0	0	0	0	24	1	0,92	24	23

Cuadro 5.3: Resultados tras el entrenamiento final en la detección de imágenes tomadas con una cámara Olympus

Capítulo 6

Conclusiones y trabajo futuro

En esta sección se realizará una valoración general del trabajo realizado. Esto incluye la evaluación del proyecto teniendo en cuenta las metas fijadas al comienzo del mismo. Este análisis comprende el reconocimiento de aquellos objetivos que hayan sido cumplidos, los requisitos que no hayan sido satisfechos y los posibles siguientes pasos a realizar a partir de los resultados obtenidos. Asimismo, se examinarán las razones que motivan los mencionados resultados.

6.1. Conclusiones generales

Para realizar una evaluación de este trabajo, primero hemos de recordar los objetivos de esta aplicación. Por un lado, se esperaba realizar una evaluación sobre las técnicas basadas en redes neuronales profundas para su aplicación en el recuento y clasificación de plásticos de pequeño tamaño. Por otro lado, se deseaba obtener al final del proceso un prototipo básico de una aplicación para realizar estas labores empleando la arquitectura que mejor se ajuste a este problema.

A niveles generales, se entiende que los objetivos iniciales del proyecto han sido satisfechos. Se considera que el análisis sobre el estado de la tecnología de este ámbito ha sido suficientemente extenso, como se muestra en el capítulo 2. A su vez, se ha elaborado una correcta comparación de las distintas arquitecturas basadas en redes neuronales. Además, se han evaluado las distintas características de cada red teniendo en cuenta la importancia que tiene cada una en conseguir la detección de esos elementos. Esto es, por ejemplo, que las arquitecturas puedan detectar fácilmente objetos de pequeño tamaño o que las detecciones sean lo más veloces posibles.

Por otra parte, también se ha conseguido desarrollar un prototipo que, usando la arquitectura que mejor se ajusta al problema, permite la detección de los microplásticos en una imagen tomada con una cámara digital. También es cierto que, a pesar de su correcto funcionamiento, se considera todavía un mero prototipo. No obstante, la obtención de un programa aún estando en sus fases iniciales era el otro fin del trabajo.

Por todo esto, podemos considerar que se han llegado a las metas propuestas en un inicio. Aún así, existe cierto resquemor pues al comienzo del proyecto se pretendía elaborar más de lo esperado. No obstante, se entiende no haber podido llegar a un paso más allá debido al terrible contratiempo imprevisto que fue la pandemia del COVID-19.

6.2. Comparación con el proyecto anterior

Como ya se menciona anteriormente, este proyecto pretende ser una mejora de otro realizado con anterioridad [27]. Por tanto, para obtener una evaluación completa de este trabajo, es necesario realizar una comparativa de los resultados obtenidos en ambos casos. Concretamente, se desea poner frente a frente la velocidad y la precisión de la red.

	Tiempo deteción (HH:MM:SS)
Muestra 1	00:03:40
Muestra 2	00:03:36
Muestra 3	00:03:36
Muestra 4	00:03:29
Muestra 5	00:02:22
Muestra 6	00:02:18
Muestra 7	00:02:19
Muestra 8	00:02:17
Tiempo medio	00:02:57

Cuadro 6.1: Resultados tiempos de detección para múltiples muestras con YOLO (CPU)

En primer lugar, se realizó una medición de los tiempos de detección para varias muestras, así como el cálculo del tiempo medio que se tarda en completar este proceso. Los resultados de esta evaluación se muestran en la figura 6.1. De esta forma, podemos hacer una comparativa con las métricas del estudio anterior, que se muestran en las figuras 1.1 y 1.1. Para facilitar la lectura, se ha resumido esta comparativa en la tabla 6.2. Como se observa en la misma, el nuevo sistema desarrollado en este trabajo mejora con creces las versiones anteriores. Simplemente con la ejecución en CPU del modelo se logra una disminución de más del 50 % con respecto al sistema que utiliza un escáner.

	Tiempo deteción (HH:MM:SS)
Manual	00:20:24
Sistema con Escáner	00:06:15
Sistema con YOLO (CPU)	00:02:57

Cuadro 6.2: Comparación de tiempos medios entre los tres sistemas existentes (CPU)

Lamentablemente, debido a problemas inesperados en la instalación de CUDA en la máquina en la que se tomaron estas medidas, fue imposible obtener los tiempos de duración cuando el modelo utiliza GPU. No obstante, es lógico pensar que si el modelo se ejecutara en la tarjeta gráfica, tardaría menos en realizar la tarea. A pesar de este desafortunado problema, los resultados con CPU son lo suficientemente buenos como para demostrar que este sistema es mucho mejor a sus antecesores.

Por otro lado, en cuanto a la calidad en la detección y la precisión en la clasificación, se puede decir que el sistema es bastante robusto. Como se muestra en las tablas 5.2 y 5.3, salvo para la clase *line*, esta versión presenta una baja cantidad de falsos positivos. Es decir, se detecta un número de elementos aproximado al real. Además, la clasificación es igualmente notable. En general, el porcentaje de elementos categorizados de forma correcta ronda el 90 % entre todas las clases.

Por todo esto, podemos decir que la aplicación desarrollada cumple con todas las expectativas planteadas al inicio del proyecto. Se postula como una seria alternativa a los sistemas

actuales por su gran velocidad y precisión.

6.3. Trabajo Futuro

En esta sección se enumerarán todas las posibles mejoras que se idearon para este sistema, pero que por diversos motivos no se llegaron a incluir en el trabajo. Estas ideas pueden ser tanto cambios en la estructura del proyecto para mejorar su funcionamiento, como la creación de nuevos sistemas que tengan como base el resultado de este trabajo.

6.3.1. Servidor y app

En la recta final de este proyecto se intentó implementar una interfaz gráfica para facilitar el uso de la aplicación. Esto podría considerarse como una primera versión básica de la aplicación, que ya dejaría de ser un prototipo. Pasaría, por tanto, a ser un *MVP, Minimum Viable Product* [45], o Mínimo Producto Viable, es decir, una aplicación que cuenta con las mínimas características básicas pero que puede ser de utilidad. El objetivo de este desarrollo es generar un programa de fácil acceso y manejo, por lo que la arquitectura del sistema debería de estar ideado para cumplirlo. El diseño ideado sería el siguiente.

En primer lugar, el detector estaría alojado en un servidor que seguiría el patrón de API RESTful [4]. De esta forma, la detección pasaría a ser un microservicio bien documentado que podría ser usado por cualquier desarrollador mediante peticiones HTTP [30]. Gracias a esta configuración, se conseguiría la democratización de este método de detección parcialmente.

Para conseguir que cualquier usuario realmente tenga acceso a este programa, se considera necesaria la implementación de una interfaz gráfica que se comunique con este servidor. Para una primera versión, lo más adecuado sería un panel web. Esto permitiría a cualquier usuario que no tenga un perfil informático enviar una fotografía de su ordenador para que el detector la analizara. Como respuesta, el usuario obtendría la imagen con las detecciones marcadas en la misma mediante recuadros. Además, se descargaría automáticamente un fichero CSV con todos los datos relativos a dicha detección. Así se democratizaría realmente el uso de este detector a cualquier persona.

No obstante, se entiende que se puede ir un paso más allá. El verdadero potencial de este proyecto se aprecia con el desarrollo de una aplicación móvil. Así, los usuarios pueden enviar a analizar fotos que saquen directamente con la cámara del teléfono. Incluso se podría realizar el proceso de tal manera que el usuario envía varias imágenes tomadas en ese instante o de la galería y que se guarden en una especie de perfil. De esta forma no se tendría que descargar la respuesta en su móvil, sino que podría consultar la información en la nube y descargar solo aquello que le interesa. En la

6.3.2. Incrementar el tamaño de imagen del detector

Como ya se mencionó en varias ocasiones, el detector de YOLO espera unas imágenes de un cierto tamaño. La solución que se tomó durante el trascurso del trabajo para solventar este problema fue la adaptación de la imagen al tamaño requerido. Una de las principales razones que motivaron la adopción de esta medida fue el escaso número de imágenes para realizar el entrenamiento.

Como se ha explicado anteriormente, dado que para adaptar la imagen a la red se dividía en fragmentos, el entrenamiento de YOLOv3 tenía que ser con dichos fragmentos. Esto permitía resolver dos problemas a la vez ya que no servía únicamente como adaptación de la imagen, sino que también permitía generar un conjunto de imágenes de entrenamiento de un tamaño más que aceptable. Sin embargo, se cree que si se hubieran dispuesto de más muestras, la solución por la que se hubiera optado sería la reconfiguración de la arquitectura. Esta pasaría a permitir una entrada de mayor tamaño y, así, se reducirían los tiempos de detección. Esto sucedería pues la red pasaría de detectar cientos de fragmentos a predecir sobre una única imagen.

Bibliografía

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] aituxiu (2020). Fatkun Batch Downloader.

[3] AntonMu (2019). TrainYourOwnYOLO: Building a Custom Object Detector from Scratch.

[4] Au-Yeung, J. (2020). Best practices for REST API design.

[5] Bharath Ramsundar, R. B. Z. (2018). "TensorFlow for Deep Learning".

[6] Chollet, F. (2017). *Deep Learning with Python*. Manning.

[7] Chollet, F. et al. (2015). Keras.

[8] Dai, J., Li, Y., He, K., and Sun, J. (2016). R-fcn: Object detection via region-based fully convolutional networks.

[9] DRUMOND, C. (2020). Scrum. Aprende a utilizar scrum de la mejor forma.

[10] Facebook (2020). React. Una biblioteca de JavaScript para construir interfaces de usuario.

[11] Foundation, P. S. (2020a). pip - The Python Package Installer.

[12] Foundation, P. S. (2020b). Virtual Environments and Packages.

[13] Foundation, P. S. (2020c). Python.

[14] Girshick, R. (2015). Fast r-cnn.

[15] Greenpeace (2020). Datos sobre la producción de plásticos.

[16] Guay, C. (2018). Should you do one week sprints?

[17] Hui, J. (2018a). mAP (mean Average Precision) for Object Detection.

[18] Hui, J. (2018b). Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3).

[19] Inc, R. H. (2020). ¿Qué es el open source?

[20] Kathuria, A. (2018). What's new in YOLO v3?

[21] Kent Beck, Mike Beedle, A. v. B. A. C. W. C. M. F. J. G. J. H. A. H. R. J. J. K. B. M. R. C. M. S. M. K. S. J. S. D. T. (2001). Principios del manifiesto Ágil.

[22] Lahiri, S. and Ghanta, K. (2009). Artificial neural network model with the parameter tuning assisted by a differential evolution technique: The study of the hold up of the slurry flow in a pipeline. *Chemical Industry and Chemical Engineering Quarterly*, 15.

[23] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551.

[24] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection.

[25] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2014). Microsoft coco: Common objects in context.

[26] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37.

[27] Lorenzo-Navarro, J., Castrillón-Santana, M., Santesarti, E., De Marsico, M., Martínez, I., Raymond, E., Gómez, M., and Herrera, A. (2020). Smacc: A system for microplastics automatic counting and classification. *IEEE Access*, 8:25249–25261.

[28] Microsoft (2020). Visual Object Tagging Tool: An electron app for building end to end Object Detection Models from Images and Videos.

[29] Monteux, A. (2019). LogoHunter: Brand Detection As A Service.

[30] Mozilla (2019). Métodos de petición HTTP.

[31] Muehlemann, A. (2018). How to train your own YOLOv3 detector from scratch.

[32] Oksuz, K., Cam, B. C., Kalkan, S., and Akbas, E. (2019). Imbalance problems in object detection: A review.

[33] Olympus (2020). Camara Olypmus.

[34] Org., E. (2020). Electron Framework.

[35] qqwweee (2018). keras-yolo3.

[36] Queensland, B. R. (2020). Photo of a Dog.

[37] Redmon, J. (2013–2016). Darknet: Open Source Neural Networks in C.

[38] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2015). You only look once: Unified, real-time object detection.

[39] Redmon, J. and Farhadi, A. (2016). Yolo9000: Better, faster, stronger.

[40] Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement.

[41] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks.

[42] Royer, S.-J., Ferrón, S., Wilson, S. T., and Karl, D. M. (2018). Production of methane and ethylene from plastic in the environment. *PLOS ONE*, 13(8):1–13.

[43] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA.

[44] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[45] S-PRO (2018). What is a Minimum Viable Product and How to Build an MVP for Your Startup.

[46] Saha, S. (2018). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way.

[47] Sharma, A. (2017). What is the differences between artificial neural network (computer science) and biological neural network?

[48] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

[49] Sony (2020). Camara Sony.

[50] Szegedy, C., Reed, S., Erhan, D., Anguelov, D., and Ioffe, S. (2014). Scalable, high-quality object detection.

[51] Tsang, S.-H. (2019). Review: RetinaNet — Focal Loss (Object Detection).

[52] Turudu, E. (2020). Photo of Cat.

[53] vision chen (2013). Photo of Dog.

[54] Zarco, B. (2020). Repositorio aplicación prototipo.