Aplicación para clasificación y recuento de microplásticos

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Raúl Vega Falcón

TUTORIZADO POR:
José Javier Lorenzo Navarro
Modesto Fernando Castrillón Santana

Curso 2022-23

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Resumen

Resumen

El uso de plásticos se ha visto incrementado, especialmente, durante las dos últimas décadas en diversas situaciones de nuestra rutina diaria, debido a sus buenas propiedades como material de fabricación para infinidad de objetos. Esto afecta de manera directa a muchos factores ambientales, como el cambio climático, ya que existe una mala gestión global en el tratamiento de estos elementos cuando dejan de ser útiles: es muy probable que la mayoria de los plásticos acaben siendo arrojados al medioambiente.

Este proyecto denominado 'Aplicación para clasificación y recuento de microplásticos', pretende desarrollar una aplicación de escritorio que permita la identificación y recuento de microplásticos a partir de fotos haciendo uso de una red neuronal entrenada e integrada en una interfaz gráfica, producto que ayudará a investigadores debido a su precisión, eficiencia y automatización durante el proceso.

Abstract

Plastic use has been incremented. Especially, in a large number of daily routine situations throughout the last two decades, due to their good properties as an industrial material for an infinity of objects. This affects directly diverse environmental factors, such as climate change, because of bad global management in the treatment of these elements when they finish to be useful: it is expected that most the plastics are more likely to be thrown away by nature.

Through this proyect called 'Application for microplastics' classification and count', has the main goal of develop a desktop application that allows to identify and count microplastics from photos through neural networks trained and integrated in a graphic interface, this product will help investigators because of the functional features it provides, such as precision, efficiency and process automation.

Capítulo 1

Introducción

1.1. Introducción

El uso de plásticos en las últimas décadas se ha visto disparado, debido principalmente a ser concebido como un material utilizado de manera habitual por sus buenas propiedades en diferentes ámbitos de nuestras vidas diarias como por ejemplo, el *packaging* [4]. Esto en contraparte, repercute en la mala gestión existente a la hora de tratar los objetos hechos de este material cuando dejan de ser útiles, 'donde se estima que el 71 % acaba finalmente arrojada al medioambiente' [2], y que 'cada minuto que pasa, se arroja a los océanos una cantidad de plásticos y microplásticos equivalente a la capacidad de un camión de basura'.

Los plásticos que son considerados como un material de descomposición muy lento a largo plazo, se acaban convirtiendo en microplásticos a lo largo de los años bajo efectos físicos, químicos y biológicos. Estas partículas de menos de 5 milímetros, llegan a este punto límite y dejan de descomponerse, no existiendo ninguna forma regular de eliminarlos, teniendo que aplicar diferentes técnicas artificiales para hacerlos desaparecer. Por ejemplo, haciendo uso de tecnología del reactor biólogico de membranas (MBR), considerada como la técnica más eficaz para la eliminación de microplásticos, este sistema separa sólidos del agua mediante membranas de filtración, consiguiendo eficiencias cercanas al 100 % [12].

Por lo tanto, aquellos elementos que acaban en sitios inaccesibles para los seres humanos debido a factores como el viento y corrientes marinas, siguen perdurando en el tiempo hasta su futura recolección y puesta en acción para su desaparición. Todo esto, afecta directamente al cambio climático, temperatura global, cadena trófica, entre otros...

Concretamente, una de las alteraciones producidas es la siguiente: todos los microplásticos que vagan en lugares remotos de los oceános y mares alrededor de la tierra, pueden ser confundidos de manera usual por comida para diferentes especies marinas, hecho que causa la muerte de toda esta biodiversidad por inanición, debido al bloqueo que les produce en el aparato digestivo. Una de estas especies afectadas es el plancton, dividido por fitoplanctons y zooplanctons. Los zooplanctons por un lado, se categorizan por ocupar una de las mayores posiciones dentro de la cadena trófica dentro las dietas de los seres marinos. Los fitoplanctons

por otro lado, 'tienen la capacidad de no solo generar el 70 % oxígeno que respiramos, si no de absorber entre el 30-50 % del CO2 generado por los humanos hacia la atmósfera' [7], a través de la realización de la fotosíntesis. Concluyendo que el principal pulmón del planeta son los planctons, denegando la extendida creencia errónea de que esta posición estaba ocupada por los bosques y selvas de nuestro planeta, acción de la que no dejan de ser partícipes y grandes contribuyentes para el medioambiente.

Aparte de destacar las consecuencias que tiene hacia los planctons y seres marinos, es importante recalcar que toda fauna marina afectada por la ingesta de microplásticos, también puede afectar a la humanidad a través del consumo de peces y/o mariscos intoxicados. Aunque la información sobre los microplásticos en la salud humana es limitada, existen varias hipótesis que indican que los microplásticos pueden tener efectos negativo para los humanos, como causar toxicidad de partículas, lesiones inflamatorias y aumentar riesgos de sufrir cáncer [10].

Una forma de controlar actualmente este problema grave a nivel mundial, es la estimación de la cantidad de microplásticos existente a partir del recuento y la contabilización, labores que se realizan de manera manual gracias a, por ejemplo, campañas que incentivan la recolección de desechos y microplásticos realizadas en muchas playas de nuestro planeta. Pero quizá pueda existir una manera más eficiente de ayudar con este problema causado por nosotros mismos.

1.2. Objetivos

El objetivo principal de este TFT es la creación de una aplicación de escritorio que permita, a partir de imágenes que contengan microplásticos, dar un resultado preciso, automatizado y eficiente del recuento e identificación de esos microplásticos plasmados, todo esto a través de una interfaz gráfica sencilla e intuitiva. Se pretende ceder esta aplicación desarrollada a los investigadores y científicos para facilitar las labores que componen su día en día en esta línea de investigación.

Concretamente, se pretende realizar este proceso con 3 tipos de microplásticos: *fragments*, *pellets* y *lines*. En la figura 1.1, se específica la apariencia de cada uno de los microplásticos a través de imágenes extraídas del conjunto de entrenamiento y validación creado:

De este modo, se podrá tener un control más exhaustivo a nivel global del número de microplásticos que permanecen existentes alrededor del globo terráqueo. Permitiendo el poder identificar con mayor facilidad si las diferentes reformas y leyes aprobadas por diferentes organismos e instituciones responsables, están obteniendo resultados favorables para la mejora del estado actual de este gran problema medioambiental.

Para el desarrollo de dicha aplicación, se ha basado de los siguientes hitos u objetivos parciales que en conjunto conformarán el producto final:

- ✔ Generación de un conjunto de datos a partir de un conjunto de fotografías de microplásticos

- ✔ Entrenamiento de un modelo para segmentación de los microplásticos
- ✔ Diseño e implementación de una aplicación gráfica
- ✔ Integración del modelo entrenado en la aplicación diseñada

De manera simultánea y para el cumplimiento de estos objetivos, se ha prefijado una planificación temporal desde el comienzo, producto de un análisis previo y un estudio de los requisitos necesarios. Concretamente, estos objetivos se presentan en la Tabla 1.1.

Fases	**Duración estimada (horas)**	**Tareas**
Estudio previo / Análisis	40	Análisis de entornos de desarrollo y librerías
		Análisis de herramientas de etiquetados
		Estudio de técnicas y modelos para segmentación de imágenes
Diseño / Desarrollo / Implementación	140	Generación de conjunto de datos etiquetados
		Generación del conjunto de entrenamiento del modelo de segmentación
		Entrenamiento modelo segmentación
		Diseño e implementación interfaz gráfica de la aplicación
		Integración modelo de segmentación en la aplicación
Evaluación / Validación / Prueba	80	Pruebas de rendimiento de la segmentación
		Evaluación de la interfaz de usuario
Documentación / Presentación	40	Redacción de memoria
		Preparación y ensayo de presentación

Tabla 1.1: Planificación temporal

Tras el cumplimiento de estos hitos asignados y del desarrollo del producto, se espera cumplir con las competencias generales recogidas por la asignatura. Dándole especial énfasis en las siguientes:

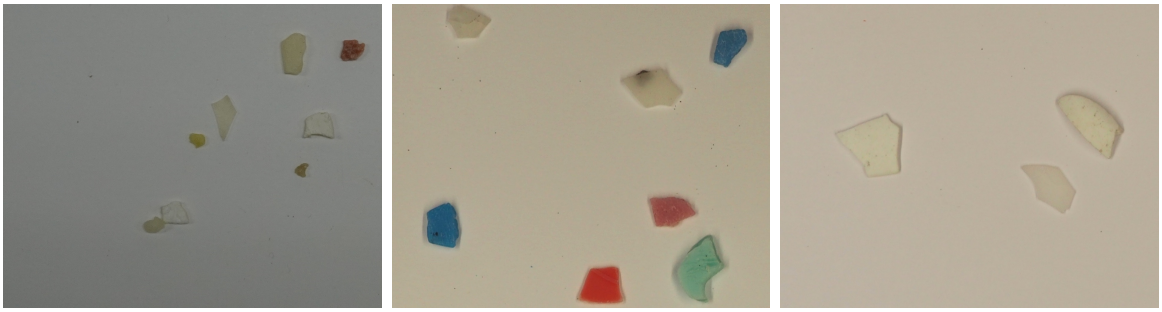
CII015: conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica.

IS03: capacidad de dar solución a problemas de integración en función de las estrategias, estándares y tecnologías disponibles.

IS06: capacidad para diseñar soluciones apropiadas en uno o más dominios de aplicación utilizando métodos de la ingeniería del software que integren aspectos éticos, sociales, legales y económicos

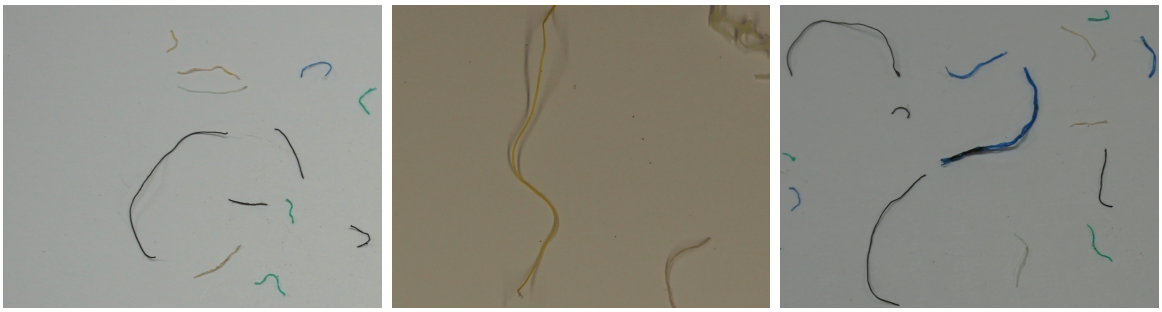
CP04: capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.

Todas estas competencias, contemplan el conocimiento adquirido junto con su puesta en acción y las capacidades que se adquieren, al combinar los diferentes fundamentos y técnicas en los ámbitos de ingeniería del software y de sistemas inteligentes aplicados para el correcto desarrollo de este proyecto.



(a) Fragments: fragmentos formados por la ruptura de desechos plásticos de mayor tamaño

(b) Pellets: gránulos formados a partir de la descomposición de plásticos primarios



(c) Lines: restos de cebos, hilos y de redes de pesca

Figura 1.1: Imágenes de los tres tipos de microplásticos a tratar

Capítulo 2

Estado del arte

'Es lugar común señalar que la crisis ambiental se inicia con la Revolución Industrial, sin embargo su percepción y conceptualización, en tanto problema social y político de carácter global es propio de la historia reciente y su mirada en perspectiva aun es relativamente poco tratada (por lo menos en nuestro país)'. A partir de este relato sacado del artículo denominado 'Antecedentes para una historia del debate político en torno al medio ambiente: la primera socialización de la idea de crisis ambiental (1945 - 1972)', el autor del mismo denota públicamente que la Revolución Industrial es el período donde se empieza a identificar la contaminación medioambiental como un problema global.

Además, es posible afirmar que la conferencia de la Organización de Naciones Unidas (ONU) celebrada en Estocolmo en 1972, puso el sello oficial a la idea de crisis ambientaL. A partir de aquí, se empiezan a ir tomando medidas y aplicando planes de acción hasta la actualidad [3]. Todo ello en busca de paliar los efectos de diferentes acciones que perjudican el medioambiente, entre las que destaca la contaminación por plásticos y microplásticos. En estos momentos, se siguen buscando y aceptando medidas para mitigar las causas de esta crisis medioambiental a través de diferentes organizaciones e instituciones tanto a nivel europeo como internacional, tales como la ONU, Unión Europea, Greenpeace, entre otros muchos...

En la actualidad, existen algunas herramientas desarrolladas que permiten la identificación y recuento de microplásticos de manera automatizada. Las más populares son *ImageJ* [11] y *SMACC* [6]. De esta manera, se pretende aportar un granito de arena mediante la concienciación en los números de residuos que se van manejando periódicamente.

ImageJ es un software de procesamiento de imágenes de código abierto basado en Java usado en una amplia variedad de campos científicos, entre las que se incluye la biomedicina y la ciencia medioambiental. Previa a la identificación de microplásticos, se realiza un proceso de teñido a la muestra con un tinte fluorescente, facilitando la visualización de los microplásticos. Tras esto, la muestra puede ser observada con un microscopio para su futura tipificación a través de las técnicas de análisis de imágenes que proporciona esta herramienta.

Uno de los proyectos más populares que hacen uso de ImageJ es el desarrollado por un conjunto de expertos en estudios ambientales, marinos y químicos en Portugal, con la colaboración

y apoyo del Fondo Social Europeo y Fondo Nacional de Portugal [10]. Donde a partir de un enfoque diferente hasta ese entonces, se tuvo como objetivo "seleccionar el tinte de tinción y la longitud de onda más apropiados para detectar microplásticos de forma selectiva y desarrollar un script de conteo automatizado basado en ImageJ".

MP-VAT, viene acompañado "de un script que establece una escala a partir de un diámetro de filtro conocido, *MP-SCALE*, y un script que permite la configuración del umbral del usuario, *MP-ACT*". Permitiendo en cada caso particular de las muestras, la existencia de parámetros ajustables que permitan una mayor precisión en el proceso de identificación.

MP-VAT que destaca por obtener resultados válidos a partir de métodos relativamente no costosos usados por investigadores con recursos limitados, también presenta ciertos limitaciones/inconvenientes:

- ✔ Necesidad de tinte y secado para cada uno de los elementos de la muestra que se pretende identificar; la parte de secado dura 1 hora.
- ✔ Resultados no suficientemente precisos y fiables.
- ✔ Alta probabilidad de falsos positivos: pueden existir otros elementos en la muestra que contengan de manera innata colores fluorescentes, tales como bacterias.

Seguidamente, tenemos la herramienta *SMACC*, producida en 2020. Consiste en un software para identificar microplásticos a través de muestras de playas tras su pertinente limpieza y filtración, donde se hacen uso de técnicas de visión artificial y aprendizaje automático para discernir entre diferentes tipos de microplásticos extraídos principalmente en zonas costeras de Canarias. Todo esto, a partir del color, la forma y la textura de los mismos.

Concretamente, esta herramienta permite la detección y cuantificación de "pellets", "fragments", "lines", partículas orgánicas y restos de alquitrán. Todo ello, a partir de dos fotos tomadas de la misma muestra: una foto con transparencia; donde la luz incide de manera directa en los elementos y se captura la sombra resultante, y otra foto a color aplicada de manera directa. Obteniendo como mejor resultado una precisión del 91.1 %. Además, se deja como trabajo a futuro, el realizar el proceso de clasificación de microplásticos con el uso y aplicación de redes neuronales convolucionales (CNN), ya que realizando una configuración experimental a partir de un clasificador basado en CNN, se obtuvieron resultados prometedores.

La principal limitación de esta herramienta, es la necesidad de realizar 2 fotos por muestra identificada con formatos específicos: imagen con transparencia aplicada e imagen a color. Además de la capacidad de mejora en resultados a través de la aplicación de CNN.

Por lo tanto, a partir de estas limitaciones de ambas herramientas y aprovechando el trabajo a futuro que se especifica en la herramienta de *SMACC*, se pretende crear una herramienta con el desempeño de una CNN que, *a priori*, aportará resultados más precisos y eliminar la restricción de requerir de 2 imágenes como requisito para la correcta ejecución del identificador y clasificador de los microplásticos, siendo necesario una única foto de la muestra en color en nuestro lugar.

Capítulo 3

Metodología y desarrollo

3.1. Metodología

En cuanto a la metodología, se ha seguido un enfoque en el que se han incorporado de manera selectiva varias características recogidas en la metodología de desarrollo ágil *Scrum*, las cuales nos han podido aportar una mejor planificación o desempeño. Concretamente, se han ido definiendo varios sprints con sus historias de usuarios predefinidas para un plazo aproximado de 1-2 semanas de duración por sprint, periodo que ha podido variar ligeramente en base al número de tareas identificadas totales para cada caso.

Para el proceso de planificación de sprints, junto con la creación de las historias de usuarios y el reconocimiento del esfuerzo de cada una de ellas, se han ido realizando reuniones periódicas con los tutores. En estas reuniones, se introducen las necesidades, requisitos y descripción de cada una de las tareas que se quiere afrontar en cada sprint, traduciéndose en historias de usuarios y se realiza un *Planning Poker* en conjunto con los tutores, para la estimación del esfuerzo con la intención de obtener una mayor organización, planificación y cumplir con los plazos establecidos.

En apoyo a esto, se hace uso de un tablero de historias de usuarios, concretamente, de Trello. Donde se posicionan esas historias de usuario identificadas de cara al futuro tratamiento de las mismas a lo largo del sprint, en este tablero se puede ir actualizando el estado del desarrollo de las mismas a través de la ubicación de estas en diferentes fases establecidas en el espacio de trabajo, de entre las que destacan:

- ✔ Lista de tareas: donde se recopila el conjunto total de historias de usuarios identificadas por nosotros. Simulando el término de Product Backlog en la metodología de Scrum.
- ✔ En proceso: en esta tarjeta, comúnmente conocida como *Work in Progress* o *WIP*, se recogen todas las historias de usuarios que se han dictaminado para ser resueltas durante un sprint en concreto. En este caso, se puede ver esta fase de manera similar al término de Sprint Backlog.
- ✔ QA: en esta fase, reservamos aquellas historias de usuarios que consideramos que están

acabadas pero que requieran de una verificación por parte de los tutores para ser ubicadas en la tarjeta de 'Hecho'.

✔ Hecho: en este apartado, ubicamos las tareas que han sido finalizadas. Con la finalización de todas las tareas inicialmente localizadas en la tarjeta 'En proceso', obtendríamos un incremento. Lo cual resultaría en un paso más cerca del objetivo del producto final.

En cada historia de usuario, se puede añadir información recopilada de posibles avances hasta entonces realizados. Asimismo, tras la finalización de cada sprint, se realiza una reunión retrospectiva donde se comprueba como ha ido el sprint, se evalúa el incremento obtenido, se exponen ideas de mejora a lo ya desarrollado y se realiza una nueva planificación para afrontar el siguiente sprint. En la figura 3.1, apreciamos la distribución de las historias de usuarios ubicadas en sus diferentes fases desde un punto en concreto de la fase de desarollo del proyecto.

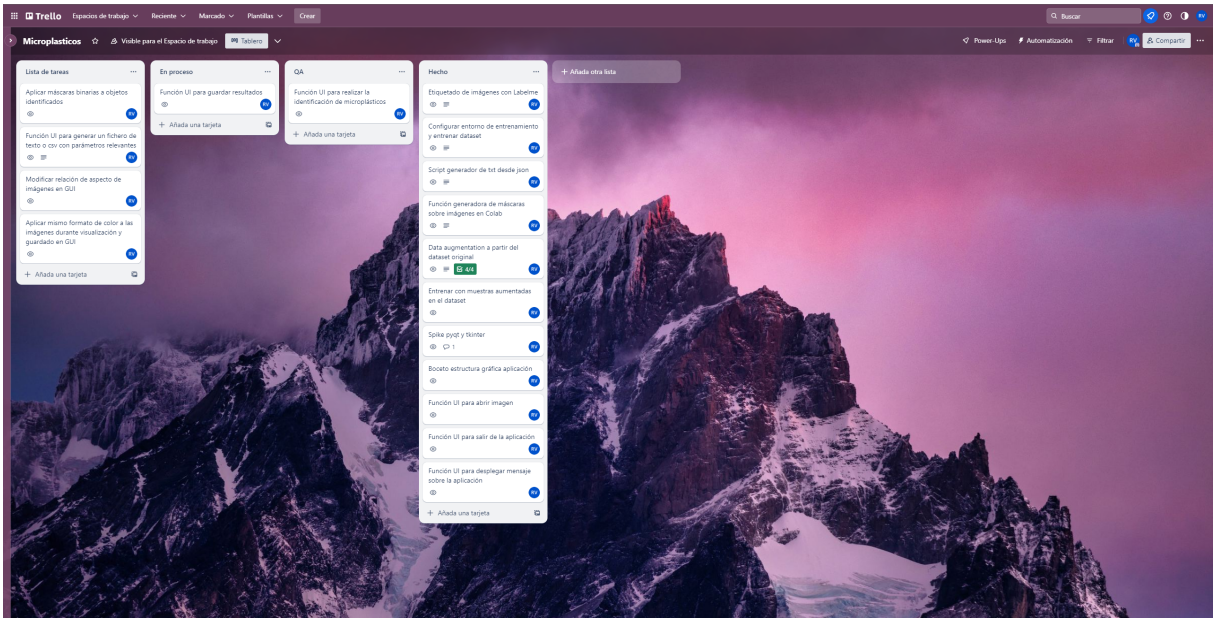


Figura 3.1: Captura de historias de usuarios dispuestas en Trello

De manera relacionada a esto, se ha planteado una disposición basada en estadios que se han ido abarcando a lo largo del periodo de desarrollo: una primera fase enfocada en el *análisis e investigación* de las herramientas que podrían llegar a ser más favorables para el proyecto, junto con la realización de *spikes* para evaluar y discernir entre diferentes recursos a utilizar: lenguajes de programación, entornos de desarrollo y librerías, herramientas de etiquetado y modelos de segmentación para imágenes. Esta fase ha permitido obtener una mejor selección de las tecnologías y los recursos involucrados en cada caso que mejor se adaptan a las futuras situaciones a afrontar.

La segunda fase fue destinada a la *creación y organización de un conjunto de imágenes* de microplásticas para el futuro entrenamiento de la red. Esta fase ha involucrado un proceso de etiquetado de imágenes y el desarrollo de scripts en Python para la automatización de procesos. Tras la finalización de esta fase, se pretende obtener un conjunto de datos completo

para obtener resultados eficientes durante la preparación de la red neuronal.

La tercera fase tiene involucrada todo el proceso del *entrenamiento de la red neuronal*, durante este periodo se han ido realizando múltiples iteraciones de entrenamientos en búsqueda de obtener el mejor resultado posible. Para ello, se ha ido probando con diferentes muestras de datos, hiperparámetros, estructuras del conjunto de muestras, entre otras técnicas.

La cuarta fase es la responsable del *diseño de la interfaz e integración de la red entrenada* en dicha interfaz. Concretamente, hemos hecho uso de una librería que nos proporciona un constructor de interfaces gráficas de usuario basado en componentes, y se hizo uso del repositorio público YOLOv5 ubicado en GitHub. Esta fase busca obtener una interfaz simple e intuitiva que permita a los usuarios hacer uso de la red de manera eficiente y efectiva.

La quinta fase involucrada en el desarrollo de la aplicación va enfocada a la *evaluación de la interfaz gráfica y de la usabilidad de la aplicación*, donde nos ponemos en la piel de los usuarios consumidores del producto final e intentamos realizar mejoras en este sentido. De manera simultánea se realizan pruebas de rendimiento en la red de segmentación usada, con el objetivo de afinar ciertas características que puedan propiciar resultados aún más favorables.

Como última fase planteada, se dedicó el tiempo restante en la *elaboración y redacción de la memoria*, realizando un análisis crítico a partir de la recopilaciópn todas los aspectos relevantes durante el desarrollo del proyecto, junto con sus decisiones tomadas, resultados obtenidos y conclusiones extraídas. Además, también se ha invertido tiempo para la preparación y ensayo de la presentación final.

En la figura 3.2, se refleja una viñeta que resume brevemente todas las fases mencionadas previamente.

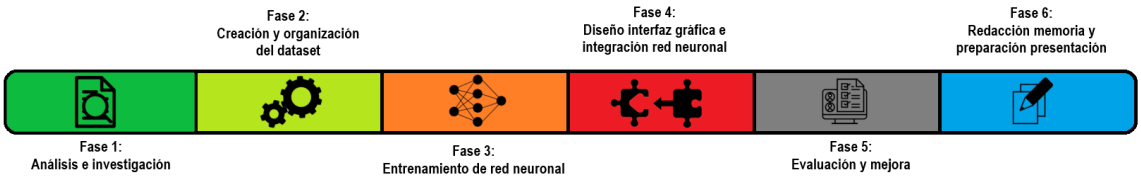


Figura 3.2: Planificación de fases para el desarrollo TFT

3.2. Desarrollo

En esta sección, se profundizará y describirá más técnicamente y en detalle todo el proceso de desarrollo seguido durante el proyecto:

3.2.1. Fase de análisis e investigación

Primeramente para la fase de *análisis e investigación*, se ha barajeado hacer uso de Java o Python como lenguajes de programación. Tras seguir un flujo de desarrollo comparando ambos lenguaje, se ha optado usar Python por encima de Java. Principalmente, por temas de limitación a la hora de integrar la futura red entrenada. El flujo de desarrollo planificado era entrenar el modelo, convertir el modelo entrenado a formato ONNX y finalmente, leer el detector en OpenCV en formato ONNX, esto último para la futura correcta integración y desarrollo en la aplicación en Java. Pero la conversión del modelo entrenado Pytorch a ONNX no admite todas las características y posibilidades necesarias del detector que se pretende utilizar.

Por este motivo, se opta por usar *Python*, cuyo logo se aprecia en la Figura 3.3, como lenguaje de programación para el desarrollo de la aplicación, de scripts para automatizar ciertos procesos y para el entrenamiento del modelo de segmentación. Que además, se posiciona como el lenguaje de programación preferido por científicos y desarrolladores en los campos de Aprendizaje Automático, Inteligencia Artificial y Deep learning, debido a las librerías de cógido abierto que ofrece este lenguaje, como PyTorch, Keras o TensorFlow, que hacen que el proceso de entrenamiento de redes neuronales sea mucho más fácil [8].

Es importante destacar que dentro de las diferentes opciones que se pueden utilizar como modelos para la identificación de elementos, usaremos los *modelos de segmentación* por encima de otros, como los de detección o clasificación. Esto es debido a que a partir de las diferentes imágenes que obtenemos de muestra, nos interesa resaltar las siluetas y áreas de los microplásticos en cuestión, y para esta necesidad los modelos de segmentación son los más indicados.

Figura 3.3: Logo Python

Como herramienta de etiquetado de imágenes, hemos optado por *Labelme*, cuyo logo se aprecia en la Figura 3.4, ejecutado a través del prompt de *Anaconda*: logo reflejado en la Figura 3.5. La razón principal de usar esta herramienta existiendo otras viables como CVAT que también es gratuita, ha sido la capacidad que tiene Labelme de tratar y gestionar muchas imágenes, la cual ofrece una interfaz de usuario, diseñada con la librería *Qt*, en la que se identifica en todo momento las imágenes etiquetadas y aún no etiquetadas. Además, ofrece un flujo secuencial a través de botones para ir navegando por cada una de las imágenes de la carpeta seleccionada. Como único inconveniente, es que el archivo resultante tras el etiquetado de la imagen produce un fichero JSON con las coordenadas de cada polígono dibujado en la imagen que delimitan su contorno, y no un fichero de texto.

Para la generación de diversos scripts encargados de la automatización de ciertos procesos, nos hemos apoyado de Pycharm que es un entorno de desarrollo desarrollado específicamente para Python, la selección de este IDE por encima de otros viene arraigada por la familiarización

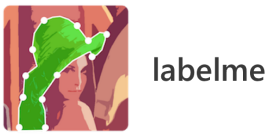


Figura 3.4: Logo Labelme

Figura 3.5: Logo Anaconda

que se tiene debido a su amplio uso en diferentes proyectos en el pasado. El logo de la misma, está ubicada en la Figura 3.6.

Figura 3.6: Logo Pycharm

Dentro del ámbito de las redes neuronales convolucionales, nos encontramos con varias opciones populares como *Faster R-CNN*, *SSD (Single Shot MultiBOX Detector)* y *YOLOv5*, todas estas son capaces de realizar el procesamiento en tiempo real mientras obtienen un ratio de precisión relativamente alta.

Faster R-CNN destaca por su alta capacidad de generar regiones de interés muy precisas en la imagen, debido a su estructura con múltiples etapas. Sacrificando por ello, el tiempo de procesamiento para obtener estos resultados. Además, también posee una complejidad computacional alta. Es usada con frecuencia en aplicaciones de seguridad, donde el nivel de precisión es crítico.

SSD (Single Shot MultiBox Detector), utilizada frecuentemente en aplicaciones que requieren respuestas rápidas, como en sistemas de vigilancia real. Destaca por su velocidad y eficiencia, debido a su enfoque de una sola pasada (single-shot). Además, puede aplicarse para la detección de objetos de diferentes tamaños, donde obtiene resultados válidos a nivel de precisión con solvencia.

YOLOv5, destaca por su alta velocidad de detección de objetos en tiempo real (dispone también de un enfoque de una sola pasada), eficiencia en el uso de recursos computacionales y su facilidad de uso. Como limitaciones, suele sacrificar precisión por eficiencia en momentos donde se requiere mayor detalle.

En nuestro caso y analizando las necesidades de la aplicación, hemos decidido hacer uso de *YOLOV5* para la identificación y recuento de microplásticos. Además, en un estudio desarrollado en Corea y publicado en 2020 donde se realizaba una comparación en el desempeño de Faster R-CNN, SSD y YOLOv4 en el reconocimiento de tipos de vehículos, el modelo

YOLOv4 superó a los otras redes, obteniendo una precisión del 93 % en la identificación de los modelos de vehículos [5]. Teniendo en cuenta a su vez, que YOLOv5 es una versión mejorada de esta, y ofrece mejor desempeño en términos de velocidad y precisión [13]. Esta herramienta puede ser encontrada en su repositorio público almacenado en GitHub, y el logo de la misma se aprecia en la Figura 3.7.

Figura 3.7: Logo YOLOv5

Para la configuración y entrenamiento de esta red neuronal, hemos hecho uso de Google Colab como entorno de desarrollo a través de Google Drive, cuyo logo se puede apreciar en la Figura 3.8. Esta herramienta compatible con Python, tiene la capacidad de combinar en su espacio de trabajo la redacción de texto y ejecución de líneas de código con su respectivo resultado. Además, cuenta con ciertas configuraciones que permite hacer uso de un entorno que ejecute el proceso a través de una GPU almacenada en la nube, para casos en los que no se disponga de un equipo suficiente potente para el procesamiento de estos procesos. Además, se dispone de un conocimiento previo de la herramienta debido a su uso en proyectos previos abarcados.

Figura 3.8: Logo Google Colab

Para el diseño de la interfaz hemos decidido hacer uso de la librería *Pyqt* frente a la librería *Pkinter*. Ambas librerías tienen el objetivo de diseñar interfaces gráficas de usuarios capaces haciendo uso de Python. Pero tras realizar una investigación, *Pyqt*, cuyo logo es apreciado en la Figura 3.9 es la que más se adapta a nuestra situación: dispone de la herramienta *QtDesigner* que permite la creacion de una GUI (Graphic User Interface) de manera automatizada y sin necesidad de escribir código adicional. A su vez, esta herramienta dispone de un amplio abanico de elementos y componentes a integrar. De esta manera, la lógica es únicamente necesaria para la programación de eventos de esos componentes integrados en la interfaz gráfica.

Para la integración del modelo de segmentación entrenado a la interfaz gráfica desarrollada, se ha hecho uso de Python. Cabe resaltar el uso de la librería *OpenCV*, esta biblioteca de código abierto ha tenido como objetivo encargarse de ciertas tareas relacionadas con el procesamiento de las imágenes de microplásticos. Esta biblioteca también ha servido para realizar un aumento en la muestra de imágenes del conjunto de datos a partir de imágenes

Figura 3.9: Logo Pyqt

ya existentes, junto con otra biblioteca llamada *Albumentations*. Ambos logo se apreacian respectivamente en las Figuras 3.10 y 3.11.

Figura 3.10: Logo OpenCV

Figura 3.11: Logo Albumentations

Finalmente, como software encargado de la gestión y control de versiones de todo el proceso de desarrollo junto con el almacenamiento del mismo en un repositorio privado, es GitHub. El logo del mismo se aprecia en la Figura 3.12.

Figura 3.12: Logo GitHub

3.2.2. Fase de creación y organización del conjunto de datos

Para la fase de *creación y organización del conjunto de datos*, se pretende ir preparando un primer conjunto de imágenes con la estructura correcta, para la realización de un primer entrenamiento a futuro que nos permitirá ir obteniendo mayor manejo y conocimiento de cara a diferentes entrenamientos finales con un conjunto de muestras mayor para obtener la mayor eficiencia y precisión.

Primeramente, contamos con un conjunto de 87 imágenes pertenecientes a las categorías de microplásticos 'pellets', 'fragments' y 'lines': en la página 5 se puede apreciar la apariencia de alguna de estas imágenes (Figura 1.1). A partir de aquí, se procede a realizar un proceso de etiquetado de imágenes haciendo uso de *Labelme* [15] para cada una de las muestras, consiguiendo aislar la silueta y el aspecto de los microplásticos capturados. Para los microplásticos de tipo fragment y lines se procede a utilizar polígonos convencionales. Para los pellets, que mayoritariamente presentan una silueta circular perfecta, se podría hacer uso de polígonos circulares que permitirían una mayor eficiencia y precisión durante este proceso. Pero más adelante, apreciamos que ese polígono circular produce una única entrada con su respectiva ubicación en el eje vertical y eje horizontal, cuya localización coincide con el centro del círculo. Mientras que nosotros necesitamos que se generen un conjunto de puntos cuyo conjunto conformen la silueta del microplástico, por esta razón, se requiere de hacer uso de los polígonos convencionales. En la Figura 3.13, se aprecia el etiquetado para caso de microplástico.

Tras la realización del etiquetado de las imágenes, se genera un JSON asociado a cada muestra etiquetada. Estos ficheros en formato JSON almacenan: la ubicación de cada uno de los polígonos que delimitan el contorno de los microplásticos, la etiqueta o sobrenombre con la que se guarda la delimitación de las muestras, el tipo de polígonos usado e información asociada a la imagen original tales como altura y ancho y ruta relativa de la imagen. Un ejemplo del contenido parcial de un JSON se puede ver en la Figura 3.14.

La herramienta YOLOv5 requiere de un fichero de texto asociado a cada imagen con su correspondiente formato. Nosotros al tener un fichero en formato JSON asociado a cada imagen generado tras el proceso de etiquetado, necesitaríamos migrar esos ficheros JSON a ficheros de texto atendiendo al formato requerido por YOLO5v5 para su correcta lectura.

El formato que se requiere es el siguiente: el fichero de texto debe tener el mismo nombre que la imagen original, el número de líneas del archivo corresponderá con el número de objetos capturados dentro de la imagen, el primer número corresponde al número de la clase, a partir de este número continuarían un conjunto de valores que pertenecerían a las coordenadas relativas X e Y correspondiente a cada uno de los polígonos que conforman el contorno del objeto. Cabe destacar que los valores X corresponderían con los valores impares y los valores Y con los valores pares a partir del valor que identifica el número de la clase. Estos valores decimales X e Y que fluctúan entre 0 y 1, resultan de dividir sus respectivos valores originales: X entre el ancho e Y entre el alto respectivamente de las imágenes. Asimismo, en nuestro caso particular en la que contamos con 3 tipos diferentes de micropláticos, los números de clases tendrán la siguiente correspondencia: 0: Fragments, 1: Lines y 2: Pellets. En la Figura 3.15, se presenta un ejemplo gráfico parcial de la estructura mencionada dentro de un fichero de texto perteneciente a un pellet:

Para la realización de esta migración de ficheros JSON a txt, hemos realizado un script que automatizará este proceso:

Algoritmo 3.1: Método para migrar ficheros JSON a ficheros txt

```
1 import os
2 import json
3
```

```
def json_to_txt_segmentation():
    path_to_json = 'C:/Users/raulv/MicroplasticosTFG/json'
    path_to_labels = 'C:/Users/raulv//MicroplasticosTFG/dataset/labels'

    for file in os.listdir(path_to_json):
        json_path = os.path.join(path_to_json, file)
        with open(json_path, "r") as json_file:
            json_data = json.load(json_file)
        base_name = os.path.splitext(file)[0]
        txt_path = os.path.join(path_to_labels, base_name + ".txt")

        with open(txt_path, "w") as txt_file:
            shapes = json_data['shapes']
            for i, shape in enumerate(shapes):
                element = shape['label']
                if element == 'Fragment':
                    number_class = '0'
                elif element == 'Line':
                    number_class = '1'
                elif element == 'Pellet':
                    number_class = '2'
                else:
                    number_class = '-1'

                if i > 0:
                    txt_file.write('\n')
                txt_file.write(number_class)
                points = shape['points']

                for j, point in enumerate(points):
                    point_x = point[0]
                    point_y = point[1]
                    width = json_data['imageWidth']
                    height = json_data['imageHeight']
                    x = str(point_x / width)
                    y = str(point_y / height)
                    content = " " + x + " " + y
                    txt_file.write(content)
            json_file.close()
            txt_file.close()

json_to_txt_segmentation()
```

Como última tarea de esta fase, se requiere de la correcta estructura y organización de directorios y ficheros. Esta fase es de vital importancia para la futura identificación por parte de la herramienta de los diferentes elementos necesarios para su correcta segmentación. Concretamente, en la Figura 3.16, se muestra la estructura a seguir:

En cuanto a la división del conjunto de datos, existen muchas opciones a utilizar que podrían dar buenos resultados. Sin una evidencia clara de que exista una metodología de división de conjuntos de datos a seguir a priori para obtener el mejor resultado posible, se requiere de un análisis previo en base a las necesidades particulares de cada situación y probar entre las diferentes opciones disponibles [16] . Por ello, se ha optado como primera opción a una división 70/30, donde el 70 % de las imágenes irán contenidas en el directorio de entrenamiento y el 30 % restante irán destinadas al directorio de validación.

3.2.3. Fase de entrenamiento de la red neuronal

Seguidamente, pasamos a la fase de *entrenamiento de red neuronal*. Teniendo el primer conjunto de imágenes ya estructurado, haremos uso de la herramienta *YOLOv5*, en el entorno de *Google Colab*, para el entrenamiento de la red neuronal con este set de datos. Es importante anotar que YOLOv5 nos da la opción de elegir entre 5 modelos preentrenados por ellos, de modo que el entrenamiento de nuestra red neuronal sea mucho más eficiente partiendo de un modelo preentrenado y no invertir tiempo adicional en la configuración de un modelo desde cero. Más específicamente, en la Tabla 3.1, se disponen de los diferentes modelos preentrenados ofrecidos para elegir.

Model	size (pixels)	mAP val 50-95	mAP val 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Tabla 3.1: Modelos preentrenados ofrecidos por YOLOv5

Nosotros, haremos uso del segundo modelo más pequeño y rápido: YOLOv5s. Debido a que no trataremos con conjuntos de datos grandes y detecciones complejas, con esta elección buscaremos el equilibrio entre la eficiencia y la precisión.

Antes que nada, realizamos una comprobación en Google Colab a partir de un código obtenido de un articulo público sobre YOLOv5 [14], que aplica máscaras binarias tras la lectura de un fichero de texto que contiene las coordenadas X e Y de los microplásticos etiquetados en una imagen. De este modo, comparamos entre el área resultante tras aplicar las máscaras y la imagen original, y así poder concluir si los ficheros de textos generados automáticamente por el script desarrollado, se han creado correctamente. El único inconveniente, es que solo se puede aplicar una máscara cada vez, debido a que detecta la primera línea del fichero de texto. Por lo que en cada ejecución, se irán eliminando las líneas que referencian a cada objeto a los que ya se les ha aplicado la máscara. A continuación, se muestra el resultado de ir ejecutando este código referente a un fichero de texto de un microplástico e ir realizando el proceso redactado anteriormente, comparándolo con su imagen original:

Tras apreciar que los ficheros de textos ligados a las imágenes originales están correctamente generados, procederemos a adentrarnos en destacar lo más significativo dentro de la primera iteración de entrenamiento del modelo.

Inicialmente, realizamos la clonación del repositorio de YOLOv5 en una hoja de Google Colab a través de Google Drive. Tras esto, instalamos todas las dependencias necesarias a través del fichero de texto ”requirements.txt” localizado en el mismo repositorio, este archivo contiene el conjunto de librerías y herramientas a instalar para el correcto uso de la herramienta.

Se debe crear un archivo de configuración denominado ’data.yaml’ ubicado al mismo nivel que los subdirectorios ’val’ y ’train’ y dentro del directorio ’images’ dentro del conjunto de datos, este fichero será el encargado de almacenar información necesaria para el entrenamiento. Todo ello, a través de la declaración de ciertos archivos asociados. A continuación se especifica el contenido para nuestras necesidades:

Algoritmo 3.2: Contenido del data.yaml

```
train: /content/microplasticosTFG-dataset/images/train
val: /content/microplasticosTFG-dataset/images/val
nc: 3
names: [ 'Fragment' , 'Line', 'Pellet' ]
```

- ✔ En el apartado ”train”, se debe indicar la ruta donde estarán las imágenes ubicadas en el directorio de entrenamiento.
- ✔ En el apartado ”val”, se debe indicar la ruta donde estarán las imágenes ubicadas en el directorio de validación
- ✔ En ”nc”, se especifica el número de clases que contendrá nuestro entrenamiento. En nuestro caso serán 3, que se corresponden con el número de tipos de microplásticos a detectar.
- ✔ En ”names”, se indica entre corchetes el nombre de cada una de las clases a detectar.

Tras esto, empezamos a preparar el setup mediante importaciones las diferentes librerías a usar para realizar el entrenamiento:

Algoritmo 3.3: Setup de librerías ejecutadas en Google Colab

```
%cd yolov5
import torch
from yolov5 import utils
import torch
import utils
from IPython import display
from IPython.display import clear_output
from pathlib import Path
import yaml
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import glob
```

```
13
14 %matplotlib inline
15 display = utils.notebook_init()
```

Finalmente, ejecutaremos el comando para el inicio del entrenamiento del modelo de segmentación, para ello haremos uso del fichero train.py suministrado por YOLOv5. Dentro de este comando, haremos uso de ciertos parámetros para facilitar ciertos procesos e intentar obtener mejores resultados:

- ✔ –weights: en este parámetro se especifica la ruta de los pesos generados por un modelo preentrenado por la herramienta o por un modelo customizado hecho desde 0.
- ✔ –img: aquí especificamos mediante un único parámetro el tamaño de las imágenes de entrada para la inferencia. Concretamente, se especifica el lado más largo. Resolviéndose de manera automática por la herramienta, el lado restante.
- ✔ –batch-size: este parámetro definido como tamaño del lote, indica el conjunto de muestras que son pasadas a la red por iteración.
- ✔ –epochs: este parámetro indica el número de veces que la red es entrenada con todas las muestras ofrecidas por el conjunto de datos.
- ✔ –project: este parámetro es usado para almacenar los resultados del entrenamiento en la ruta especificada.
- ✔ –name: aquí específicamos la ruta del archivo de configuración .yaml.

En nuestro caso particular, hemos optado con los siguientes hiperparámetros para la primera prueba de entrenamiento:

Algoritmo 3.4: Ejecución del comando para el entrenamiento del modelo

```
1 !python segment/train.py --data "/content/microplasticosTFG-dataset/images/data.
    yaml" --weights yolov5s-seg.pt --img 640 --batch-size 8 --epochs 300 --project
     "/content/gdrive/MyDrive/TFG" --name "300epochs-batch8-weights"
```

Tras la finalización de esta ejecución, obtendríamos diferentes archivos que contienen parámetros relativos a los resultados de este entrenamiento. Más adelante, se pretenderá dar un mayor énfasis a los diferentes entrenamientos realizados y a analizar en profundidad cada uno de los resultados obtenidos.

3.2.4. Creación y organización del conjunto de datos final

Tras realizar una prueba del flujo de trabajo de la herramienta, se pretende volver hacia la fase de *creación y organización del conjunto de datos* para la creación de un conjunto de imágenes con suficientes muestras para futuros entrenamientos del modelo de una manera más exhaustiva. De esta manera, haremos uso de la técnica de *image augmentation* la cual permite generar imágenes nuevas a partir de imágenes existentes y ha demostrado ser un enfoque que mejora la robustez de los modelos de aprendizaje profundo, además a través

del uso de la librería Albumentations, se dan posibilidades novedosas y confiables para la generación de nuevas muestras a través de diferentes enfoques [1]. Concretamente, haré uso de cambios en contrastes y brillos, y de volteos horizontales y verticales.

A continuación se aprecia un ejemplo gráfico en la que se contribuye a elaborar un conjunto de datos que pasa de 87 imágenes a 524 muestras:

Este paso ha sido realizado gracias al siguiente script:

Algoritmo 3.5: Método para realizar el image augmentation

```
import os
import cv2
import albumentations as A

def data_augmentation():
    directory = "C:/Users/raulv/Desktop/prueba"
    transform = A.Compose([
        A.RandomBrightnessContrast(p=1),
    ])
    for filename in os.listdir(directory):
        if filename.endswith(".JPG"):
            name, ext = os.path.splitext(filename)
            img = cv2.imread(os.path.join(directory, filename))

            img_flipped_vertical = cv2.flip(img, 0)
            img_flipped_horizontal = cv2.flip(img, 1)

            cv2.imwrite(os.path.join(directory, name
            +"-ver-flip" + ext), img_flipped_vertical)
            cv2.imwrite(os.path.join(directory, name
            + "-hor-flip" + ext), img_flipped_horizontal)

            transformed = transform(image=img)
            img_bright_contrast = transformed["image"]

            cv2.imwrite(os.path.join(directory, name
            + "-bright-contrast" + ext), img_bright_contrast)

data_augmentation()
```

Tras la ejecución de este script y generar todas las imágenes adicionales, es necesario generar los ficheros de texto con el formato requerido por la herramienta para ubicar los diferentes polígonos que conformarían la silueta de los microplásticos capturados. Para ello, existirían dos posibilidades posibles: etiquetar una a una cada imagen nueva generada en Labelme y ejecutar el script existente que genera el fichero de texto a partir del fichero JSON resultante tras el proceso de etiquetado, lo cual resultaría en una inversión de tiempo para el proceso manual elevada. O, crear un nuevo script que a partir del fichero de texto ligado a la imagen original y las imágenes nuevas generadas, se puedan generar automáticamente nuevos fiche-

ros de texto asociados a esas imágenes generadas, lo cual resultaría en invertir tiempo en desarrollar el script y ahorrar tiempo en la automatización de este proceso tras su ejecución.

Se optó por la segunda opción, donde se generan los ficheros de textos de las nuevas imágenes generadas a partir del fichero de texto original, teniendo en cuenta que las imágenes a las que no se le han aplicado ningún volteo: correspondiente a los cambios en brillo y contraste, coincidirían con los ficheros de texto originales. El problema vendría con los imágenes que se les ha aplicado un volteo vertical u horizontal, la solución sería hacer uso del círculo trigonométrico o círculo unitario, herramienta utilizada en conceptos de trigonometría. De este modo, teniendo en cuenta los diferentes valores de los senos y cosenos, podremos adaptar las coordenadas a los cambios en los volteos a través de la resta. En la Figura 3.20, se aprecia de manera gráfica el enfoque que se le quiere dar.

Tras la figura que explica el comportamiento a abordar por el script, se adjunta el código que realiza el proceso automatizado:

Algoritmo 3.6: Método para generar ficheros asociados al image augmentation

```
import os
import shutil

def txt_generator():
    path_to_images = "C:/Users/raulv/Desktop/prueba/images"
    path_to_txts = "C:/Users/raulv/Desktop/prueba/txts"

    imgFiles = os.listdir(path_to_images)

    for img in imgFiles:
        if img.endswith("-ver-flip.JPG") or img.endswith("-ver-flip-bright-
    contrast.JPG"):
            title = img.replace(".JPG", ".txt")
            txts_path = os.path.join(path_to_txts, title)

            originalTxtTitle = title.replace("-ver-flip", "")
            originalTxt = os.path.join(path_to_txts, originalTxtTitle)
            with open(originalTxt, "r") as f:
                lines = f.readlines()
                result = ""
                for line in lines:
                    content = line.split()
                    index = 0
                    for i in content:
                        if index % 2 == 0 and "." in i:
                            result += (str(float(1) - float(i))) + " "
                            index += 1

                        elif index % 2 == 1 and "." in i:
                            result += (str(i)) + " "
                            index += 1
```

```
31
32                      else:
33                          result += (str(i)) + " "
34                          index += 1
35
36                  result += "\n"
37              with open(txts_path, "w") as f:
38                  f.write(result)
39
40      elif img.endswith("-hor-flip.JPG") or img.endswith("-hor-flip-bright-
    contrast.JPG"):
41          title = img.replace(".JPG", ".txt")
42          txts_path = os.path.join(path_to_txts, title)
43
44          if img.endswith("-hor-flip.JPG"):
45              originalTxtTitle = title.replace("-hor-flip", "")
46
47          if img.endswith("-hor-flip-bright-contrast.JPG"):
48              originalTxtTitle = title.replace("-hor-flip-bright-contrast", "")
49
50          originalTxt = os.path.join(path_to_txts, originalTxtTitle)
51          with open(originalTxt, "r") as f:
52              lines = f.readlines()
53              result = ""
54              for line in lines:
55                  content = line.split()
56                  index = 0
57                  for i in content:
58                      if index % 2 == 0 and "." in i:
59                          result += (str(i)) + " "
60                          index += 1
61
62                      elif index % 2 == 1 and "." in i:
63                          result += (str(float(1) - float(i))) + " "
64                          index += 1
65
66                      else:
67                          result += (str(i)) + " "
68                          index += 1
69
70                  result += "\n"
71          with open(txts_path, "w") as f:
72              f.write(result)
73
74      elif img.endswith("-bright-contrast.JPG"):
75          title = img.replace(".JPG", ".txt")
76          txts_path = os.path.join(path_to_txts, title)
77          src = txts_path.replace("-bright-contrast","")
```

```
78              shutil.copy(src, txts_path)
79
80  txt_generator()
```

A continuación, con la creación de este conjunto de datos más completo se realizan varios entrenamientos en los que se varía la división de las imágenes en los directorios de entrenamiento y validación y se prueban con diferentes cambios en hiperparámetros.

3.2.5. Diseño de la interfaz gráfica e integración de la red neuronal

Tras la obtención de una combinación que aporte resultados favorables, pasamos a la fase del *diseño de la interfaz gráfica e integración de la red neuronal*.

En esta fase se pretende crear una interfaz intuitiva y sencilla para la futura integración del comportamiento obtenido tras el entrenamiento del modelo, o en otras palabras la *inferencia*.

El término de inferencia en el ámbito de la inteligencia artificial, se denomina a la habilidad obtenida por parte de la red neuronal tras realizar el proceso de entrenamiento, de modo que pueda aplicar y poner en práctica lo aprendido para categorizar y clasificar los problemas a afrontar en cuestión. En nuestro caso, tras el proceso de entrenamiento, permitirá clasificar y segmentar los diferentes microplásticos identificados en las diferentes imágenes a tratar.

Para realizar la integración de la inferencia a la interfaz gráfica, nos hemos basado en un diseño por menús donde se distribuirán los diferentes procesos a tratar. El flujo de procesos que se pretende seguir en esta primer versión será el siguiente:

1. Un menú 'File' - 'Open image' que permita la apertura de imágenes, a partir de una interfaz navegable que permite elegir la imagen a abrir dentro del almacenamiento del dispositivo de escritorio que se ejecute.
2. Tras la apertura de la imagen, se permite la posibilidad realizar la detección de los microplásticos dentro de la imagen en una opción que se puede encontrar en un menú 'Process' - 'Microplastics'
3. Tras la correcta detección, la imagen previamente abierta se verá modificada resaltando los microplásticos detectados a través de cuadros delimitadores.
4. Tras esto, se podría proceder a guardar la imagen con los resultados obtenidos a través de otra interfaz navegable, en la que se puede elegir la ubicación de destino de la imagen del dispositivo en el que se ejecuta el procedimiento. Esta acción se ubica en 'File' - 'Save results'.
5. Como opciones adicionales, se habilita una acción de 'Exit' para cerrar la interfaz gráfica y otra acción 'About the app', que permite conocer información relativa al funcionamiento de la aplicación.

Tras crear la interfaz gráfica mediante la disposición de diferentes componentes ofrecidos en la GUI que nos brinda la librería pyqt, se genera un archivo .ui con toda la estructura. A partir

de aquí, se procedería a realizar la integración del modelo que mejor resultados ha obtenido en la interfaz gráfica, procediendo a programar los diferentes eventos que permitirían aportar el dinamismo a los elementos estáticos establecidos. Para crear la conexión entre componentes-acciones nos basamos en Python, donde a partir de varias definiciones se van desarrollando los diferentes procesos a abordar.

Concretamente, para el proceso de identificación de microplásticos, se aplica un comportamiento similar al que usa YOLOv5 de manera predeterminada, de modo que la forma de identificar cada objeto detectado se basa en una caja delimitadora que los rodea.

Para la elaboración de estas delimitaciones, se procede a obtener de las predicciones del modelo las coordenadas que comprendían los límites de cada objeto identificado. Y tras la obtención de estas coordenadas, se consiguen dibujar los rectángulos alrededor de los microplásticos identificados a través del uso de la librería OpenCV, que se comportarían como cajas delimitadoras. Tras procesar la imagen de esta manera para las muestras identificadas, se acopla esa imagen en un componente compatible con la interfaz gráfica para ser mostrado en la aplicación de escritorio. En la figura 3.21, se aprecian los resultados obtenidos, tras realizar el proceso de identificación en esta versión de aplicación ejecutada para diferentes muestras.

3.2.6. Fase de evaluación y mejora

Tras observar los resultados obtenidos en la 3.21, apreciamos a primera vista que a pesar de que el proceso de detección cumple con la necesidad que deseábamos suplir, gracias al correcto comportamiento del modelo entrenado. Se aprecian inconvenientes visuales relacionados con: cambios en el formato de color en comparación entre la imagen original y la imagen tras el proceso de detección realizado y su consiguiente guardado, y otro problema relacionado con la relación de aspecto, donde se aprecia un cambio claro en las resoluciones de las imágenes original y tras su detección.

Además, existen varios casos de muestras, especialmente en Lines, que contienen varios microplásticos capturados de manera cercana, en el que las detecciones derivan en solapamientos de estas cajas delimitadoras, pudiendo dificultar la lectura de los resultados por parte los usuarios finales. En las viñetas de la Figura 3.22, se aprecian algunos casos de lo descrito.

Otro inconveniente que afecta de manera mayoritaria a los microplásticos tipo Line, corresponde a otro detalle visual en la delimitación del área de los objetos identificados, ya que la caja delimitadora captura mucho ruido adicional irrelevante correspondiente con el fondo de la imagen, por tener que adaptarse a la morfología alargada de esta categoría. Esto podría resultar confunso y aportar incertidumbre en escenarios donde el fondo no fuese totalmente liso, y pudieran existir otros elementos independientes que aparezcan dentro del area delimitada y que no queramos identificar. A continuación, se adjuntan algunos casos de lo expresado previamente:

Como último detalle a mejorar de cara a la versión final, sustituiremos el uso de las barras de desplazamientos aplicadas en la anterior interfaz gráfica, y la cambiaremos por la posibilidad

de redimensionar y ajustar el tamaño de la imagen mostrado en la pantalla principal de la aplicación de escritorio a través de cualquiera de las esquinas de la misma.

De cara al desarrollo de la versión final, se tiene como objetivo conseguir un incremento a partir de la anterior iteración pretendiendo: conseguir aplicar una máscara binaria a los objetos identificados, de cara a resaltar no solo cada microplástico a través de cuadros delimitadores, sino *el área de cada microplástico identificado*, la adición de una etiqueta que categorice cada tipo de microplástico detectado al que se corresponde, junto con el grado de confianza para cada caso, representado como un valor entre 0 y 1.

Con este nuevo enfoque, se podrá identificar de una manera más fácil y precisa los diferentes tipos de microplásticos ajustándose a su morfología singular gracias a la máscara aplicada. Además, categorizando en forma de texto el tipo de la clase de microplástico que ha sido identificado, y aportando una diferenciación adicional, a través del uso de diferentes colores para la segmentación de los mismos. Asimismo, se pretenderán resolver los últimos detalles a corregir descritos previamente.

Para la elaboración de este nueva versión mejorada, realizaremos una modificación del código en la que sustituiremos la manera de representar los rectángulos delimitadores de los objetos identificados, por el mínimo código viable extraido de lo ya interiormente desarrollado por YOLOv5, adaptándose a nuestra situación particular. Para ello, se ha requerido de invertir tiempo de investigación entre los diferentes ficheros que componen la estructura de archivos de la herramienta YOLOv5, donde se estima un número total de 6.000 líneas de código aproximadamente.

Concretamente, se identifica que dentro del directorio ”segment” es el espacio donde se ubica el código bruto en cuestión. Específicamente, el fichero en cuestión se denomina ”predict.py”.

El código correspondiente al fichero predict.py permite realizar inferencias de segmentación utilizando el modelo YOLOv5. Mediante la carga del modelo, permite el procesamiento de diferentes fuentes a través de su selección, como imágenes, vídeos o muestras extraídas a través de webcams, y generar predicciones de segmentación a partir de las mismas. Asimismo, permite la muestra de los resultados visualmente, el guardado de las imágenes o videos, y también existe la posibilidad de guardar las predicciones en archivos de texto. El código también brinda opciones customizadas para adaptarse a diferentes escenarios, como seleccionar el tamaño de imagen, ajustar parámetros de confianzas, entre otras...

Por lo tanto, el *modus operandi* para pasar de un código extenso y específico ya desarrollado y obtener el mínimo código aplicable a nuestra situación particular, se basa en la realización de varias trazas minuciosas a través de depuraciones del fichero en cuestión.

Tras varias iteraciones, el código funcional obtenido para nuestra situación incorporado en el método identifyMicroplastics(), es el siguiente:

Algoritmo 3.7: Versión final método para segmentación de microplásticos

```
1    def identifyMicroplastics(self):
2        device = select_device("cpu")
3        model = DetectMultiBackend("best.pt", device=device, dnn=False, data="data
    .yaml", fp16=False)
```

```
        names = model.names
        dt = (Profile(), Profile(), Profile())

        image = cv2.imread(self.fileName)

        with dt[0]:
            im = torch.from_numpy(image).to(model.device)
            im = im.half() if model.fp16 else im.float()
            im /= 255
            if len(im.shape) == 3:
                im = im[None]
                im = np.transpose(im, (0, 3, 1, 2))

        with dt[1]:
            pred, proto = model(im, augment=False, visualize=False)[:2]

        with dt[2]:
            pred = non_max_suppression(pred, conf_thres=0.5, iou_thres=0.5,
    max_det=1000, nm=32)

        for i, det in enumerate(pred):
            annotator = Annotator(image, line_width=3, example=str(names))
            if len(det):
                masks = process_mask(proto[i], det[:, 6:], det[:, :4], im.shape
    [2:], upsample=True)
                det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], image.shape).
    round()

                annotator.masks(
                    masks,
                    colors=[colors(x, True) for x in det[:, 5]],
                    im_gpu=torch.as_tensor(image, dtype=torch.float16).to(device).
    permute(2, 0, 1).flip(
                        0).contiguous() / 255)

                for j, (*xyxy, conf, cls) in enumerate(reversed(det[:, :6])):
                    c = int(cls)
                    label = f'{names[c]} {conf:.2f}'
                    annotator.box_label(xyxy, label, color=colors(c, True))

            image = annotator.result()

        height, width, channel = image.shape
        bytes_per_line = channel * width
        q_image = QtGui.QImage(image.data, width, height, bytes_per_line, QtGui.
    QImage.Format_BGR888)

        pixmap = QtGui.QPixmap.fromImage(q_image)
```

```
47
48          self.image = q_image
49          self.label.setPixmap(pixmap)
50          self.label.setScaledContents(True)
51          self.actionSave_results.setEnabled(True)
```

La ejecución de este método, genera los resultados gráficos dentro de la aplicación apreciados en las Figuras 3.24a, 3.24b y 3.24c.

Apreciamos cambios de tamaño notorios en las dimensiones de la pantalla a mostrar por la herramienta desarrollada, representándose el resultado de la manera que se requería, a través de la aplicación de máscaras mediante la segmentación a los elementos identificados. Todo ello mediante el uso de los métodos ya desarrollados por YOLOv5, en vez del uso de OpenCV.

Además, se aprecia como el formato de color no se desvía tras la ejecución de los diferentes procesos en la aplicación de escritorio. Esto lo hemos solucionado a través del cambio de formato de colores de RGB a BGR previo al muestreo en la interfaz gráfica. Este formato de color se mantiene igual tras ejecutar el proceso de guardado de las imágenes en el equipo.

Por último, también realizamos los ajustes en la interfaz gráfica concediendo a nivel de usuario la posibilidad de redimensionar y ajustar la pantalla principal mostrada por la aplicación, sustrayendo las anteriores barras de desplazamiento que no se llegaban a requerir. Debido a que la imagen se redimensionaba automáticamente al tamaño del componente que lo contenia tras su apertura. Junto a esto, arreglamos la relación de aspecto para el muestreo y guardado de las imágenes tras ser tratadas, manteniéndose del mismo tamaño que la imagen original.

Realizando pruebas de rendimiento del modelo de segmentación integrado en la interfaz gráfica, apreciamos de manera general muy buenos resultados con un grado de confianza en la mayoría de situaciones, mostradas en las figuras 3.24a, 3.24b, 3.24c. Pero es importante anotar ciertos casos en los que no se aprecian resultados esperados

En este estadio de desarrollo, surge la idea de acoplar una nueva y última funcionalidad. El desempeño de esta nueva función consistirá en la capacidad de exportar un fichero de texto, tras haber realizado el proceso de identificación de microplásticos en una imagen concreta, cuyo nombre del fichero será el mismo de la imagen original referida, y cuyo contenido generado por cada elemento identificado, estará estructurado en las siguientes categorías:

- ✔ 'Tipo de microplástico': discerniendo entre *pellet, fragment o line.*
- ✔ 'Coordenadas': compuesto por 4 valores, que corresponde con las coordenadas (x1,x2,y1,y2) que distinguirían los elementos identificados en un cuadro delimitador.
- ✔ 'Confianza': este valor representa el grado de confianza de la clasificación realizada
- ✔ 'Número de píxeles': número de píxeles activos para realizar el proceso de segmentación del área cubierta

El código referente a esta nueva funcionalidad es el siguiente:

Algoritmo 3.8: Método que permite la exportación de un fichero con contenido relativo al proceso de segmentación

```
    def exportResults(self):
        dir_name = QFileDialog.getExistingDirectory(self, "Select location for
    exporting file")

        file_name = os.path.basename(self.fileName)
        file_root, _ = os.path.splitext(file_name)
        new_file_name = file_root + ".txt"
        new_file_path = os.path.join(dir_name, new_file_name)

        with open(new_file_path, 'w') as file:
            file.write("Tipo de microplstico,Coordenadas,Confianza,Nmero de pxeles
    \n")

            for m in self.dataframe:
                tipo = m['tipo']
                coordenadas = ','.join([str(c) for c in m['coordenadas']])
                confianza = str(m['confianza'])
                pixeles = str(m['pixeles'])

                row = f"{tipo},{coordenadas},{confianza},{pixeles}\n"
                file.write(row)

        msg = QMessageBox()
        msg.setIcon(QMessageBox.Information)
        msg.setText("The file has been exported successfully.")
        msg.setWindowTitle("Export file")
        msg.setStandardButtons(QMessageBox.Ok)
        msg.exec_()
```

Este método sigue un patrón similar al método ya desarrollado llamado saveResults, que permitía el guardado de la imagen resultante brindando una interfaz navegable de cara a elegir el destino, tras generarse las anotaciones correspondientes después de ejecutar el proceso de identificación. En este caso, mantenemos el mismo procedimiento pero generando un fichero de texto.

Además, en el método de identifyMicroplastics(), fue necesario añadir unas instrucciones justo después de generar las anotaciones a integrar en la imagen mostrada en la interfaz de usuario, que permitirán preparar la lista que contendrá el contenido especificado previamente. Concretamente, se mantiene el resto del código implementado pero se añadirían las siguientes líneas en la ubicación a mostrar:

Algoritmo 3.9: Código adicional añadido en método identifyMicroplastics()

```
    def identifyMicroplastics(self):
        (...)
                for j, (*xyxy, conf, cls) in enumerate(reversed(det[:, :6])):
                    c = int(cls)
                    label = f'{names[c]} {conf:.2f}'
```

```
                    annotator.box_label(xyxy, label, color=colors(c, True))

                    xyxy = [coord.tolist() for coord in xyxy]
                    enabledPixels = np.where(masks[j] > 0)
                    pixelsPerMicroplastics = len(enabledPixels[0])

                    microplastic = {
                        "tipo": names[c],
                        "coordenadas": xyxy,
                        "confianza": "{:.2f}".format(conf),
                        "pixeles": pixelsPerMicroplastics
                    }
                    self.dataframe.append(microplastic)

            image = annotator.result()

        (...)
```

En la Figura 3.25, se aprecia gráficamente un ejemplo del contenido de un fichero de texto generado por la función desarrollada.

Con la implementación de esta última funcionalidad, la cual sigue un enfoque opuesto al del proceso de segmentación, permitirá la realización de un análisis de los datos obtenidos sin necesidad de tener que mirar la imagen original o la imagen tras aplicar el proceso de segmentación. Además, el formato del contenido del fichero basado en separaciones por comas, permitirá la versatilidad de abrir este fichero de texto en otros formatos como .csv. Lo que permitirá adaptarse a un mayor número de usuarios en este sentido.

3.3. Demostración funcionamiento aplicación

En este espacio se demuestra, a través de imágenes, la secuencia de pasos de cómo sería el flujo de funcionamiento completo del producto final desarrollado para un ejemplo en particular:

Para ello, primeramente comenzamos con la ejecución de la aplicación en Pycharm, que resulta en la pantalla principal mostrada en la Figura 3.26.

Tras esto, procederemos a realizar la apertura de la imagen a tratar, la secuencia se aprecia en las Figuras 3.27, 3.28, 3.29.

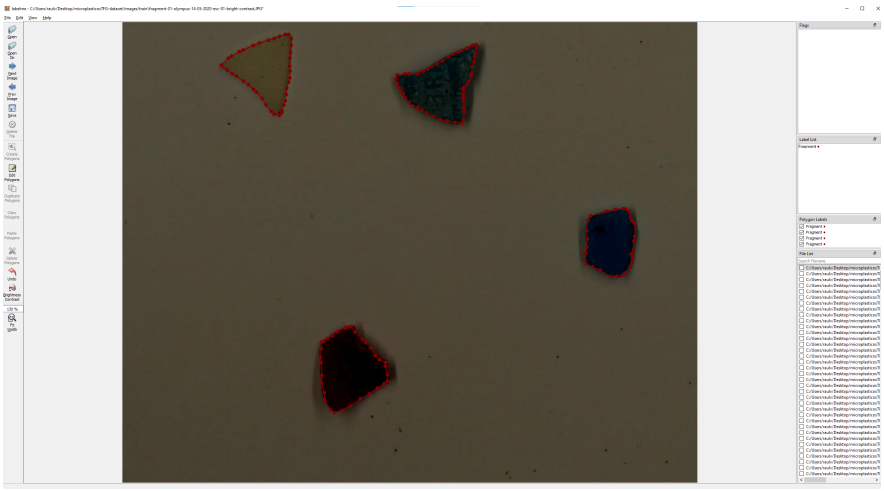
Tras localizar la imagen que se pretende a abrir y realizar la apertura de la misma, se procede a ejecutar la segmentación sobre la imagen, apreaciado en las Figuras 3.30, 3.31.

Tras el proceso de segmentación ejecutado en la imagen a tratar, se liberan las opciones de o guardar la imagen tratada en el equipo o de exportar un fichero de texto con información relativa a la identificación realizada. En las figuras 3.32, 3.33, 3.37 se muestra como sería el flujo para cada el proceso de guardado.

Para el proceso de exportación del fichero de texto, el proceso estaría documentado en las Figuras 3.35, 3.36, 3.37.

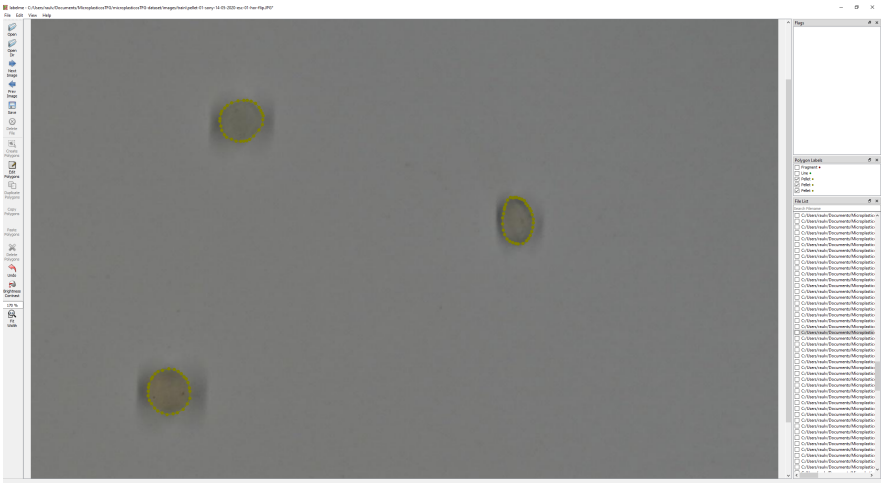
Las dos acciones previamente mencionadas, resultan en generar los siguientes archivos mostrados en las Figuras 3.38, 3.39.

Como funciones adicionales, se permite la manipulación de las dimensiones de la pantalla principal a partir de su manipulación con el ratón en las esquinas o límites inferiores y superiores de la pestaña. Incluido con una opción de 'Exit' para finalizar la ejecución de la aplicación y una opción de 'About the app', con información relativa de la aplicación, en la Figura 3.40 se aprecia el contenido de esta.



(a) Etiquetado de muestras fragments

(b) Etiquetado de muestras lines



(c) Etiquetado de muestras pellets

Figura 3.13: Etiquetado de los diferentes tipos de microplásticos

Figura 3.14: Parte del contenido de un texto JSON generado tras etiquetado de imagen

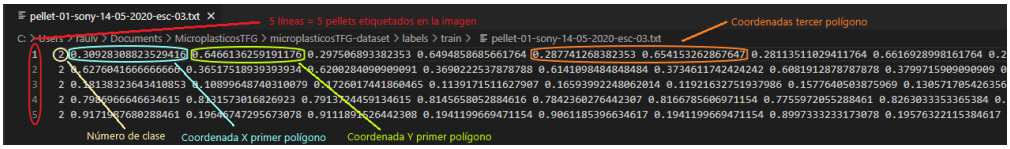


Figura 3.15: Estructura de fichero de texto de un pellet

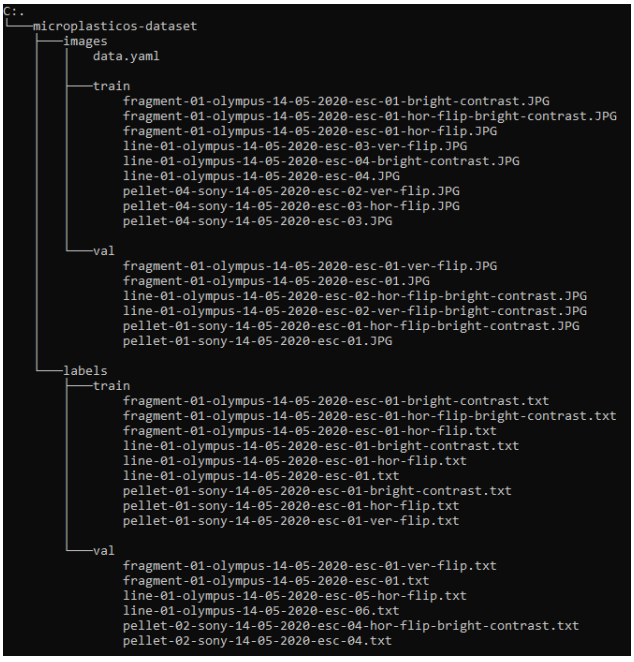


Figura 3.16: Estructura y organización del conjunto de datos por directorios

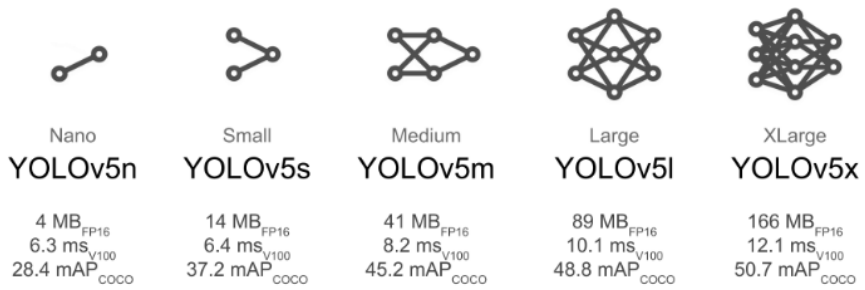


Figura 3.17: Modelos YOLOv5 extraídos de documentación oficial

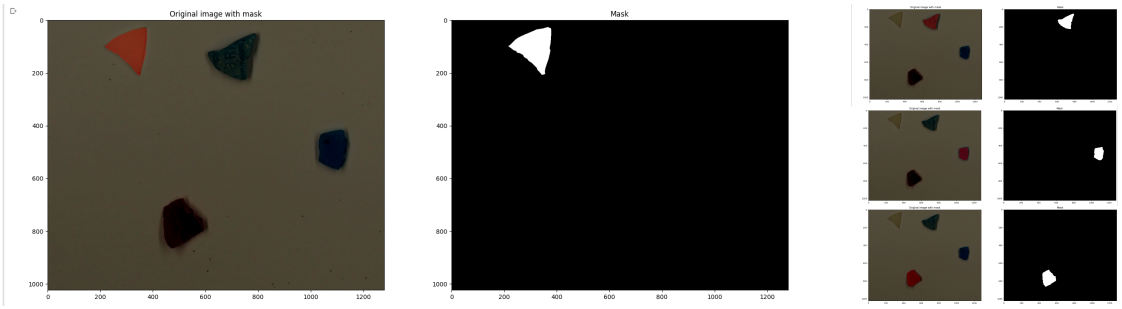


Figura 3.18: Máscaras aplicadas a un fichero de texto con elementos identificados

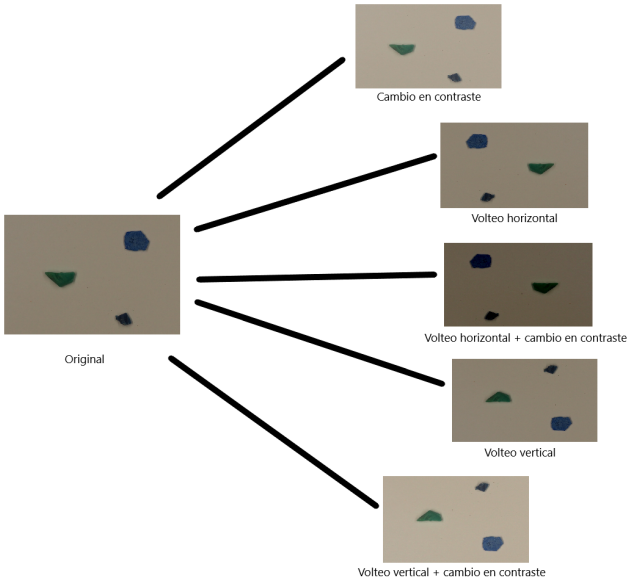
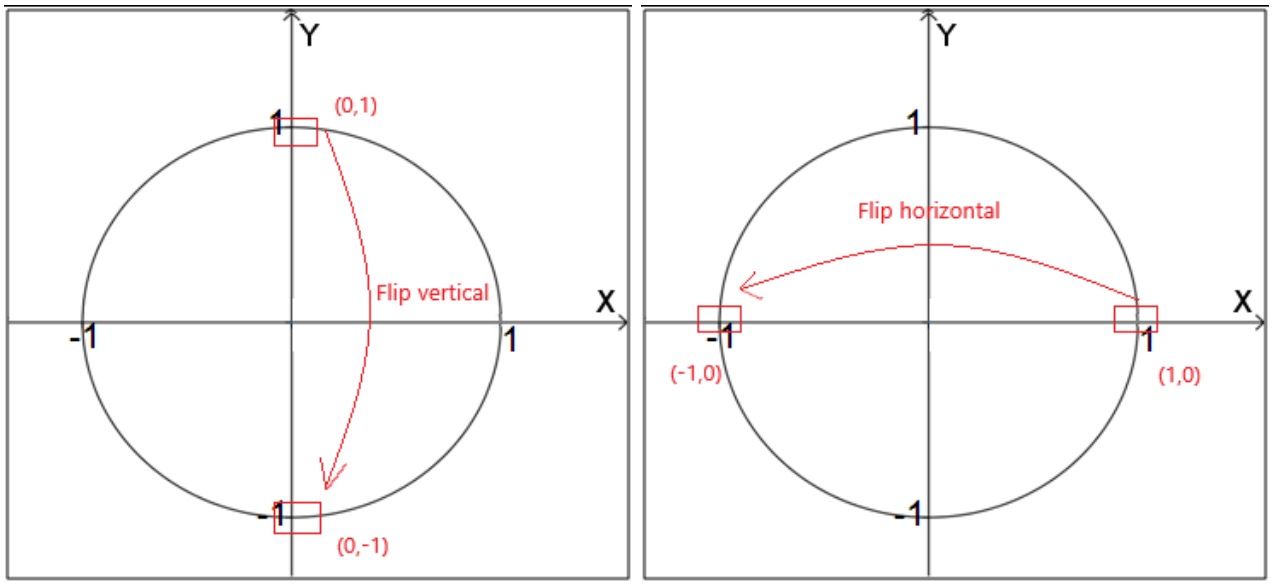


Figura 3.19: Técnica de *image augmentation*



(a) Explicación gráfica volteo vertical (b) Explicación gráfica volteo horizontal

Figura 3.20: Explicación gráfica volteos de imágenes con coordenadas

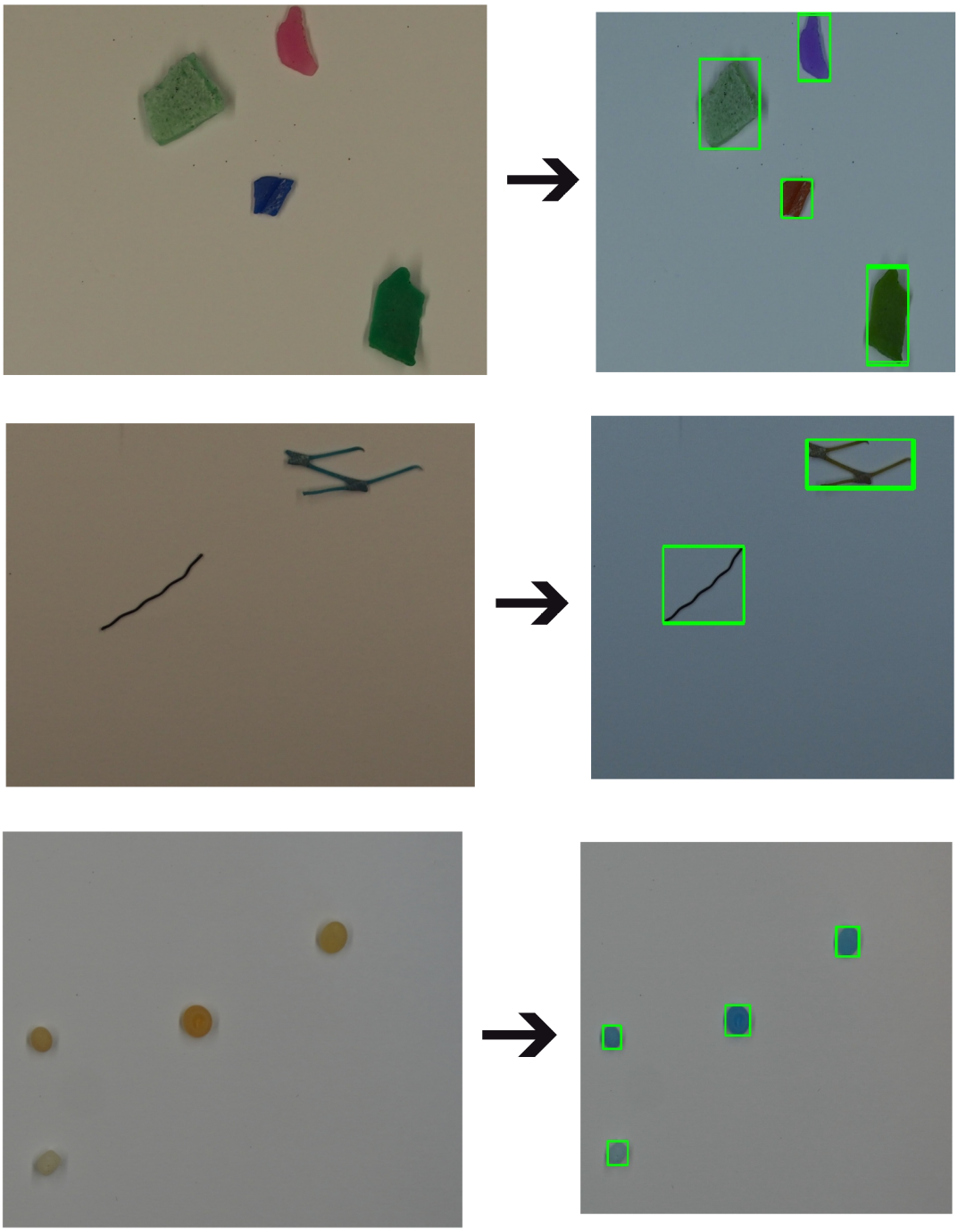
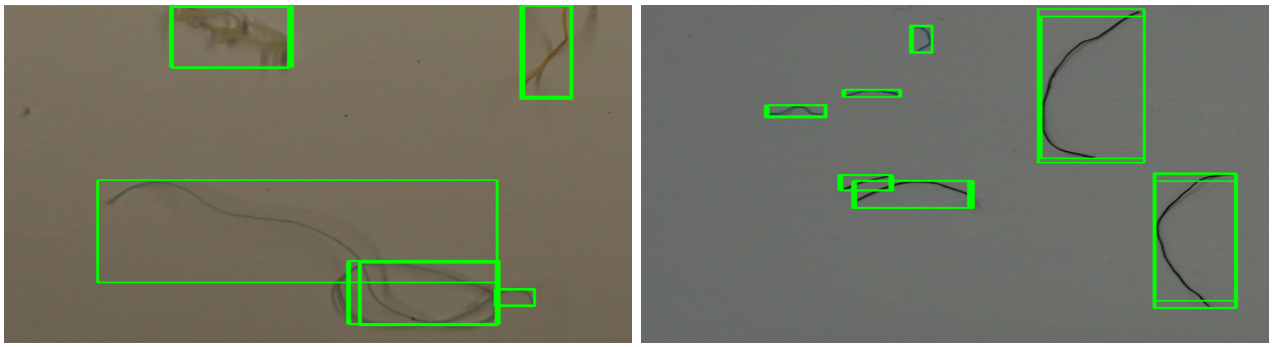
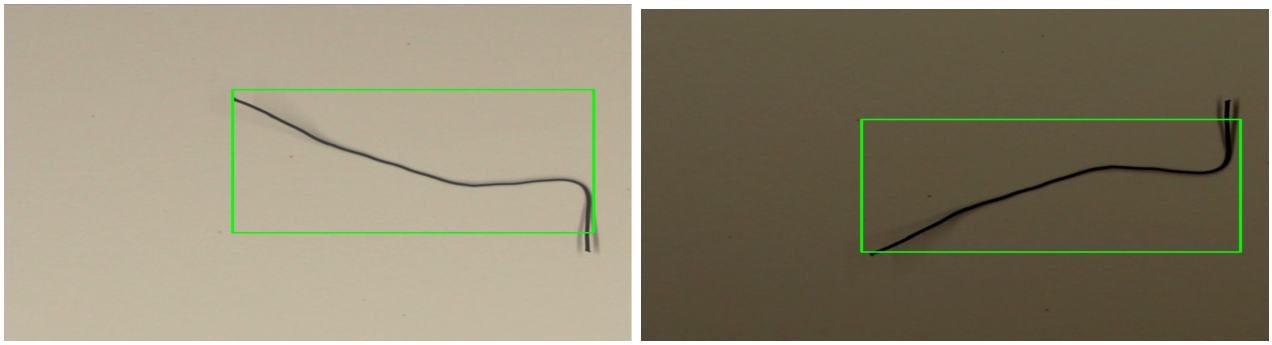


Figura 3.21: Ejemplos obtenidos tras la ejecución de la primera versión de la aplicación



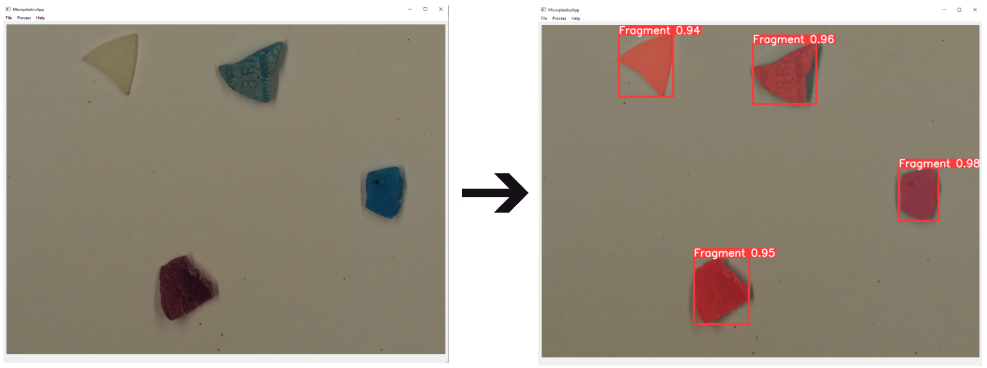
(a) Caso 1 (b) Caso 2

Figura 3.22: Solapamientos de cajas delimitadoras

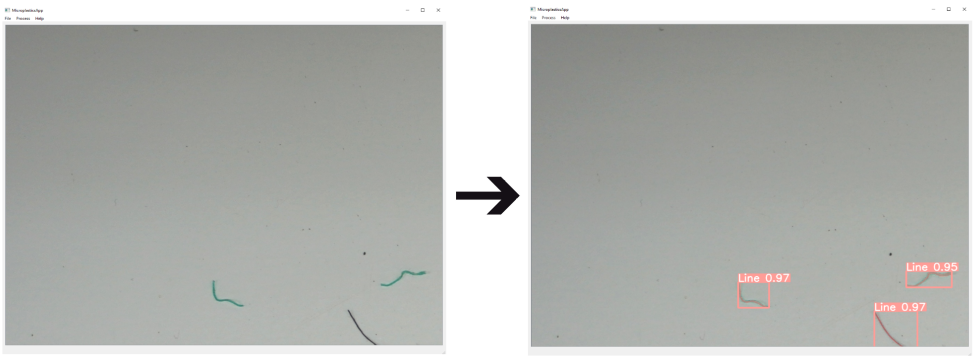


(a) Caso 1 (b) Caso 2

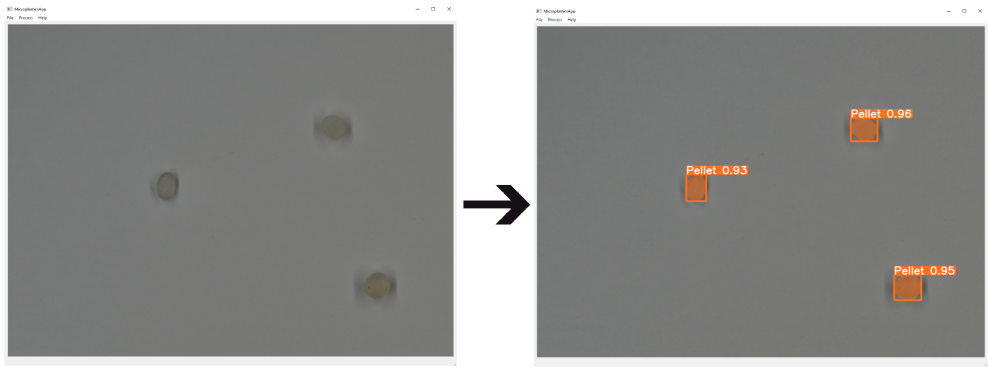
Figura 3.23: Ruidos visuales en el fondo de cajas delimitadoras



(a) Ejemplo con muestra de fragmentos

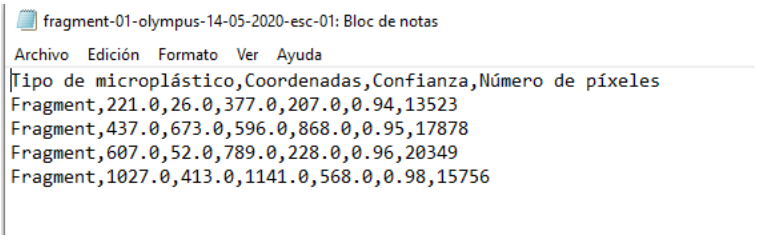


(b) Ejemplo con muestra de cebos de pesca



(c) Ejemplo con muestra de gránulos

Figura 3.24: Ejemplos obtenidos tras la ejecución de la versión final de la aplicación


fragment-01-olympus-14-05-2020-esc-01: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
Tipo de microplástico,Coordenadas,Confianza,Número de píxeles
Fragment,221.0,26.0,377.0,207.0,0.94,13523
Fragment,437.0,673.0,596.0,868.0,0.95,17878
Fragment,607.0,52.0,789.0,228.0,0.96,20349
Fragment,1027.0,413.0,1141.0,568.0,0.98,15756
```

Figura 3.25: Ejemplo de contenido exportado en fichero de texto

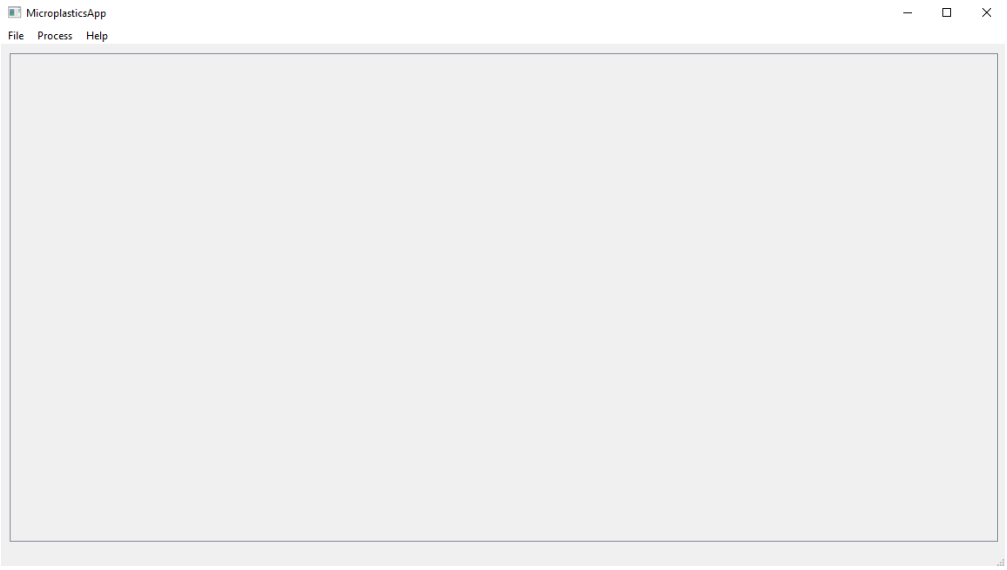


Figura 3.26: Pantalla principal de la aplicación

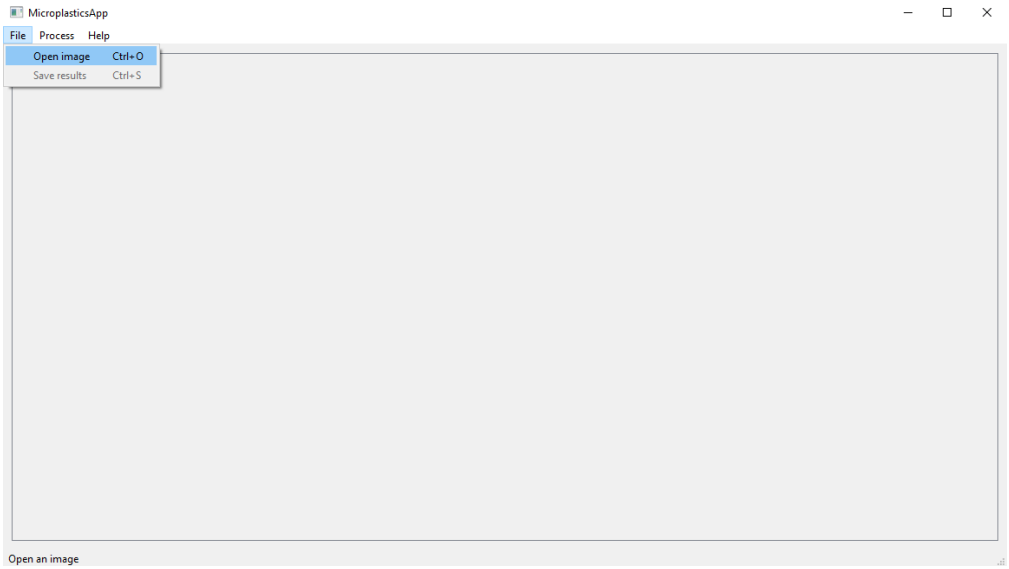


Figura 3.27: Apertura de la imagen

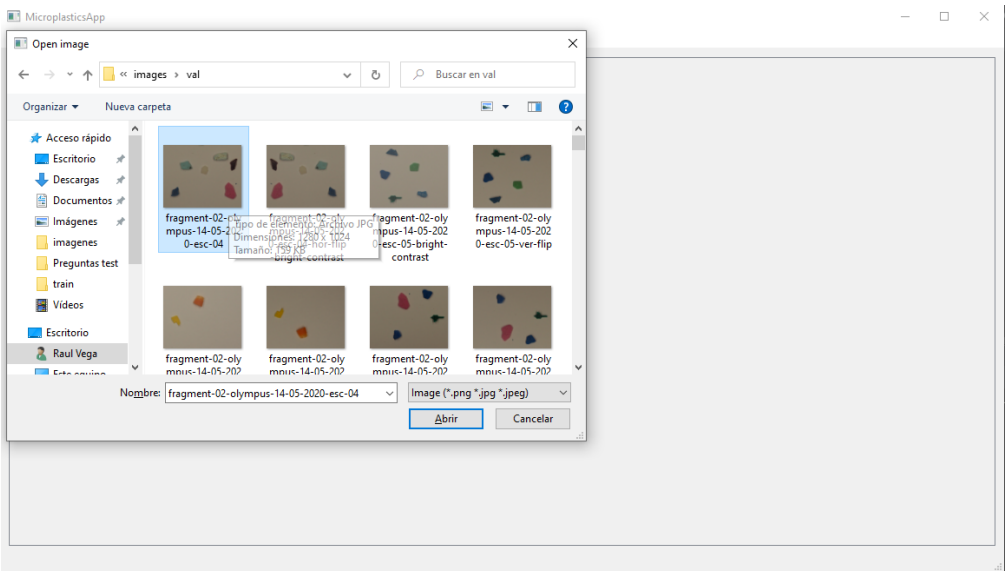


Figura 3.28: Selección de la imagen

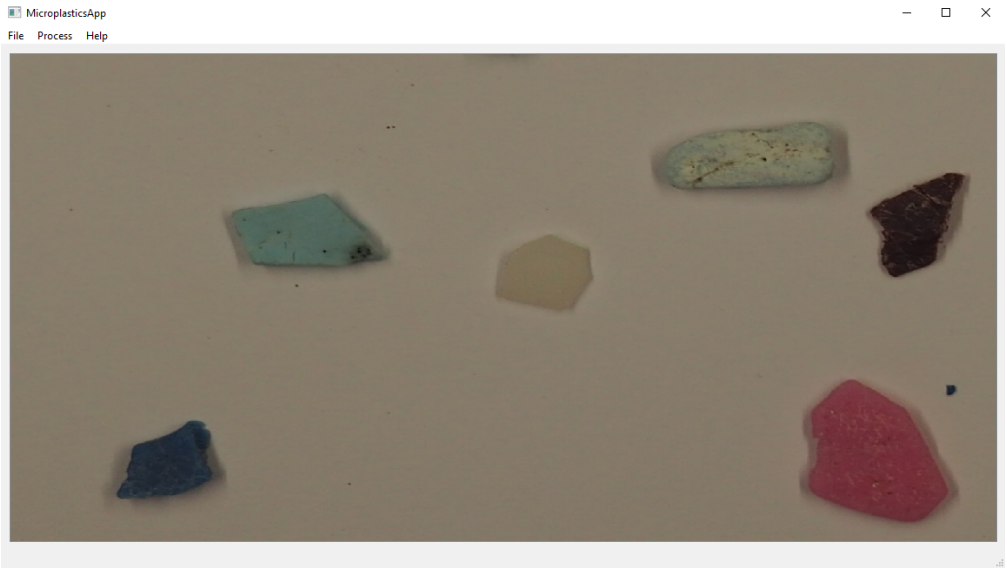


Figura 3.29: Pantalla principal con imagen abierta

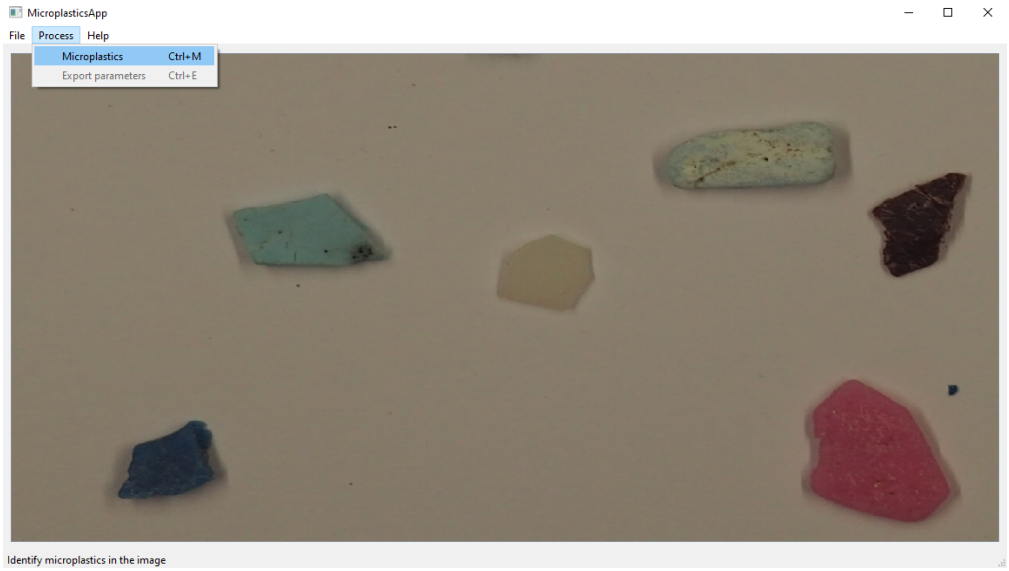


Figura 3.30: Selección de procesamiento de la imagen

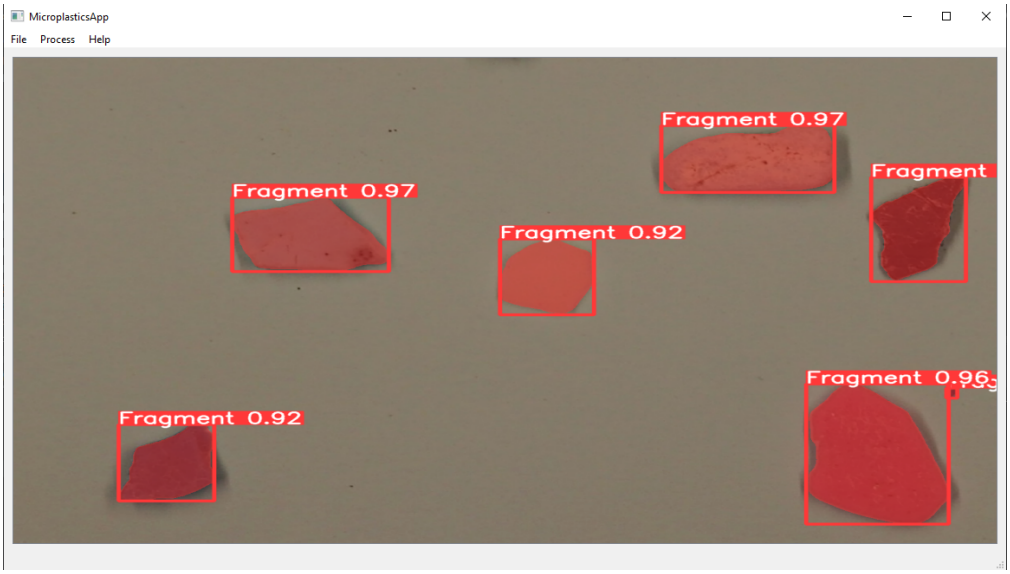


Figura 3.31: Segmentación realizada en la imagen a tratar

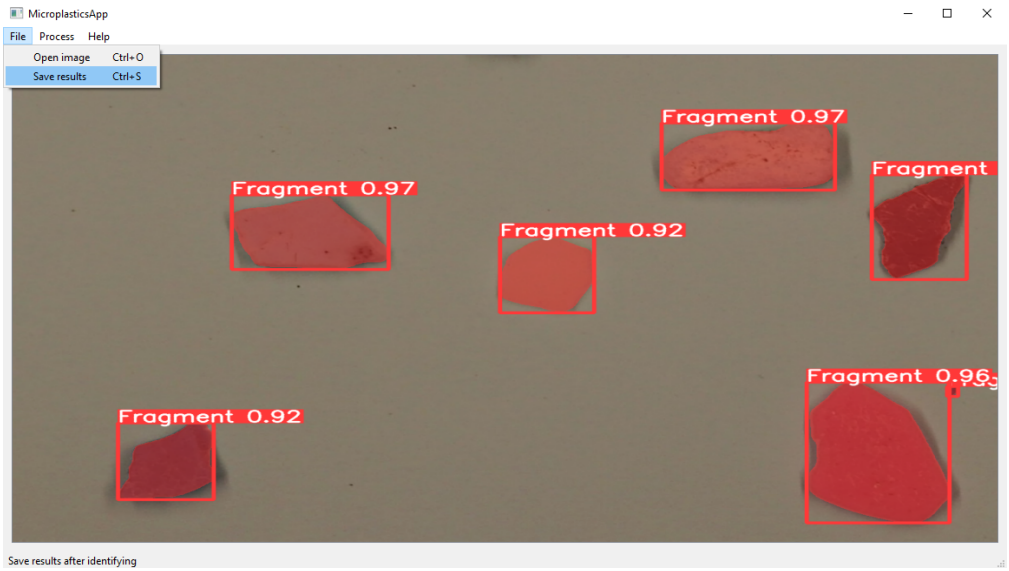


Figura 3.32: Proceso de guardar imagen tratada

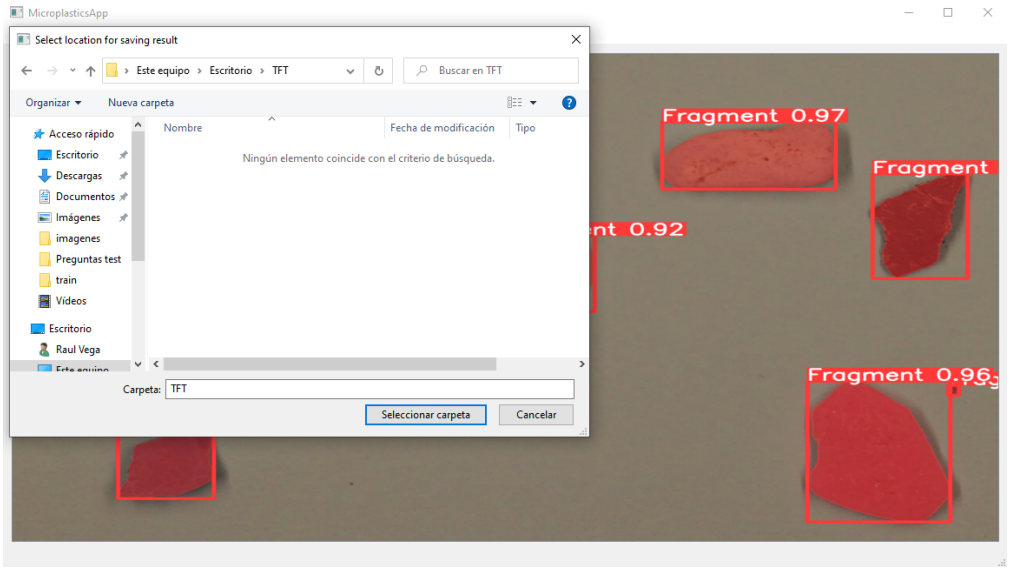


Figura 3.33: Selección del destino de la imagen a guardar

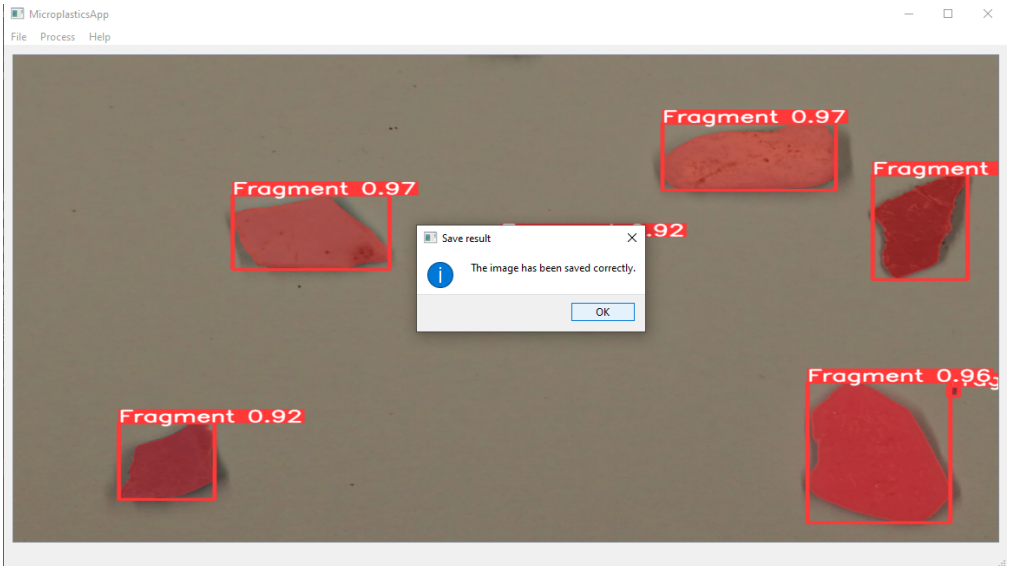


Figura 3.34: Aviso de guardado éxitoso

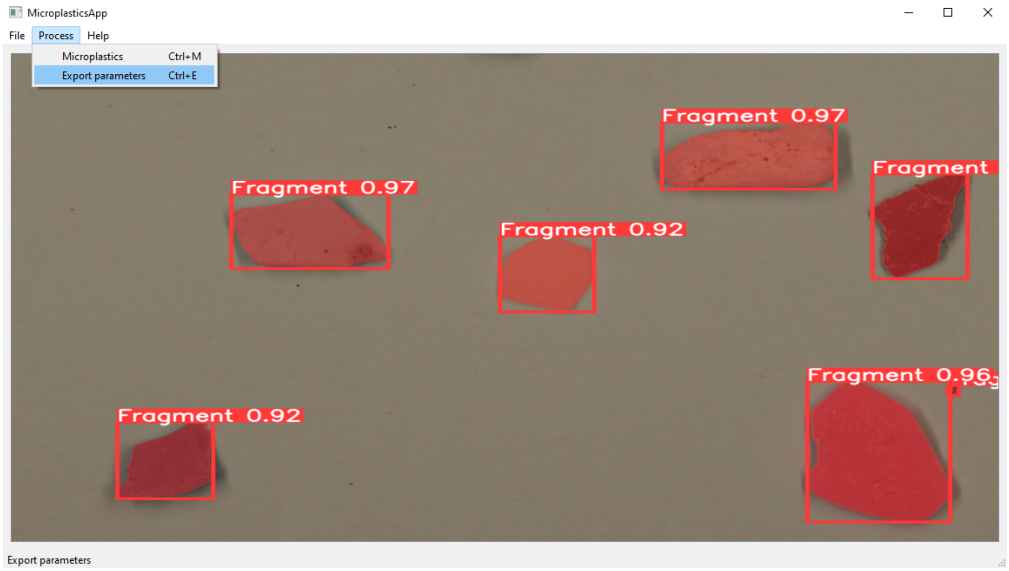


Figura 3.35: Proceso de exportación de parámetros en un fichero

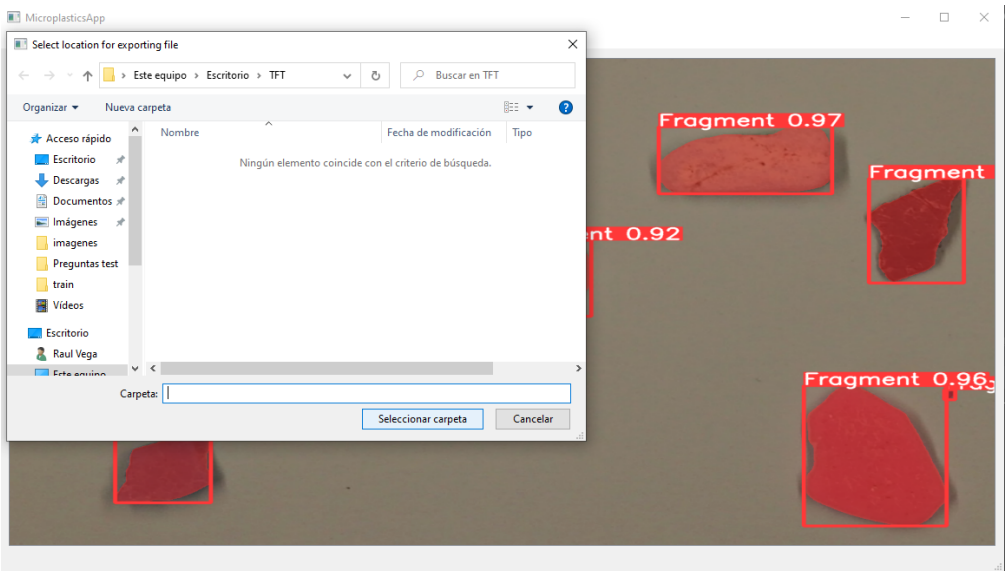


Figura 3.36: Selección del destino de la imagen a guardar

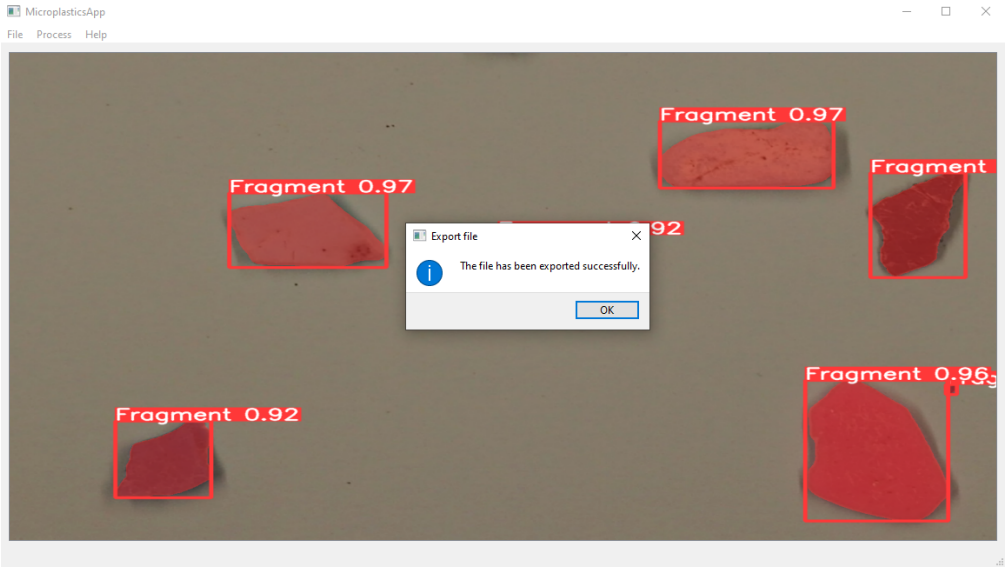


Figura 3.37: Aviso de guardado éxitoso

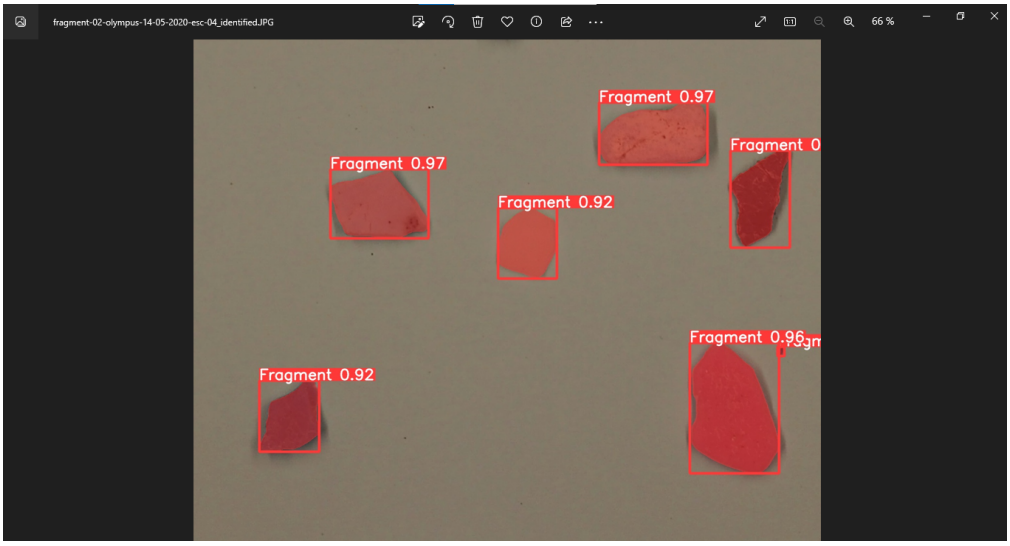


Figura 3.38: Apariencia de la imagen guardada tras la segmentación

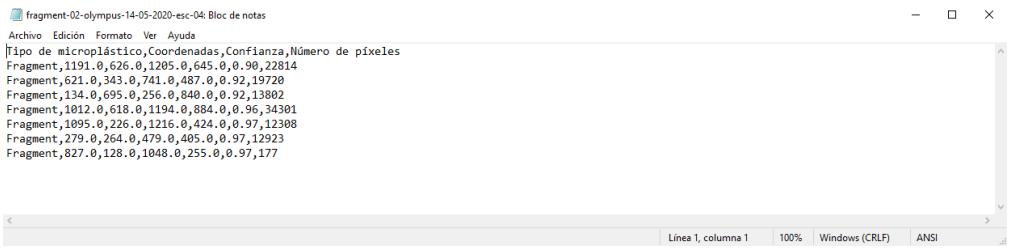


Figura 3.39: Apariencia del fichero tras la exportación

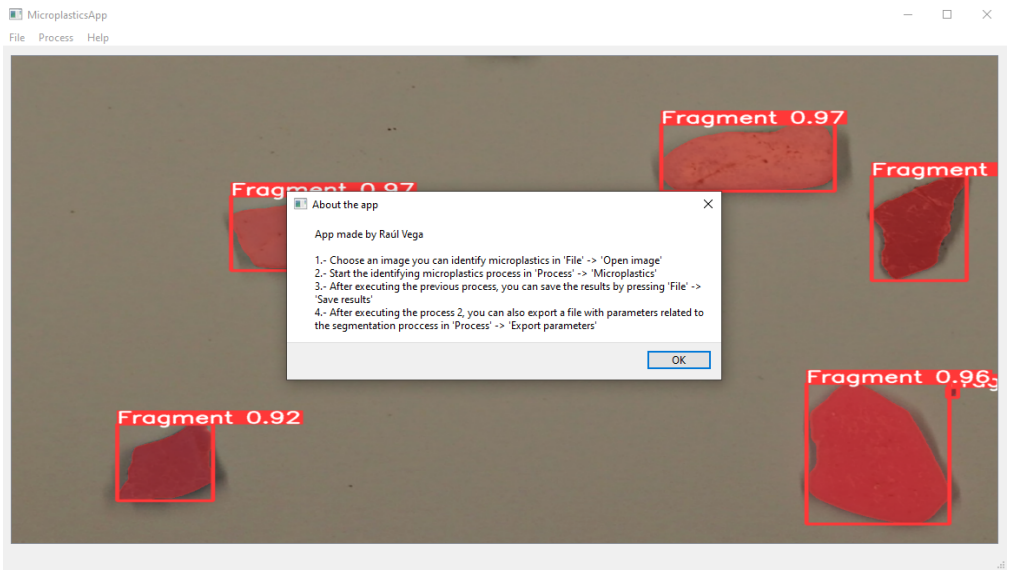


Figura 3.40: Instrucciones mostradas en 'About the app'

Capítulo 4

Experimentos y resultados

En este capítulo se describen los diferentes experimentos que se han realizado entrenando la red y analizando los resultados obtenidos. Tras la primera iteración de prueba con un conjunto de datos cuya cantidad de muestras era demasiado pequeña, se pretendió crear un conjunto de muestras con una mayor cantidad de imágenes de cara a obtener resultados más favorables y precisos. Tras obtener este conjunto de datos más completo y robusto con un total de 524 muestras, se realizaron los siguientes experimentos:

4.1. Primer experimento y análisis de resultados

Primeramente, se ha procedido a realizar una primera iteración, realizando una división del conjunto de imágenes 70/30, siendo uno de los *splits* considerados de los más populares y de los que mejores resultados promedios suele obtener [9]. Como hiperparámetros usados a destacar, se pretende hacer uso de 100 épocas y de un tamaño del batch de 8 imágenes. A continuación, se muestra el script utilizado para este entrenamiento:

Algoritmo 4.1: Script primera iteración entrenamiento red neuronal

```
!python segment/train.py --data "/content/microplasticos-dataset70-30/images/data.
    yaml" --weights yolov5s-seg.pt --img 1280 --batch-size 8 --epochs 100 --
    project "/content/gdrive/MyDrive/TFG" --name "100epochs-batch8-weights-data-
    augmentation-completed-70-30"
```

Tras la finalización de este primer entrenamiento, se generan paralelamente ciertos archivos relativos que contienen diferentes parámetros en base a los resultados obtenidos. De todos estos archivos, destacamos el archivo llamado ”results.png” y la matriz de confusión generada para comparar resultados y generar conclusiones.

El archivo llamado ”results.png” contiene gráficos de diferentes métricas captadas durante todo el entrenamiento: la precisión en detecciones de las cajas delimitadoras y de las

máscaras aplicadas encontradas en los parámetros **(metrics/precision(B)** y **metrics/precision(M))**, el *recall* o exhaustividad en detecciones de cajas delimitadoras y máscaras aplicadas generadas en los parámetros **(metrics/recall(B)** y **metrics/recall(M))**, diferentes datos referentes a las pérdidas sobre el conjunto de entrenamiento y el de validación consolidados en los parámetros **(train/box_loss, train/seg_loss, train/obj_loss, train/cls_loss, val/box_loss, val/seg_loss, val/obj_loss** y **val/cls_loss)**, y variables relacionados con valores medio de precisión media, abreviado como *mAP* apreciadas en los parámetros **(metrics/mAP_0.5(B), metrics/mAP_0.5:0.95(B), metrics/mAP_0.5(M)** y **metrics/mAP_0.5:0.95(M))**. Dentro de toda esta información generada, nos fijaremos especialmente en los valores del mAP_0.5, el cual es considerada como una de las métricas más utilizadas en la detección de objetos, para evaluar la calidad de las predicciones [17]. A continuación, se explican brevemente las métricas con mayor relevancia:

La precisión es una métrica que nos brinda la proporción de predicciones correctas realizadas por el modelo.

La exhaustividad es una métrica que mide la proporción de instancias positivas que son correctamente clasificadas por el modelo.

El promedio de precisión media (comúnmente denominado como mAP, por sus siglas en inglés) es una métrica que mide el rendimiento general del modelo en cuestión tomando en cuenta tanto la precisión como la exhaustividad.

En la matriz de confusión, apreciamos en el eje X los tipos de microplásticos reales y en el eje Y los tipos de los microplásticos predecidos, junto con una clase adicional generada automáticamente por la herramienta en cada eje denominada "background". Cuya intención es clasificar objetos, que el detector no detecta y que se consideran como otros objetos de fondo (background definido en el eje X), casos categorizados como falsos negativos. O identificar objetos de fondo que no pertenecen a ninguna de las clases pero que se detectan como una de ellas (background definido en el eje Y), casos categorizados como falsos positivos. Seguido de esto, apreciamos la misma estructura de una matriz de confusión convencional, donde la diagonal principal correspoonde con los valores estimados de forma correcta por el modelo, tanto los verdados positivos, como los verdaderos negativos. A partir de esta matriz diagonal, se aprecian los falsos positivos en el resto de valores de la primera columna y los falsos negativos en el resto de valores de la primera fila. El resto de valores no mencionados anteriormente, se categorizarían como verdaderos negativos nuevamente.

En la Figura 4.1, apreciamos gráficamente lo explicado en el párrafo anterior.

Tras la ejecución del script mencionado previamente , se puede apreciar gráficamente los resultados generados en este entrenamiento en las Figuras 4.2 y 4.3.

Apreciamos que se obtienen unos resultados muy precisos como una primera iteración a partir de un conjunto de datos completo, obteniendo resultados como verdaderos positivos que rondan entre el 97 %-100 %. En adición, apreciamos una ínfima cantidad de falsos negativos para las clases 'Line' y 'Pellet' que como máximo rondan un 3 %. En contrapartida, se observan resultados menos favorables en cuanto a falsos positivos del background. Especialmente en la clase 'Pellet' y 'Line', donde se llegan a obtener un porcentaje de 76 % en este sentido.

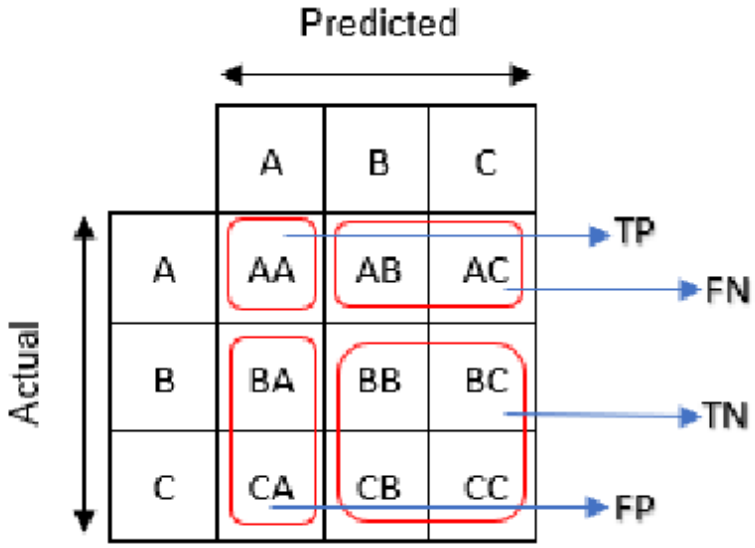


Figura 4.1: Explicación gráfica matriz de confusión

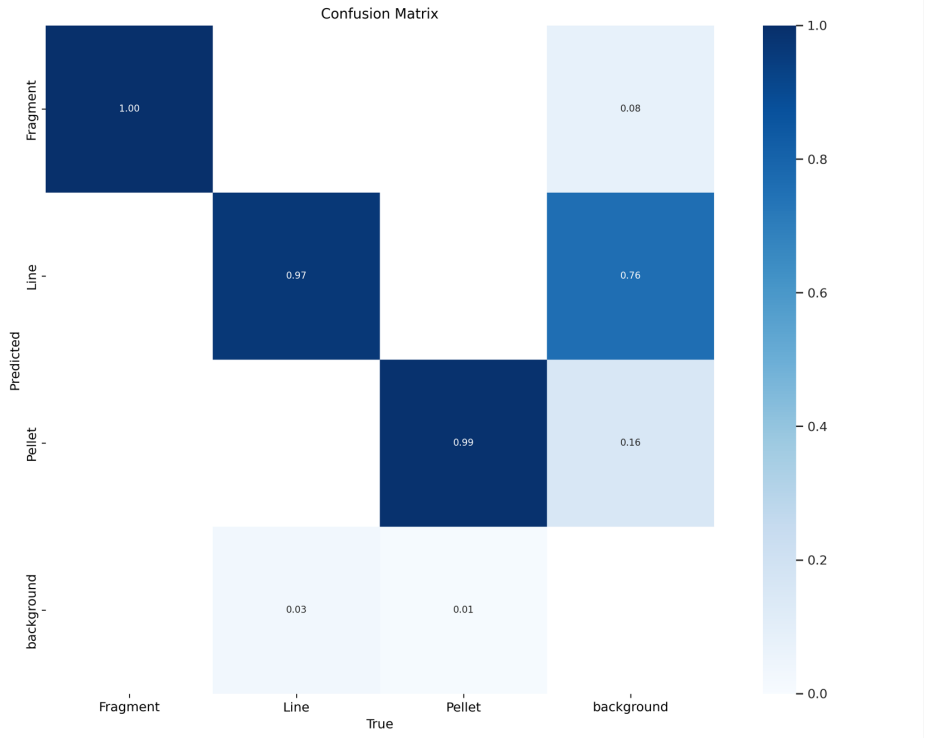


Figura 4.2: Matriz de confusión primer entrenamiento

En cuanto al mAP, manejamos unos números de aproximadamente 0.99 y 0.97 para este entrenamiento.

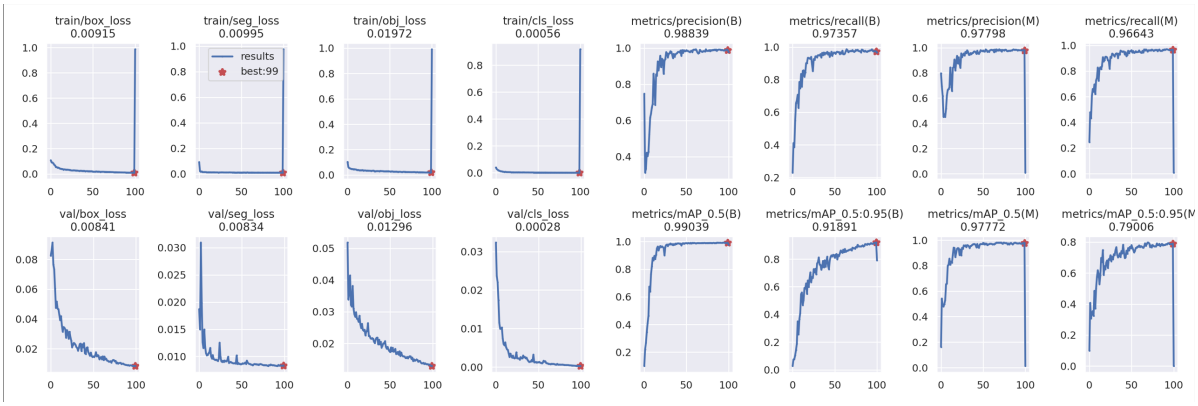


Figura 4.3: Métricas del primer entrenamiento

4.2. Segundo experimento y análisis de resultados

A continuación, se procede a realizar un segundo entrenamiento con un nuevo enfoque. Concretamente, se pretende organizar el conjunto del conjunto de imágenes con una división 80/20 en búsqueda de apreciar mejores resultados a los obtenidos previamente.

A continuación, se pretende especificar el script que se quiere ejecutar:

Algoritmo 4.2: Script segunda iteración entrenamiento red neuronal

```
!python segment/train.py --data "/content/microplasticos-dataset80-20/images/data.
    yaml" --weights yolov5s-seg.pt --img 1280 --batch-size 8 --epochs 100 --
    project "/content/gdrive/MyDrive/TFG" --name "100epochs-batch8-weights-data-
    augmentation-completed-80-20"
```

Los resultados obtenidos referente a este segundo entrenamiento, aparecen en la Figuras 4.4 y 4.5.

Se obtienen resultados ligeramente más precisos que en la anterior iteración en el apartado de verdaderos positivos rondando entre el 98 %-100 %. Se aprecian concentrados los resultados de falsos negativos en la clase 'Line' con un 2 %. Y se concentran también los valores de falsos positivos en la clase 'Line' creando confusión con el fondo de las muestras, obteniendo un alto 89 %. En cuanto al mAP, manejamos unos números de aproximadamente 0.99 y 0.98 para este entrenamiento. Concluímos que pasar del 70/30 al 80/20 en división del conjunto de datos, no ha proporcionado unas diferencias exageradamente notorias, principalmente porque ya se parten con buenos resultados. Pero si, ligeramente mejores.

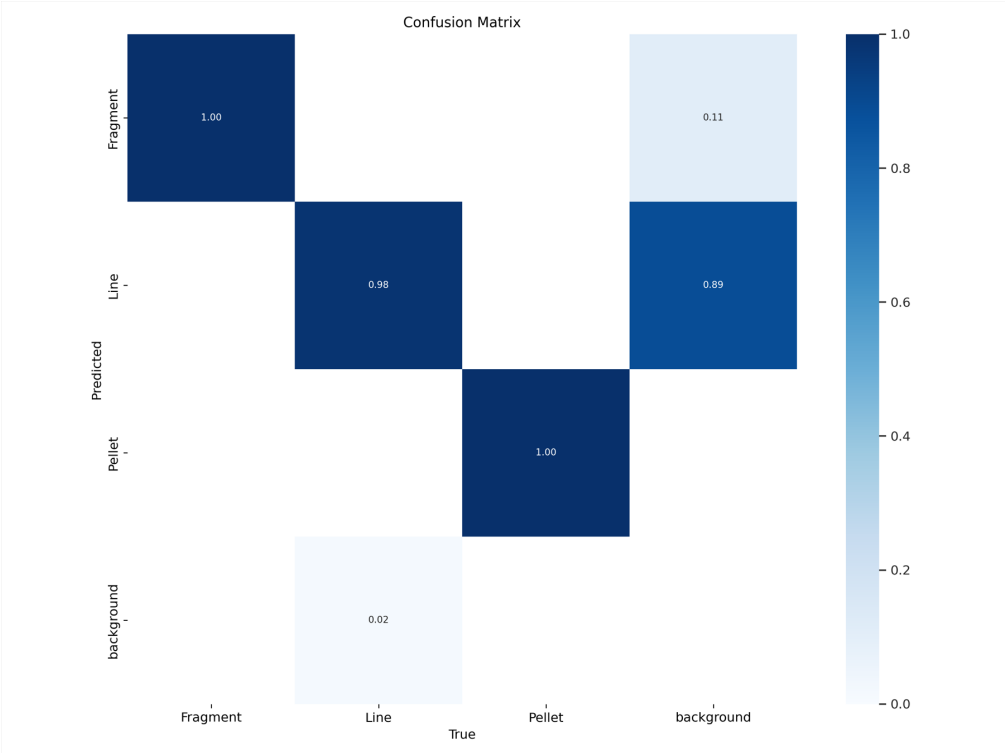


Figura 4.4: Matriz de confusión segundo entrenamiento

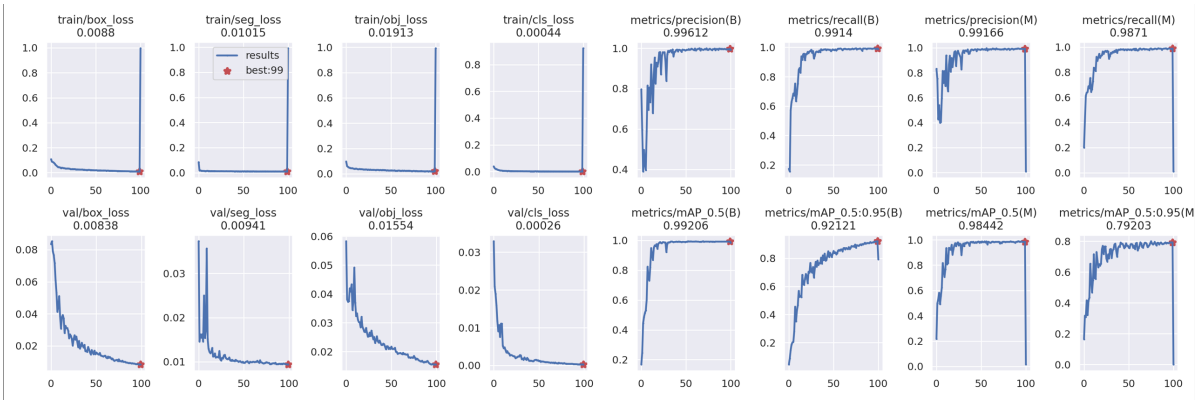


Figura 4.5: Métricas del segundo entrenamiento

4.3. Tercer y cuarto experimento y análisis de sus resultados

A partir de aquí, pretendemos realizar una mejora de los anteriores intentos realizados, a partir de un aumento en el número de épocas definidos en el entrenamiento. A continuación, se muestra el tercer script ejecutado:

Algoritmo 4.3: Script tercera iteración entrenamiento red neuronal

```
!python segment/train.py --data "/content/microplasticos-dataset70-30/images/data.yaml" --weights yolov5s-seg.pt --img 1280 --batch-size 8 --epochs 200 --project "/content/gdrive/MyDrive/TFG" --name "200epochs-batch8-weights-data-augmentation-completed-70-30"
```

La información relativa a los resultados conseguidos para este entrenamiento, se localiza en las Figuras 4.6 y 4.7.

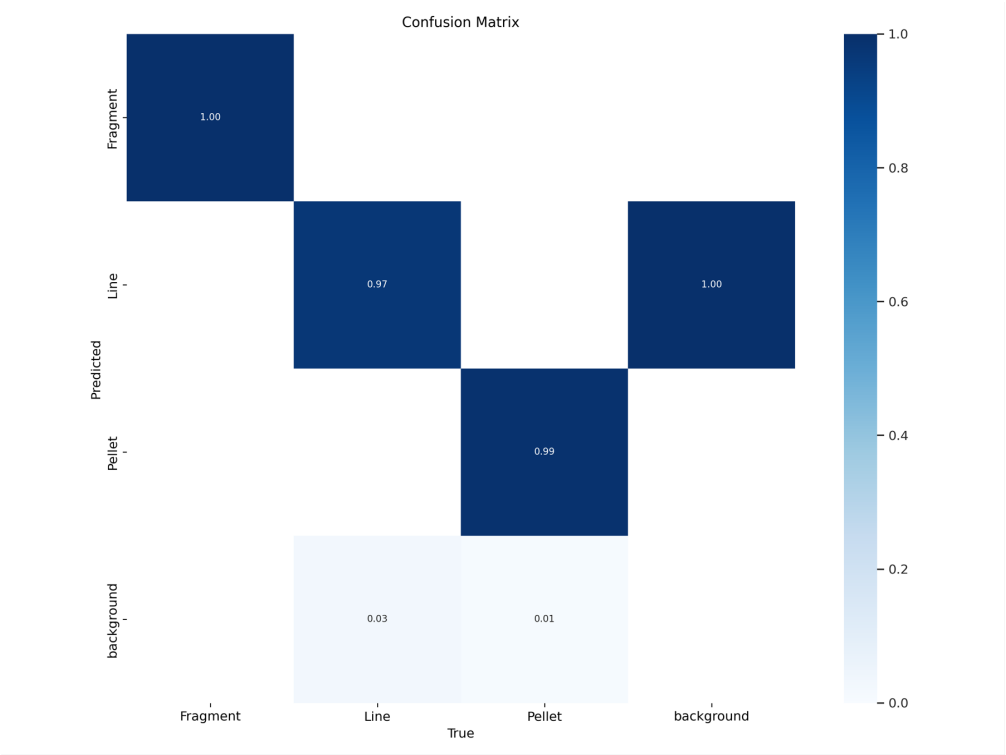


Figura 4.6: Matriz de confusión tercer entrenamiento

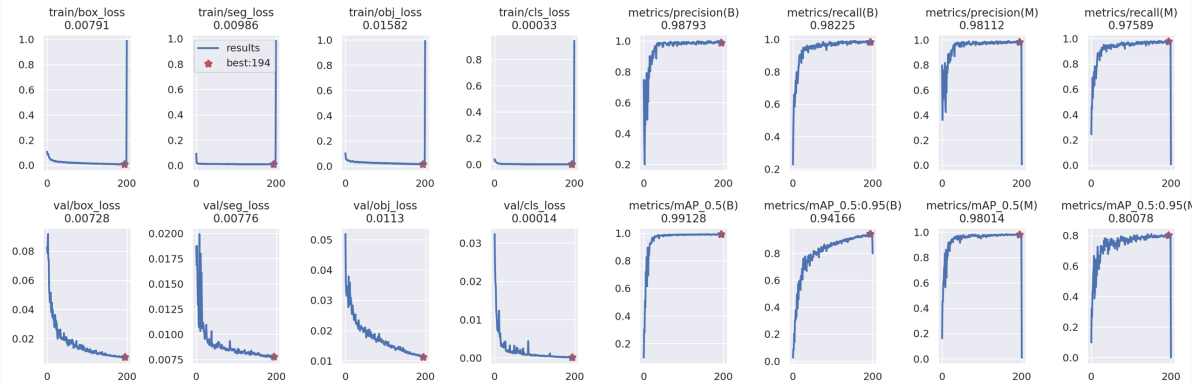


Figura 4.7: Métricas del tercer entrenamiento

En base a los resultados obtenidos en este nuevo enfoque, no apreciamos mejora en los resultados, a partir de la duplicación en el número de épocas durante el entrenamiento.

Mantenemos un intervalo del 97 %-100 % para la precisión. Destacando también la existencia ínfima de algunos falsos negativos del 3 % en Lines y 1 % en Pellets. Se concentra también el valor de falso positivo en la clase 'Line' creando confusión con el fondo de las muestras, obteniendo un pleno 100 %. En cuanto al mAP, manejamos unos números de aproximadamente 0.99 y 0.98 para este entrenamiento.

Para la última iteración a ejecutar y experimentar, hacemos uso de este script:

Algoritmo 4.4: Script última iteración entrenamiento red neuronal

```
!python segment/train.py --data "/content/microplasticos-dataset80-20/images/data.
    yaml" --weights yolov5s-seg.pt --img 1280 --batch-size 8 --epochs 200 --
    project "/content/gdrive/MyDrive/TFG" --name "200epochs-batch8-weights-data-
    augmentation-completed-80-20"
```

Los resultados del último entrenamiento efectuado, se encuentran en las Figuras 4.8 y 4.9.

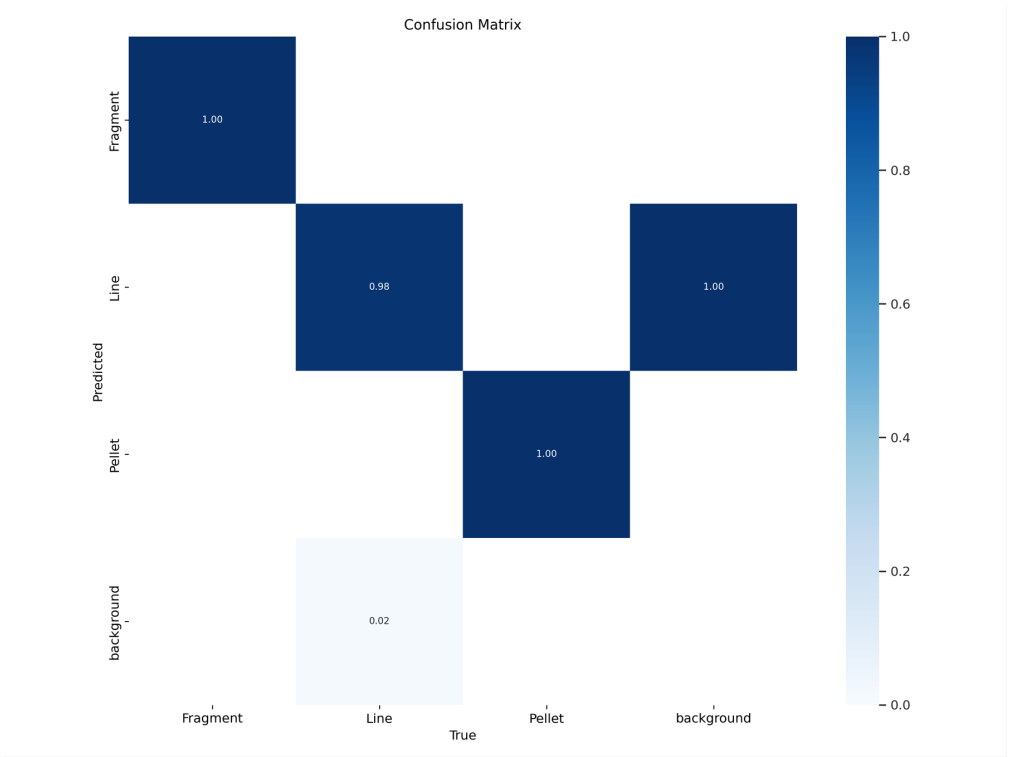


Figura 4.8: Matriz de confusión último entrenamiento

En base a los resultados obtenidos en este nuevo enfoque, tampoco apreciamos mejora en base a la duplicación en el número de épocas durante el entrenamiento.

Mantenemos un intervalo del 98 %-100 % para la precisión. Destacando también la existencia ínfima de algunos falsos negativos del 2 % en Lines. En cuanto a falsos positivos, se concentra también en la clase 'Line' creando confusión con el fondo de las muestras, obteniendo un pleno 100 %. En cuanto al mAP, manejamos unos números de aproximadamente 0.99 y 0.98 para este entrenamiento.

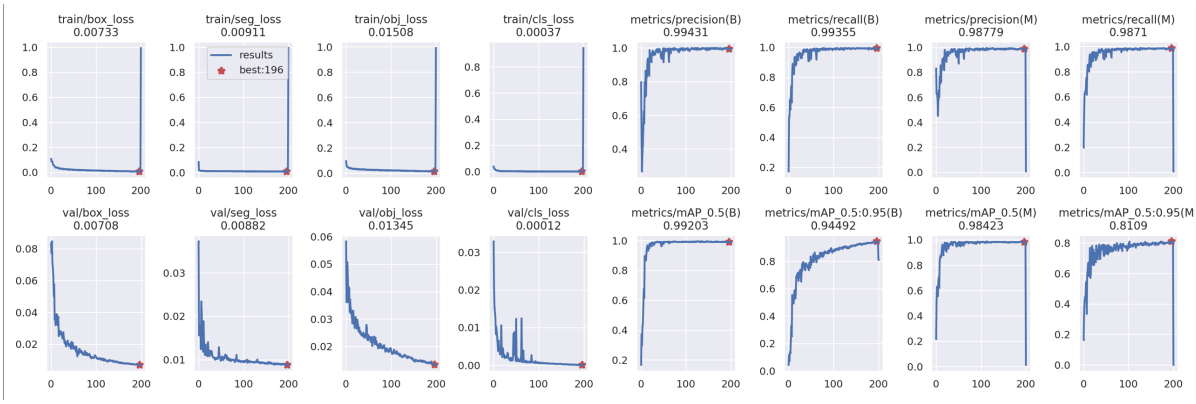


Figura 4.9: Métricas del último entrenamiento

4.4. Elección del modelo óptimo y resultados de su ejecución

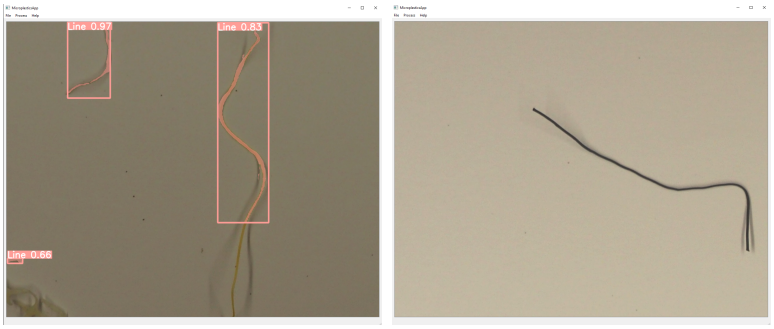
Teniendo esto en cuenta, optaremos por hacer uso del segundo modelo entrenado, correspondiente con la división del conjunto de datos 80/20 y con 100 épocas, donde se aprecian ligeramente mejores resultados con respecto a los otras iteraciones. Especialmente, en cuanto a las comparaciones del parámetro mAP.

La ejecución de la aplicación con la integración del modelo de inferencia resultante con este último modelo entrenado, resulta en resultados muy precisos de manera general, apreciándose en el grado de confianza que aparece al lado de la segmentación. Algunos de estos ejemplos se muestran en las figuras 3.24a, 3.24b, 3.24c. Pero también es importante anotar, que existen ciertos casos a mencionar donde la precisión y rendimiento del modelo de inferencia no es suficiente para clasificar con éxito. Las situaciones son las siguientes:

- ✔ Se aprecian varios casos de muestras de tipo Pellets, en los que ciertos microplásticos son clasificados como Fragments, debido a que son presentados con rasgos no tan circularmente estructurados.
- ✔ Existen algunas situaciones en la segmentación de microplásticos de tipo Line, donde la inferencia no resulta del todo precisa o efectiva. De modo que resulta en la no identificación o identificación parcial de estos elementos.

  Estos casos se pueden ver gráficamente en las viñetas de la Figura 4.10.

(a) Falsos positivos de pellets clasificados como fragments



(b) Imprecisiones en clasificación de cebos de pesca

Figura 4.10: Casos de clasificaciones erróneas por la inferencia

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

Tras el desarrollo de este proyecto, contamos con una aplicación de escritorio totalmente funcional para la identificación y recuento de microplásticos, diseñada para ser usada por usuarios que no requieran de conocimientos técnicos para el empleo del mismo. Gracias a la integración de una interfaz gráfica simple e intuitiva.

Además, este producto final que consigue cumplir con todos los objetivos propuestos en la Sección 1.2. Podrá ser cedido para todos aquellos investigadores que se encuentan inmersos en aportar su granito de arena en la mejora del medioambiente a través de su conocimiento en combinación con el uso de esta aplicación. Especialmente, para los biólogos de nuestras islas, los cuales han estado en mente como usuarios finales de nuestra aplicación, desde el comienzo de la planificación de este proyecto.

Se espera que el uso de esta aplicación pueda permitir un conteo más exhaustivo, automático y detallado de los microplásticos que desembocan en nuestras playas. Permitiendo la obtención del número de microplásticos que rondan las zonas desde las que se fotografían las muestras y se procesan las imágenes. Con la misión de comparar números a nivel global de la presencia de estos microplásticos en el medioambiente, con respecto a años anteriores. De modo que se puedan obtener conclusiones sobre si las diferentes gestiones y planes aprobados por las instituciones pertinentes a nivel mundial, están resultando útiles y efectivas en contra de este problema grave y universal.

A nivel personal, el desarrollo de este trabajo me ha permitido ampliar horizontes y adquirir conocimientos en ámbitos de la informática, como el Machine Learning, con los que no estaba tan familiarizado. No solo a partir del uso de tecnologías, lenguajes o librerías con las que no había estado tan involucrado a lo largo de la carrera, si no de afrontar ciertas situaciones arraigadas con el desarrollo exitoso del trabajo como el cumplimiento de los objetivos, la gestión de la planificación a seguir, entre otros.

5.2. Trabajo a futuro

En el estadio en el que se concluye este proyecto de fin de título, se puede seguir iterando sobre el mismo a partir de diferentes matices que pueden ser mejorados.

Una de las posibilidades que se proponen es la disminución de los casos no deseados tras la ejecución de la aplicación haciendo uso del modelo de inferencia conseguido, como la existencia de casos como falsos negativos, verdaderos negativos y falsos positivos. Esto se conseguiría a partir de obtener un dataset más robusto mediante la incorporación de un mayor número de muestras que será usado a futuro para entrenar. Este proceso puede conseguirse a partir de la obtención de mayores muestras a partir de las ya existentes mediante la técnica de image augmentation, realizando cambios en parámetros de las imágenes que aún no hayan sido aplicados hasta ahora, como por ejemplo: las rotaciones, los cambios de escalas y los cambios de saturación de las imágenes. Teniendo en cuenta que si han sido utilizadas las técnicas de volteos de imágenes y de cambios en brillo y contraste. En base a los resultados obtenidos, se requeriría de un mayor número de muestras y de una mayor atención para los microplásticos de tipo Line, ya que son los que actualmente obtienen una precisión inferior en la segmentación en base al resto de tipos.

Por último, se podría incorporar un diseño más vistoso a la aplicación en cuanto a estilo. Acción que por nuestra parte ha sido tenida en cuenta vagamente, debido a que existían objetivos a cumplir con mayor preferencia en comparación a este.

En resumen, aún existe margen de mejora en ciertos aspectos para seguir incrementando el producto desarrollado hasta ahora.

Bibliografía

[1] Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., and Kalinin, A. A. (2020). Albumentations: fast and flexible image augmentations. *Information*, 11(2):125.

[2] Enfrin, M., Dumée, L. F., and Lee, J. (2019). Nano/microplastics in water and wastewater treatment processes–origin, impact and potential solutions. *Water research*, 161:621–638.

[3] Estenssoro Saavedra, J. F. (2007). Antecedentes para una historia del debate político en torno al medio ambiente: la primera socialización de la idea de crisis ambiental (1945-1972). *Universum (Talca)*, 22(2):88–107.

[4] Evode, N., Qamar, S. A., Bilal, M., Barceló, D., and Iqbal, H. M. (2021). Plastic waste and its management strategies for environmental sustainability. *Case Studies in Chemical and Environmental Engineering*, 4:100142.

[5] Kim, J.-a., Sung, J.-Y., and Park, S.-h. (2020). Comparison of faster-rcnn, yolo, and ssd for real-time vehicle type recognition. In *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, pages 1–4.

[6] 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.

[7] Mandal, S., Islam, M., and Biswas, M. (2020). Modeling the impact of carbon dioxide on marine plankton. *Int. J. Math. Comput. Sci*, 14:197–202.

[8] Nagpal, A. and Gabrani, G. (2019). Python for data analytics, scientific and technical applications. In *2019 Amity international conference on artificial intelligence (AICAI)*, pages 140–145. IEEE.

[9] Nguyen, Q. H., Ly, H.-B., Ho, L. S., Al-Ansari, N., Le, H. V., Tran, V. Q., Prakash, I., and Pham, B. T. (2021). Influence of data splitting on performance of machine learning models in prediction of shear strength of soil. *Mathematical Problems in Engineering*, 2021:1–15.

[10] Prata, J. C., da Costa, J. P., Lopes, I., Duarte, A. C., and Rocha-Santos, T. (2020). Environmental exposure to microplastics: An overview on possible human health effects. *Science of the total environment*, 702:134455.

[11] Rasband, W. (2019). Image Processing and Analysis in Java. https://imagej.nih.gov/ij/.

[12] Sol, D., Laca, A., Laca, A., and Díaz, M. (2020). Approaching the environmental problem of microplastics: Importance of wwtp treatments. *Science of the Total Environment*, 740:140016.

[13] Thuan, D. (2021). Evolution of yolo algorithm and yolov5: The state-of-the-art object detention algorithm.

[14] Vladislav Efremov, M. (2022). Yolov5 for segmentation. https://medium.com/mlearning-ai/yolov5-for-segmentation-fab39c3487f6.

[15] Wada, K. (2016). Image Polygonal Annotation with Python. https://github.com/wkentaro/labelme/.

[16] Xu, Y. and Goodacre, R. (2018). On splitting training and validation set: a comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *Journal of analysis and testing*, 2(3):249–262.

[17] Zhiqiang, W. and Jun, L. (2017). A review of object detection based on convolutional neural network. In *2017 36th Chinese control conference (CCC)*, pages 11104–11109. IEEE.