

Trabajo de Fin de Grado

Evaluación de técnicas de visión por computador para la clasificación de microplásticos por color

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Alberto Fidalgo Cabantac

TUTORIZADO POR:
Modesto Fernando Castrillón Santana y José Javier Lorenzo Navarro

Mayo 2025

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D./D^a Alberto Fidalgo Cabantac, autor del trabajo Evaluación de técnicas de visión por computador para la clasificación de microplásticos por color, correspondiente a la titulación Grado en Ingeniería Informática

SOLICITA

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida, haciendo constar que

☒ [X] se autoriza / ☐ [] no se autoriza la grabación en audio de la exposición y turno de preguntas.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

☐ [] Se ha iniciado o hay intención de iniciarlo (defensa no pública).

☒ [X] No está previsto.

Y para que así conste firma la presente.

El/La estudiante

Fdo. _____

A rellenar y firmar **obligatoriamente** por el/la/los/las tutores

En relación con la presente solicitud, se informa: (firmar donde corresponda)

Positivamente (en caso de detección de copia, esta firma quedará invalidada)

Fdo. _____

Negativamente (justificación en TFT05)

Fdo. _____

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Agradecimientos

A mis tutores, Modesto y José Javier, por su constante guía, ayuda y respuestas a infinitas preguntas y paciencia con la cual este proyecto no hubiera podido ser posible. Y a mi familia y amigos, por el apoyo constante y a impulsarme a seguir mi camino.

Resumen

Pese a las muchas ventajas del plástico como material, su amplio uso en todo el mundo causa a día de hoy múltiples problemas en el ecosistema mundial, siendo uno de ellos la presencia de microplásticos en varias zonas naturales. Una de las labores que componen la lucha contra la contaminación de microplásticos en los entornos consiste en la cuenta, catalogación e identificación de estos para mejor comprender su presencia.

En este proyecto se ha llevado a cabo una breve investigación sobre distintas tecnologías existentes de visión de computador mediante modelos de redes neuronales para el desarrollo de una aplicación que detecte, cuente y catalogue microplásticos basados en su color, sin la necesidad de entrenar un modelo nuevo. Esta aplicación ayudaría al proceso de medición de contaminación por microplásticos en zonas medioambientales.

Abstract

Despite the many advantages of plastic as a material, its widespread use around the world causes numerous problems for the global ecosystem, one of which is the presence of microplastics in various natural areas. One of the tasks in the fight against microplastic pollution in environments involves the counting, cataloging, and identification of these particles to better understand their presence.

In this project, a brief investigation was carried out on various existing computer vision technologies using neural network models for the development of an application that detects, counts, and catalogs microplastics based on their color, without the need to train a new model. In theory, this application would help in the process of measuring microplastic pollution in environmental areas.

Índice general

1. Introducción	1
1.1. Motivación y contexto del proyecto	1
2. Estado actual y objetivos iniciales	3
2.1. Descripción general y objetivos	3
2.2. Plan de trabajo	4
3. Aportaciones del trabajo	6
3.1. Principales aportaciones	6
3.2. Competencias específicas	6
3.3. Alineamiento con los objetivos de desarrollo sostenible	7
4. Estado del arte	8
4.1. Visión por computador y redes neuronales	8
4.2. Segmentado de imágenes	10
4.2.1. Mask R-CNN	12
4.2.2. YOLO	12
4.2.3. Segment Anything Model	13
4.3. Modelos de lenguaje visual	14
4.3.1. BLIP	14
5. Herramientas y recursos	16
5.1. Entorno de desarrollo	16
5.1.1. Python	16
5.1.2. Jupyter Notebook y Markdown	17
5.1.3. Pytorch	17
5.1.4. LAVIS	18
5.1.5. VSCode	18
5.1.6. Especificaciones del equipo	18
5.2. Modelos	19
5.3. Bibliotecas importantes	19
5.3.1. Supervision	19
5.3.2. Webcolors	19
5.3.3. QTDesigner	19

6. Desarrollo	21
6.1. Estudio previo	21
6.1.1. Familiarización con SAM	21
6.1.2. Familiarización con LAVIS	24
6.1.3. Conjuntos de datos	26
6.2. Pipeline básico	27
6.2.1. Resultados	29
6.3. Migración a diseño OOP	31
6.4. Módulos implementados	33
6.4.1. Balance de blanco	33
6.4.2. Función de superposición	37
6.4.3. Escalado de imagen	39
6.4.4. Filtrado de máscaras	41
6.4.5. Uso de parámetros	41
6.5. Ajuste fino del modelo	44
6.5.1. Preparado de datos	44
6.5.2. Entrenamiento	45
6.6. Interfaz de usuario	47
6.6.1. Línea de comando	48
6.6.2. Interfaz gráfica de usuario	48
7. Conclusiones	50
Bibliografía	53
Anexo - Manual de usuario	54
Instalación	54
Uso	56
Línea de comando	56
Interfaz gráfica	57

Índice de figuras

1.1. Foto de bastoncillos y otros residuos plásticos	2
4.1. Ejemplo de detección de objetos por visión por computador	9
4.2. Diagrama de red neuronal artificial	10
4.3. Segmentado de imagen semántico de una calle	11
4.4. Máscaras de segmentación semántica	12
4.5. Segmentación de imágenes por prompt en SAM	13
4.6. Proceso de Segment Anything Model	14
4.7. Funcionamiento de VQA	15
5.1. Ejemplo de celdas de código en Jupyter Markdown	17
5.2. Desarrollo en QtDesigner	20
6.1. Demo SAM 1	22
6.2. Demo SAM 2	23
6.3. Demo SAM 3	24
6.4. Estatua de merlion	24
6.5. Demo LAVIS BLIP 1	25
6.6. Conjunto de datos de microplásticos	26
6.7. Diagrama del pipeline básico	27
6.8. Código de recorte de microplásticos	27
6.9. Imágenes de microplásticos de prueba	28
6.10. Imágenes de microplásticos segmentados	29
6.11. Primer resultado de segmentación de microplásticos	29
6.12. Diagrama del modelo OOP	31
6.13. Ejemplo 1 del algoritmo mundo gris	34
6.14. Ejemplo 2 del algoritmo mundo gris	34
6.15. Ejemplo 3 del algoritmo mundo gris	34
6.16. Ejemplo 1 del algoritmo mundo gris 2	35
6.17. Ejemplo 2 del algoritmo mundo gris 2	35
6.18. Ejemplo 3 del algoritmo mundo gris 2	35
6.19. Ejemplo 1 de algoritmo parche blanco	36
6.20. Ejemplo 2 de algoritmo parche blanco	36
6.21. Ejemplo 3 de algoritmo parche blanco	36
6.22. Ejemplo de superposición de máscaras	37

6.23. Resultado de modulo de superposición de máscaras	38
6.24. Código de modulo de escalado	39
6.25. Múltiples microplásticos sobre una sola máscara	41
6.26. Parámetros de la interfaz SAM	42
6.27. Segmentación con todos los módulos activos 1	43
6.28. Segmentación con todos los módulos activos 2	43
6.29. Ejemplo de preparado de máscaras de entrenamiento	45
6.30. Código de ajuste fino 1	45
6.31. Ejemplo de una entrada de entrenamiento	46
6.32. Código de ajuste fino 2	46
6.33. Código de ajuste fino 3	46
6.34. Comparativa de funciones de pérdida	47
6.35. Ejecución del comando ayuda del programa	48
6.36. Interfaz gráfica de usuario	48
7.1. Instalación de Pytorch y Torchvision	54
7.2. Instalación desde CMD de Windows - Supervision	55
7.3. Uso de aplicación por CMD de Windows	56
7.4. Uso de aplicación por interfaz gráfica	57

Índice de cuadros

2.1. Plan de desarrollo	4
3.1. Grado de relación del TFG con los objetivos de desarrollo sostenible.	7
5.1. Especificaciones de equipo	18
6.1. Resultados de segmentación de primera imagen	30
6.2. Resultados de segmentación de segunda imagen	30
6.3. Resultados de segmentación de tercera imagen	30
6.4. Resultados de segmentación y clasificación con y sin balance de blanco	37
6.5. Resultados de segmentación y clasificación con y sin modulo de superposición	39
6.6. Resultados de segmentación y clasificación con aumento de escalado	40
6.7. Resultados de segmentación y clasificación con reducción de escalado	40
6.8. Resultados de segmentación y clasificación con y sin todos los módulos	43

Capítulo 1

Introducción

1.1. Motivación y contexto del proyecto

Se espera que la cantidad de plástico en el océano se duplique en los próximos 15 años, y se estima que para 2050 podría haber más plástico en el océano que peces (por peso).

*(Ellen MacArthur Foundation
2016). [1]*

Desde su invención y descubrimiento, uno de los materiales responsables del rápido desarrollo humano en los últimos 100 años y sinónimo de la era de la industrialización ha sido el plástico. Si bien sus propiedades beneficiosas para el desarrollo son múltiples tales como su fácil manufacturación, versatilidad, durabilidad y resistencia, y sobre todo eficiencia en coste, también lleva consigo un igual número de propiedades adversas. Una de las más importantes es el gran papel que tiene el plástico en la contaminación de los entornos naturales. Debido a que el plástico es un material que no biodegrada, se mantiene largos periodos de tiempos en la biosfera, contaminando el medio ambiente, filtrando sustancias tóxicas al subsuelo o incluso dañando a la vida animal que por error pueda llegar a consumir dicho material tóxico. Asimismo, el plástico también se vuelve frágil con el tiempo, y rompe en fragmentos de diversos tipos, son estos los llamados microplásticos que están causando gran daño a la biosfera, y en especial a los entornos marinos y de playa.

Actualmente la Unión Europea tiene varias medidas con el objetivo de prevenir y recortar la gran filtración de microplásticos a los entornos marinos, por medio de iniciativas que combaten la producción de microplásticos añadidos o la liberación involuntaria de microplásticos al medio ambiente [14][15]. Hoy día, no existe ninguna ley europea que abarque comprensivamente a los microplásticos, si bien se quiere reducir en un 30 por ciento su producción.



Figura 1.1: Foto de bastoncillos y otros residuos plásticos encontrados en la zona de La Cícer, en la playa de Las Canteras, en Las Palmas de Gran Canaria. (ULP GC [10])

Uno de los procesos llevados a cabo en la lucha contra la contaminación causada por los microplásticos, como se puede observar en la Figura 1.1 es la constante monitorización de las zonas potencialmente afectadas, realizando estudios sostenidos y frecuentes para la medición de cantidad de microplásticos, categorizándolos por variedad de tipos, e introduciendo dichos datos para su análisis estadístico, el cual ha tenido investigaciones para su automatización con redes neuronales [11]. Es gracias a este tipo de monitorización constante, junto con otros muchos datos científicos como el movimiento de corrientes oceánicas y marinas, o localidad de filtraciones de microplásticos importantes, que los investigadores son capaces de señalar aquellos sitios, con una consideración especial en las zonas de playas, los entornos más afectados y su procedencia. Como es natural en este tipo estudios, a mayor cantidad de datos mayor es la fiabilidad de conclusión, y mayor la capacidad de producción, para así ayudar a las posibles iniciativas que busquen lidiar con la contaminación de microplásticos.

Asimismo, de las disciplinas con mayor crecimiento en el ámbito tecnológico es la visión por computador, y con igualdad de importancia las redes neuronales. Estas dos tecnologías han crecido en desarrollo en los últimos años, y los numerosos usos que a cada día son más, hacen ver claramente que son importantes herramientas de cara al futuro para el desarrollo humano y tecnológico. La gran cantidad de aplicaciones, su versatilidad y su dominio de la visión por computador y detección en imágenes hacen de estas tecnologías un candidato idóneo para el desarrollo de este proyecto.

Capítulo 2

Estado actual y objetivos iniciales

2.1. Descripción general y objetivos

Durante la tarea de monitorización de contaminantes por microplásticos en las playas, una de las tareas realizadas consiste en el proceso de recogida de muestras de playas, su filtrado para extracción de microplásticos, recuento de cantidad de microplásticos encontrados, categorización en base a varios factores, y por último la introducción de los datos recogidos en base de datos [6]. Sin ningún tipo de tecnología de por medio, este proceso se lleva a cabo manualmente por los investigadores al cargo. Es el recuento y categorización de microplásticos donde nace la primera función de este proyecto, el cual plantea la sustitución de conteo manual y categorización por colores por una aplicación que realice la tarea de forma más rápida y eficientemente, ayudando a agilizar el proceso de observación de microplásticos en playas.

Siendo la labor una de visión de computador, realizarla usando las crecientes tecnologías de redes neuronales nace naturalmente. Siendo este uno de los campos más florecientes en el ámbito tecnológico actualmente, se deriva el segundo objetivo de este proyecto. La breve investigación de los ya existentes modelos de redes neuronales que puedan ser usados o combinados para el rápido desarrollo de aplicaciones para acometer las tareas ya descritas, y medir su eficacia y coste.

Son estos dos objetivos principales desde los que nace el origen de este proyecto, y la razón que existe detrás de él. Abarcarlos, por lo tanto, se convierte en las tareas que se proceden a acometer y toda la planificación subyacente se centra en cumplir estos dos temas centrales al proyecto.

La aplicación a desarrollar cuenta con dos principales funcionalidades, la localización y recuento de microplásticos en imágenes sacadas en un entorno de laboratorio, donde los microplásticos han sido filtrados y extendidos en un fondo de papel para su cuenta manual, y la categorización de los microplásticos detectados en base a su color. Esto se llevará a cabo usando modelos de redes neuronales ya existentes para el inmediato desarrollo.

2.2. Plan de trabajo

El plan de trabajo del proyecto, desglosado en la Tabla 2.1, se divide en tres fases fundamentales.

En la primera, se realizará un estudio previo de las tecnologías y herramientas que mejor encajen con el desarrollo del proyecto. Se hará una breve investigación acerca del funcionamiento de los recursos que compondrán el proyecto, su viabilidad dentro de este, y sus recursos. Una vez familiarizado con las herramientas a utilizar, se pondrán en práctica con una primera fase a modo de comprobación de los recursos y los conocimientos adquiridos.

Durante la segunda fase, habiendo decidido el entorno de desarrollo del proyecto, se procede a desarrollar las funcionalidades básicas e importantes de la aplicación, cumplimentando de la mejor forma posible los objetivos ya anteriormente descritos. Esta fase se plantea como la más costosa en recursos y la más larga, y se llevará a cabo con un desarrollo que se asemejara a un proceso iterativo, ampliando constantemente, solucionando los problemas que aparezcan durante el desarrollo, y si es necesario realizando investigaciones adicionales sobre posibles tecnologías para su uso en el proyecto. Para poder realizar un seguimiento de los objetivos durante esta fase, se plantea realizar evaluaciones constantes de la aplicación que ayuden a indicar la calidad y cercanía con la cual nos acercamos a los objetivos planteados, métricas familiares en la visión por computador tales como tiempo de ejecución o precisión de los resultados usando un conjunto preparado de datos como apoyo.

Por último, durante la tercera fase una vez desarrolladas las funcionalidades de la aplicación, se procede a desplegar los prototipos para uso de usuarios. Durante esta fase, se plantearán varios métodos de despliegue para uso cotidiano de usuario sin entrenamiento que permitan su uso fácil tales como aplicaciones web, aplicaciones por línea de comando, o entorno gráfico ejecutable.

Fase	Tareas
1. Estudio previo / Análisis	1.1: Estudio de tecnologías para el modelado del color 1.2: Familiarización con las bibliotecas 1.3: Familiarización con técnicas y evaluación de clasificación
2. Diseño / Desarrollo / Implementación	2.1: Preparación de datos 2.2: Desarrollo de modelos de clasificación basados en diversas tecnologías
3. Evaluación / Validación / Prueba	3.1: Diseño experimental 3.2: Evaluación de resultados 3.3: Despliegue de prototipo web

Tabla 2.1. Plan de desarrollo

Es de importancia destacar que durante el desarrollo del TFG se estudiaron varias posibilidades de despliegue de prototipo web, pero ninguna de ellas eran realizables con recursos y

tiempos razonables, luego se tomó la decisión de desplegar el programa para ser descargable por los usuarios.

Capítulo 3

Aportaciones del trabajo

3.1. Principales aportaciones

Las principales aportaciones de este proyecto son multiples, ya discutidas brevemente con anterioridad. Por un lado, la creación de un programa que ayude a la lucha contra la contaminación global, y mas específicamente la de la contaminación de microplásticos encontrados crecientemente en todo tipo de entornos naturales y urbanos. Seguidamente, la ayuda al desarrollo del ámbito de Ciencias del Mar, y a la creciente necesidad de herramientas que puedan automatizar el seguimiento de contaminación de microplásticos en entornos marinos y de playa, tanto para un seguimiento constante como para el desarrollo de labores de investigación. Y finalmente, mediante el uso de herramientas y tecnologías ya existentes y a la vanguardia, la aportación de investigación de tales ámbitos en entornos reales y de investigación, y su viabilidad y facilidad de uso y acoplamiento.

3.2. Competencias específicas

Las competencias específicas aplicadas durante este TFG son las siguientes:

CI15 - Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica : Siendo un trabajo de fin de título centrado principalmente en la investigación y uso de redes neuronales y sistemas inteligentes, y su aplicación en entornos prácticos, esta competencia específica es una utilizada ampliamente durante el desarrollo de la práctica en varios niveles de profundidad, tanto de comprensión de los sistemas inteligentes utilizados como la modificación y ajuste de estos para las tareas pertinentes al TFG.

3.3. Alineamiento con los objetivos de desarrollo sostenible

Puesto que este TFG lidia con la mejora de identificación y clasificación de microplásticos en playas para su posible comprensión y tratamiento, tiene un alto grado de relación con los ODS en relación a la contaminación, consumo sostenible o salud de la población, tales como:

- 3. Salud y bienestar
- 6. Agua limpia y saneamiento
- 9. Industria, innovación e infraestructuras
- 11. Ciudades y comunidades sostenibles
- 12. Producción y consumo sostenibles

Un desglosado completo del alineamiento con los objetivos de desarrollo sostenible se puede encontrar en la Tabla 3.1.

ODS	Grado de relación			
	0 No procede	1 Bajo	2 Medio	3 Alto
1 Fin de la Pobreza	X			
2 Hambre cero	X			
3 Salud y bienestar			X	
4 Educación de calidad	X			
5 Igualdad de género	X			
6 Agua limpia y saneamiento				X
7 Energía Asequible y no contaminante		X		
8 Trabajo decente y crecimiento económico	X			
9 Industria, innovación e infraestructuras				X
10 Reducción de las desigualdades	X			
11 Ciudades y comunidades sostenibles				X
12 Producción y consumo sostenibles				X
13 Acción por el clima		X		
14 Vida submarina	X			
15 Vida de ecosistemas terrestres			X	
16 Paz, justicia e instituciones sólidas	X			
17 Alianzas para lograr objetivos	X			

Tabla 3.1. Grado de relación del TFG con los objetivos de desarrollo sostenible.

Capítulo 4

Estado del arte

4.1. Visión por computador y redes neuronales

Si bien la esencia de la computación yace en la habilidad de manejar y manipular datos, desde la simple suma de dos números al cálculo complejo de miles de filas en una base de datos, entonces podemos decir que la visión por computador no es más que esta ciencia aplicada a las imágenes, usando representaciones digitales de fotografías como dato de entrada. La visión por computador, también llamada visión artificial, es la disciplina que busca imitar la forma que las personas comprenden el mundo a través del sentido de la vista en una máquina, dotándola por medio de funciones y procesos, la capacidad de percibir y entender una o una secuencia de imágenes. De esta forma, una computadora sería capaz de extraer de una imagen o vídeo valiosos datos de la misma forma que lo hace una persona, e identificar que muestra la imagen, que objetos se encuentran en la imagen y cuantos hay, y en base a estos incluso tomar decisiones.

Hoy día, la tecnología de visión por computador es una de las más importantes y que con más rapidez crecen. Problemas técnicos como los de detección facial, detección de personas y objetos (mostrada en la Figura 4.1) o la reconstrucción en 3D a partir de imágenes están fundamentalmente solventados y podemos encontrar su aplicación en numerosos entornos tanto profesionales como personales.

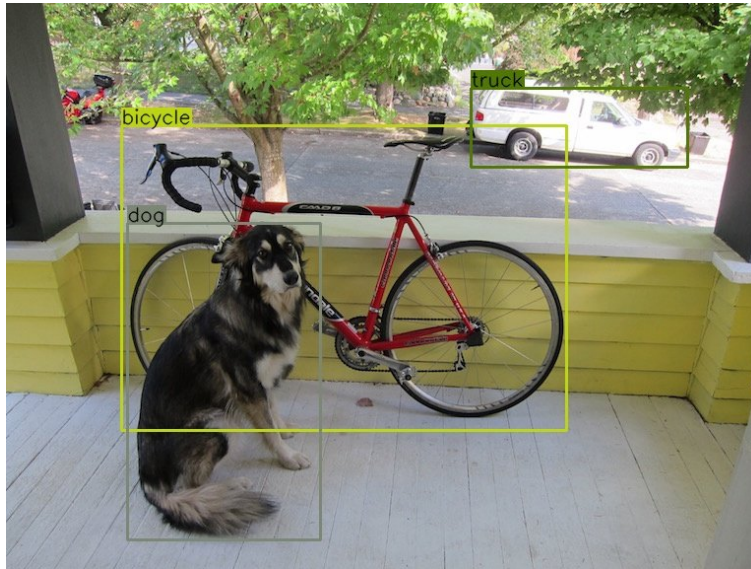


Figura 4.1: Ejemplo de detección de objetos por visión por computador (Imagen de YOLO [3])

Por otro lado, si la visión por computador se puede describir como la digitalización del ojo humano, entonces de la misma forma una red neuronal se puede definir como la sintetización del cerebro humano en una máquina. Una red neuronal artificial (abreviada en inglés ANN o NN - Artificial Neural Network o Neural Network) consiste en la abstracción de los cerebros animales biológicos a un sistema matemático que pueda ser fácilmente computable, y mediante la recogida de datos de entrada realice funciones que transformen dicha entrada en un conjunto de datos de salida que se traduzcan a la comprensión de los datos de entrada, realice algún tipo de interpretación o incluso toma de decisiones al respecto.

Todo este complejo proceso empieza en la abstracción de una neurona biológica a una función matemática, la cual constituye la unidad atómica, la más pequeña, de una red neuronal. Dichas neuronas reciben un conjunto de datos de entrada y realizan sobre este una función que genera un dato de salida. A la salida de la neurona existe una función de activación que seguidamente convierte los datos de salida en datos más fácilmente manejables, ya que una vez sale el dato de la neurona, este es enviado a través de sinapsis a una o varias neuronas, de la misma forma que funciona un cerebro biológico. Seguidamente, todas las neuronas de una red artificial están estructuradas en capas, entrando todos los datos de entrada que se quieran manejar por la capa de entrada, las capas intermedias llamadas capas ocultas, y el resultado del manejo de esos datos por la red neuronal por el extremo opuesto, la capa de salida. Esta estructura se puede ver simplificada en la Figura 4.2.

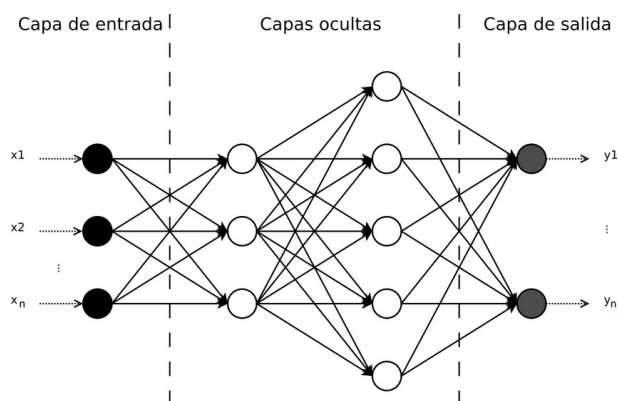


Figura 4.2: Diagrama de red neuronal artificial de cuatro capas (ResearchGate [21])

A medida que la red neuronal artificial procesa estos datos, las funciones que dictan cada neurona se cambian de forma que el resultado saliente se ajuste mejor al resultado que esperamos que resulte, cambiando así lo que llamamos su peso. Este proceso es denominado entrenamiento lo que permite que el modelo de red neuronal aprenda de cada conjunto de datos introducido. De esta forma, permitimos que la red neuronal aprenda patrones y realice tareas complejas, como el reconocimiento de imágenes o la predicción de resultados. Con redes más complejas y grandes, una red neuronal es capaz de imitar de manera simplificada el funcionamiento del cerebro humano para resolver problemas específicos de manera autónoma.

Realizando este proceso con una imagen como conjunto de datos de entrada, obtenemos una de las combinaciones más potentes de campos en el ámbito tecnológico y la razón por la que la visión por computador y las redes neuronales son unas de las disciplinas a la vanguardia del desarrollo tecnológico, pues somos capaces de entrenar redes neuronales con conjuntos de datos masivos para interpretar imágenes de tal forma que podamos detectar objetos, obtener descripciones o identificar personas. Este potencial es una de las motivaciones desde las que surge este proyecto, ya mencionada con anterioridad.

Con una comprensión básica del funcionamiento de la visión por computador y las redes neuronales, se proceden a explicar los ámbitos tecnológicos específicos que daremos uso en este proyecto.

4.2. Segmentado de imágenes

En la visión por computador la segmentación de imágenes consiste en el análisis de imágenes por un proceso automático para clasificar y dividir las distintas regiones de la imagen, de tal forma que asignamos un valor a cada píxel de la imagen catalogando así zonas de la imagen como pertenecientes a los distintos grupos en los que queremos dividirla. Se puede apreciar en la Figura 4.3 un ejemplo de este proceso. El valor de este proceso existe en la forma en la que clasifiquemos la imagen, ya que puede realizarse desde algo tan simple como categorizar la imagen por colores, señalando las distintas regiones similares en la imagen, como por segmentación semántica, en la que buscamos etiquetar las regiones de la imagen que pertenecen a la misma clase o categoría, tales como persona, coche o cielo. Este tipo de segmentación

otorga gran capacidad de comprensión a la aplicación sobre la que se ejecuta ya que adquiere una primera forma de discernir importantes valores que existen en la imagen aplicada tales como que objetos se pueden ver en la imagen, cuantos, que tamaño tienen o cuanto ocupan, entre algunos.



Figura 4.3: Segmentado de imagen semántico de una calle (SAM - Segment Anything Model / Meta [16])

Seguidamente, en el contexto de la segmentación de imágenes existe una herramienta fundamental conocida como máscara, que permite representar las áreas de interés de una imagen de forma binaria o multiclase, mostradas en la Figura 4.4. Las máscaras representan las distintas áreas segmentadas en una imagen, asignando a cada píxel de la imagen un valor, normalmente un 0 ó 1, pero pueden ser tantos como clases se encuentren, produciendo así una imagen en blanco y negro en el que las clases que hemos aislado y clasificado se muestran claramente sobre fondo negro. Esta herramienta es esencial tanto en la visualización del resultado de segmentación, como en el entrenamiento de modelos de aprendizaje automático, donde las máscaras actúan como el objetivo al que la red neuronal debe acercarse, actuando como las etiquetas que guían el aprendizaje. Es importante señalar que son extremadamente importante también en la evaluación del rendimiento de segmentación, al comparar las máscaras generadas por el modelo con las verdaderas, dándonos un seguimiento de la precisión del modelo. Las máscaras usadas durante este proceso de seguimiento que marcan la realidad a la que el modelo debe acercarse son llamadas 'ground truth'.

Además, es destacable que las máscaras son útiles en el postprocesado de imágenes, donde nos facilitan la extracción o análisis de objetos detectados.



Figura 4.4: Máscaras de segmentación semántica (ResearchGate [18])

4.2.1. Mask R-CNN

R-CNN (Del Inglés, Region-based Convolutional Neural Network - Red Neuronal Convolutiva basado en Regiones), es un tipo de arquitectura de aprendizaje profundo, desarrollada en 2013 por Ross Girshick y su equipo, utilizada en tareas de visión por computador para la detección de objetos. R-CNN fue pionero en el ámbito de los modelos de detección y ayudó a impulsar el campo tremendamente dando uso al enfoque basado en regiones del que gana su nombre y celebridad.

Originalmente, el primer modelo solo detectaba cajas contenedoras de los objetos, pero tras varias versiones y mejoras se llega a Mask R-CNN, la cual es capaz de detectar y generar máscaras de segmentación [5]. El modelo sigue la misma ideología basada en la múltiple iteración de búsqueda de regiones, haciendo tantas pasadas a la red neuronal como posibles objetos encuentre, llegando a los miles y ralentizando enormemente la búsqueda.

4.2.2. YOLO

YOLO (Del Inglés, You Only Look Once - Solo Se Mira Una Vez), es un sistema de detección de objetos en el campo de la visión por computador que revolucionó la tecnología al reformar la forma en la que se enfocaba la detección de objetos. YOLO unificó la detección de objetos en una sola red neuronal convolutiva, la cual requería una sola pasada de la imagen a través de la misma, reduciendo enormemente el tiempo de ejecución y recursos necesarios para la detección, y prediciendo simultáneamente clases y coordenadas de los objetos de la imagen detectados. Esta nueva arquitectura le permitió a YOLO alcanzar altas velocidades de procesamiento, y junto a una precisión que competía con el resto, lo convirtió en un enfoque ideal para aplicación de detección de objetos a tiempo real, en ámbitos como la vigilancia, robótica, o incluso conducción autónoma.

YOLO fue introducido por primera vez en 2015 por Joseph Redmon et al. [2], y hoy día llega a la versión número 12 con YOLO12, el cual cuenta con mejor soporte para más clases de objetos, mejor velocidad, e incluso segmentación plena en vez de detección de cajas contenedoras [27].

4.2.3. Segment Anything Model

Segment Anything Model [8] (En español - Modelo de Segmentar Cualquier Cosa), es un modelo para la segmentación desarrollado por Meta, con la principal característica de tener la capacidad de segmentar objetos en imágenes partiendo de un prompt, o estímulo, el cual puede ser uno o varios puntos de coordenada en la imagen, una caja contenedora, un área, o incluso texto. El modelo, usando ambos, la imagen y el prompt, producirá una máscara como salida conteniendo el objeto segmentado, mostrado en la Figura 4.5

Una de las mejores ventajas de este modelo es la enorme cantidad de datos de imágenes y segmentaciones que se ha usado para entrenarlo, más de mil millones de máscaras y once millones de imágenes con licencia de uso. Este entrenamiento le da a SAM la capacidad de generalizar su segmentación sobre entornos y contextos con los que no se ha entrenado, y producir resultados bastante favorables comparables a modelos entrenados con ese contexto en mente. Esta capacidad se denomina clasificación 'zero-shot', en la cual un modelo es capaz de extrapolar sobre datos de entrada con una naturaleza diferente a la que se ha entrenado.

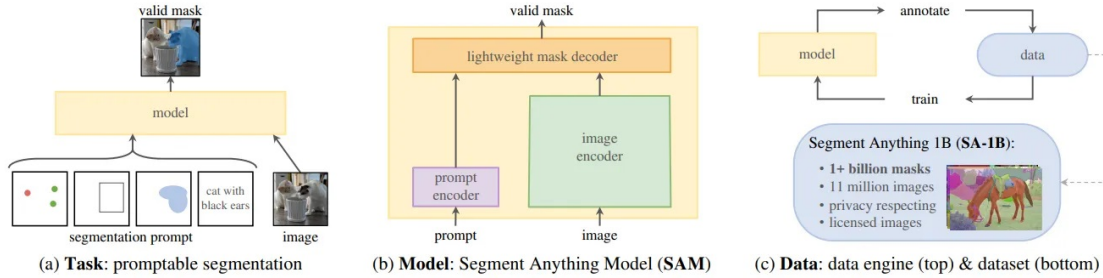


Figura 4.5: Segmentación de imágenes por prompt en SAM (Segment Anything Model [8])

El proceso de funcionamiento de SAM con el que podremos interactuar, mostrado gráficamente en la Figura 4.6, es el siguiente:

En primer lugar, la imagen es codificada para su inmediato uso como datos de entrada. Se utiliza un modelo Vision Transformer preentrenado, adaptado para el procesamiento de imágenes de alta calidad. El codificador ejecuta una vez por imagen, transformándola en el proceso en una imagen de entrada de 1024x1024.

Seguidamente, se codifica el prompt de entrada, en nuestro caso, solo nos interesa la entrada como forma de punto de coordenada, el cual será más útil para la detección de microplásticos individuales por toda la imagen.

Por último, el decodificador de máscaras transforma la entrada en una máscara binaria, escalada al tamaño original, mostrando la segmentación de objeto producida. Con una máscara como salida, el modelo produce una máscara media de salidas igualmente válidas si el prompt es suficientemente ambiguo, y para lidiar con esto el modelo produce varias máscaras de salidas las cuales se puntúan según la confianza del modelo, estimada con un cálculo de IoU (Intersection Over Union).

Este modelo se diseñó para alta eficiencia y ejecución en tiempo real. Con una imagen pre-procesada, el codificador de prompts y el decodificador de máscaras pueden ejecutarse en un navegador web, en CPU, en aproximadamente 50ms, habilitando la interacción casi a tiempo real.

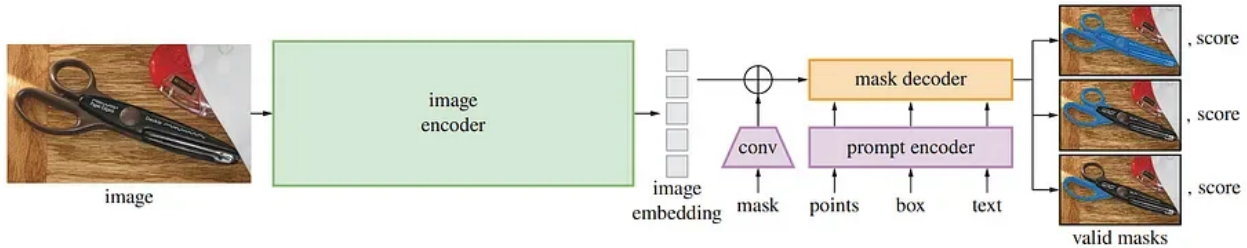


Figura 4.6: Proceso de Segment Anything Model (Segment Anything Model [8])

4.3. Modelos de lenguaje visual

Los Modelos de lenguaje visual (VLMs, por sus siglas en inglés - Visual Language Models) son modelos de inteligencia artificial, redes neuronales, que combinan conjuntos de datos de entrada tanto de texto como de imágenes o vídeo para comprender, interpretar y razonar sobre el contenido introducido. Estos modelos están diseñados para procesar y relacionar lenguaje natural con información visual, lo que les permite abordar tareas más complejas que involucren ambas modalidades.

Los modelos de lenguaje visual son aplicables en un amplio espectro de tareas, que solo crece con el tiempo, tales como: generación de texto a partir de imágenes, generando descripciones de imágenes, guía de búsqueda visual por texto, en el cual se ayuda a encontrar una imagen correspondiente a su descripción textual dada, o más importante y aplicable a este proyecto, preguntas y respuestas visuales, en el que el modelo es capaz de responder preguntas basadas en el contenido de la imagen.

4.3.1. BLIP

Dentro del campo de los modelos de lenguaje visual, podemos encontrar redes neuronales entrenadas para distintas tareas, tales como la clasificación de imágenes, búsqueda de imágenes a partir de un texto dado o descripción automática de imágenes. Pero con relevancia al proyecto, el modelo al que vamos a dar uso es uno entrenado para resolver tareas VQA (Del inglés - Visual Question Answering), o respuesta a preguntas visuales. En un sistema VQA, el modelo recibe una imagen junto con una pregunta en lenguaje natural y debe generar una respuesta precisa basada en la imagen de entrada, este proceso se puede ver en la gráfica de la Figura 4.7

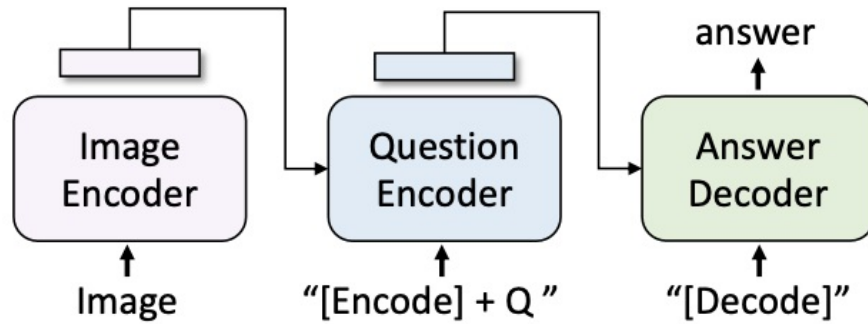


Figura 4.7: Funcionamiento de VQA (ResearchGate [9])

BLIP (Del Ingles Bootstrapping Language-Image Pre-training - Preentrenado de Lenguaje-Imágenes Autoimpulsado) es un framework de preentrenamiento de modelos de lenguaje visual que busca unificar las distintas tareas de lenguaje visual bajo una misma red neuronal. Usando un generador de subtítulos, crea subtítulos sintéticos y un filtro elimina los ruidosos, logrando no solo resultados novedosos y mejoras sobre la competencia, si no además obteniendo un model multimodal capaz de realizar tareas de recuperación de imágenes, generación de descripciones a partir de imágenes y respuestas a preguntas visuales [7].

Con un marco modular, el modelo de BLIP es además capaz de usar distintos modelos extensos de lenguaje como es el caso de OPT o Flan T5, distintos modelos de lenguaje que toman el proceso de generación de texto en el modelo de BLIP.

Capítulo 5

Herramientas y recursos

La selección de herramientas y recursos es un aspecto crítico y de los primeros a llevar a cabo durante el desarrollo de cualquier proyecto informático, ya que influye directamente en cada etapa de desarrollo y la producción final de la aplicación. Más aun, siendo este un proyecto de investigación en el que se busca explorar la viabilidad de los modelos de redes neuronales ya existentes, realizar una indagación sobre los recursos accesibles es un elemento crucial.

5.1. Entorno de desarrollo

El entorno de desarrollo es el conjunto de herramientas, recursos, configuraciones y tecnologías que los desarrolladores utilizan para crear, probar, y depurar software o aplicaciones. Engloba herramientas tales como lenguajes de programación o software de desarrollo.

Algunas de las herramientas más destacables del entorno de desarrollo escogido son las siguientes:

5.1.1. Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Fue creado por Guido van Rossum en 1991, diseñado para ser fácil de leer y escribir, ha ganado popularidad en los últimos años con el avance de redes neuronales, y sus características más apreciadas son su simplicidad, versatilidad, y clara y fácil sintaxis [22].

Python es el lenguaje preferido en el desarrollo de inteligencia artificial y aprendizaje profundo. Su favoritismo yace en sus extensas bibliotecas constantemente actualizadas y mantenidas, las cuales facilitan multitud de obstáculos normalmente presentes en el desarrollo de redes neuronales. Frameworks como TensorFlow, un marco de desarrollo de aprendizaje profundo desarrollado por Google que permite construir y entrenar redes neuronales de forma eficiente, simplifica la tarea de crear y entrenar redes neuronales de forma innata [26]. Y múltiples bibliotecas como NumPy, Pandas o Matplotlib ayudan al manejo de conjuntos de datos masivos muy presentes en el desarrollo de redes neuronales, optimizando y facilitando los cálculos de estos, y la visualización de resultados en gráficas.

5.1.2. Jupyter Notebook y Markdown

Jupyter Notebook es una herramienta de desarrollo ampliamente utilizada en el campo de la ciencia de datos y el ámbito de redes neuronales [19]. Su principal funcionamiento consiste en combinar bloques de código ejecutable, llamados celdas, sus salidas visualizadas, y texto en un solo archivo interactivo. La gran ventaja de usar Jupyter como herramienta de desarrollo radica en la habilidad de ejecutar celdas de código individualmente, lo que facilita la experimentación y depuración de código rápida y reflexivamente.

Además, Jupyter es ejecutable tanto en nativo como en servicio red, lo que otorga gran modularidad de entornos hardware a la hora de realizar pruebas de ejecución o entrenamiento de redes neuronales, dispensando una muy codiciada utilidad en el campo de la inteligencia artificial, la cual tiene elevados requerimientos de cómputo para su desarrollo. Como aparte, esta cualidad también hace de Jupyter Notebooks una herramienta que facilita el intercambio de código y documentos en este campo.

Dentro de Jupyter, Markdown desempeña un pequeño papel al permitir la fácil documentación de las celdas de código, permitiendo una estructuración clara y sencilla. Es por esto por lo que estas dos herramientas son muy comúnmente usadas en el desarrollo de inteligencia artificial, y lo convierten en una clara elección para el desarrollo de este proyecto. Se puede comprobar en la Figura 5.1 esta característica.



Figura 5.1: Ejemplo de celdas de código en Jupyter Markdown (Google Colab [13])

5.1.3. Pytorch

PyTorch es una biblioteca optimizada de tensores para el aprendizaje profundo de código abierto desarrollada por Meta AI [23]. Su diseño facilita enormemente la construcción y el entrenamiento de redes neuronales mediante una interfaz flexible e intuitiva, integrando funcionalidades como la eficiente operación de tensores que simplifica el trabajo con redes neuronales. Además, permite su ejecución tanto en GPU como CPU, dando uso del hardware que acelera enormemente el entrenamiento o en su ausencia el soporte físico por defecto. Debido a estas características y más, Pytorch se ha consolidado como una herramienta fun-

damental en la investigación y en la industria, convirtiéndola en un añadido necesario al TFG que se usará tanto para dar uso a los modelos ya existentes, como el entrenamiento y alteración de las redes neuronales visitadas en el proyecto.

5.1.4. LAVIS

LAVIS, (Del inglés, Language and Vision - Lenguaje y Vision), es una biblioteca orientada a Python para la investigación y desarrollo de aplicaciones de aprendizaje profundo [20]. Tiene como objetivo proporcionar a ingenieros informáticos e investigadores una plataforma flexible y eficiente para desarrollar rápidamente modelos que cumplan escenarios multimodales propuestos, y evaluarlos con conjuntos de datos tanto estándar como personalizados.

Algunas de las características que hacen de LAVIS un gran refuerzo en el desarrollo de redes neuronales son las herramientas disponibles tales como una interfaz unificada y modular que da acceso a acciones relevantes a la inteligencia artificial como etiquetado de imágenes con todo tipo de estándares, clasificación multimodal, acceso a conjuntos de datos y más importantemente, modelos ya entrenados y adaptados a tareas específicas de visión y lenguaje. En el contexto del TFG, se usó LAVIS para el desarrollo del modelo de lenguaje visual encargado de clasificar las detecciones.

5.1.5. VSCode

VSCode (Por su nombre en inglés - Visual Studio Code) es un entorno de desarrollo integrado multiplataforma desarrollado por Microsoft. Es ampliamente utilizado debido a su flexibilidad y soporte para múltiples lenguajes de programación, uno de los cuales es Python. También integra la posibilidad de ejecutar código de documentos Jupyter Notebook nativamente, lo que lo convierte en uno de los entornos favoritos en el campo de la inteligencia artificial. Incluye múltiples características favorables como depuración integrada, control de versiones con Git y una amplia variedad de extensiones descargables que permiten personalizar su funcionalidad para asemejarse a la tarea de desarrollo escogida [28].

5.1.6. Especificaciones del equipo

Durante el desarrollo de el TFG se usó el siguiente equipo personal:

Sistema Operativo	Windows 10
CPU	Intel Core i7 10700F @ 2.90GHz
GPU	8176MB ATI AMD Radeon RX 6600
Memoria RAM	32,0 GB DDR4

Tabla 5.1. Especificaciones de equipo

Es importante recalcar que la GPU mostrada en la Tabla 5.1 siendo de AMD, no tiene compatibilidad con CUDA y por lo tanto no puede ser usado para el uso y entrenamiento de

redes neuronales. Por lo tanto, se dará uso a la CPU para esta tarea, y los tiempos de ejecución tomados durante el desarrollo de el TFG serán enormemente ralentizados al respecto.

5.2. Modelos

Una parte crucial del desarrollo de este proyecto es la selección de modelos de red neuronales a usar, que representaran las distintas funcionalidades descritas en los objetivos. Tras realizar una investigación y lectura sobre los modelos existentes, se ha escogido el modelo SAM debido al hecho que requiere entrenamiento casi nulo, y siendo además un modelo novedoso que hoy día sigue recibiendo actualizaciones y mejoras, se ha decidido centrar la tarea de investigación y objetivo de segmentación de los microplásticos en torno a este modelo. En el caso de clasificación, la novedad del modelo BLIP de LAVIS es una de las principales razones por las que se ha escogido para el objetivo de clasificación de los microplásticos ya segmentados.

5.3. Bibliotecas importantes

Durante el desarrollo del proyecto se usaron diversas bibliotecas, pero las que tienen especial importancia fueron:

5.3.1. Supervision

La biblioteca Supervision desarrollada por Roboflow, es una herramienta diseñada para el posprocesamiento de resultados en el campo de visión por computador en tareas como la detección de objetos, segmentación y seguimiento en vídeo. Posee varias funciones que permiten la manipulación y fácil edición de cuadros delimitadores, máscaras y anotaciones en imágenes o videos de entrada.

Siendo una biblioteca diseñada para cualquier modelo, es extremadamente útil en el manejo de resultados y permite un rápido análisis visual de la precisión de la aplicación y por lo tanto un perfecto apoyo al proyecto [25].

5.3.2. Webcolors

La biblioteca de Python Webcolors es una simple herramienta que facilita el uso de trabajo con colores, y sus diferentes formatos encontrados. Durante el desarrollo de la aplicación se vio la necesidad de traducir entre estándares, y esta ligera biblioteca ofrece todos los requisitos al respecto.

5.3.3. QTDesigner

Basado en el framework multiplataforma Qt, Qt Designer es una herramienta gráfica para el diseño de interfaces de usuario en aplicaciones, siendo independiente de plataformas y

lenguajes de programación, es un entorno increíblemente útil para el desarrollo de aplicaciones en Python, que por defecto no tiene biblioteca que apoyan el desarrollo de interfaces gráficas [24].

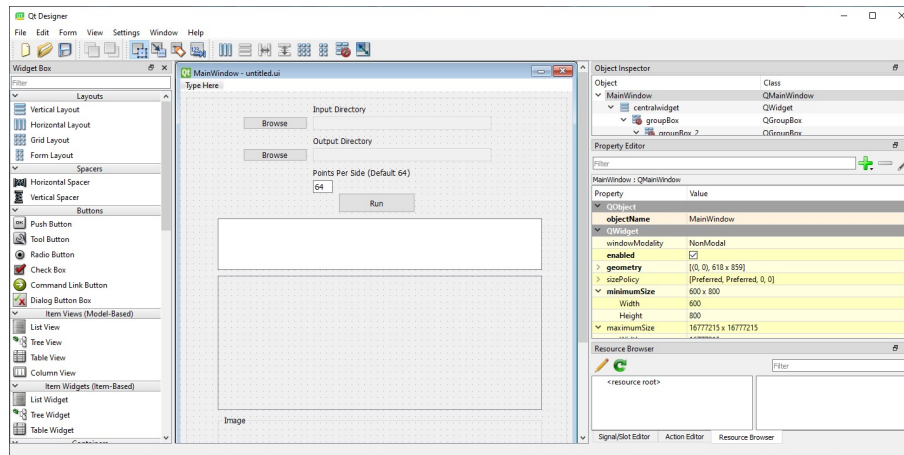


Figura 5.2: Desarrollo en QtDesigner

Durante el desarrollo de el TFG se vio la necesidad de desarrollar una interfaz gráfica para la aplicación para el uso de usuarios medios. QtDesigner fue la herramienta seleccionada debido a su fácil y rápido aprendizaje, permitiendo la creación de ventanas y elementos interactivos por medio de una interfaz gráfica intuitiva. Además, la integración con PyQt, una biblioteca adicional para Qt, permite la conexión de la interfaz con la lógica de la aplicación, asegurando un flujo de trabajo eficiente y modular, donde cambios al elemento lógico se veían reflejados instantáneamente en la interfaz gráfica. En la Figura 5.2 se puede ver un ejemplo del uso de este framework durante el desarrollo del TFG.

Capítulo 6

Desarrollo

6.1. Estudio previo

Como preámbulo al desarrollo de la aplicación, se realizó una instalación y comprobación de los modelos a usar, y comprobar su funcionamiento. Esto incluyó la descarga de los archivos requeridos, la configuración del entorno de trabajo y la instalación de todas las dependencias requeridas. Posteriormente, tras la comprobación del entorno, se realizaron pruebas iniciales para adquirir una primera valoración del funcionamiento y eficacia de los modelos a usar en el TFG.

6.1.1. Familiarización con SAM

Para empezar con el TFG, primero es de vital importancia familiarizarse con las herramientas que vamos a utilizar. Por lo tanto, se empezó este preámbulo realizando una breve lectura y comprensión del estudio del modelo SAM, descrito anteriormente en esta memoria.

Una vez adquirida una primera visión del modelo, se procedió a la instalación descrita en el GitHub del mismo, en la que brevemente se describen las versiones de las dependencias requeridas para el propio uso del entorno. Además, se realiza una comprensión del programa demostrador de la implementación de segmentación en el cual se apoya el modelo, que será la base de nuestro objetivo de segmentación de microplásticos. Es importante destacar que en el manual de usuario del programa SAM existen 3 distintos modelos de segmentación base, 'B', 'L' y 'H', en función del tamaño del modelo - Base, Large (Grande) y Huge (Enorme), siendo la diferencia entre ellos una ligera diferencia de rendimiento donde la más grande tiene la mejor precisión, pero mayor tamaño y por lo tanto mayor tiempo de ejecución, y donde la más pequeña tiene menor precisión, pero menor tamaño y tiempo de ejecución. Por defecto, el programa usa el modelo 'H'.

Usando el estándar de Jupyter Notebook descrito anteriormente, y conociendo el funcionamiento del programa SAM, se realiza una pequeña demo de segmentación con una imagen de microplásticos a modo de prueba, ofreciendo el siguiente resultado:

```
import cv2
from PIL import Image
from segment_anything import SamAutomaticMaskGenerator, sam_model_registry

model_name = "default"
model_path = "C:/Users/Alberto/Desktop/Models/sam_vit_h_4b8939.pth"
image_path = "C:/Users/Alberto/Desktop/Data Set/ejemplo_segmentacion.png"

raw_image = cv2.imread(image_path)
pil_image = cv2.cvtColor(raw_image, cv2.COLOR_BGR2RGB)
im_pil = Image.fromarray(pil_image)
display(im_pil)

sam = sam_model_registry[model_name](checkpoint=model_path)
mask_generator = SamAutomaticMaskGenerator(sam)
masks = mask_generator.generate(raw_image)

✓ 1m 22.6s
```

Figura 6.1: Primera celda de ejecución SAM

```
import cv2
from PIL import Image
from segment_anything import SamAutomaticMaskGenerator, sam_model_registry

model_name = 'default'
model_path = 'X'
image_path = 'X'

raw_image = cv2.imread(image_path)
pil_image = cv2.cvtColor(raw_image, cv2.COLOR_BGR2RGB)
im_pil = Image.fromarray(pil_image)
display(im_pil)

sam = sam_model_registry[model_name](checkpoint=model_path)
mask_generator = SamAutomaticMaskGenerator(sam)
masks = mask_generator.generate(raw_image)
```

La celda de código mostrada en la Figura 6.1 extrae una imagen de microplásticos, la muestra en pantalla, y la usa como entrada de datos al modelo SAM. Como podemos comprobar la celda finaliza y ejecuta en 1 minuto 22 segundos, el modelo por defecto, y el más grande, 'H'. El resultado de la ejecución se guarda en la variable 'masks', la cual contiene todas las máscaras de segmentación detectadas por el modelo.

```
import supervision as sv

mask_annotator = sv.MaskAnnotator(color_lookup = sv.ColorLookup.INDEX)
detections = sv.Detections.from_sam(masks)
annotated_image = mask_annotator.annotate(raw_image.copy(), detections)
annotated_pil = Image.fromarray(annotated_image)
display(annotated_pil)

masks_array = []
for mask in masks:
    masks_array.append(mask['segmentation'])

mask_pil = Image.fromarray(masks_array[2])
display(mask_pil)

sv.plot_images_grid(
    images=masks_array,
    grid_size=(8, int(len(masks) / 8)+1),
    #size=(16, 16)
)
```

En la siguiente celda se ejecuta el siguiente fragmento de código, el cual usamos para visualizar los datos de salida del modelo. Se puede ver el uso de la biblioteca Supervision, el cual ayuda a la presentación de datos, junto con la manipulación de la variable masks, la cual es un objeto con estructura interna, para la extracción de las máscaras binarias. El resultado se muestra en la Figura 6.2.

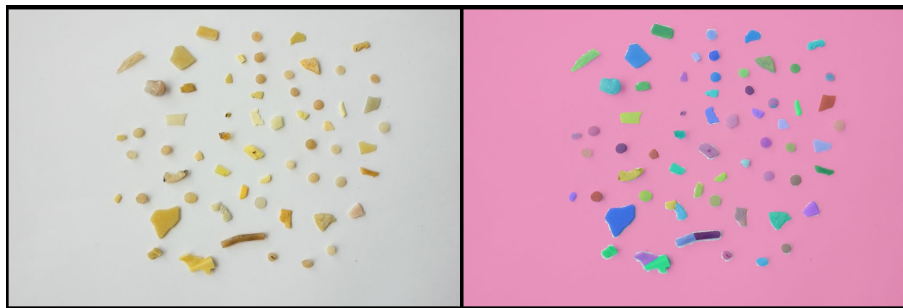


Figura 6.2: Imagen demo usada y las máscaras extraídas superpuestas

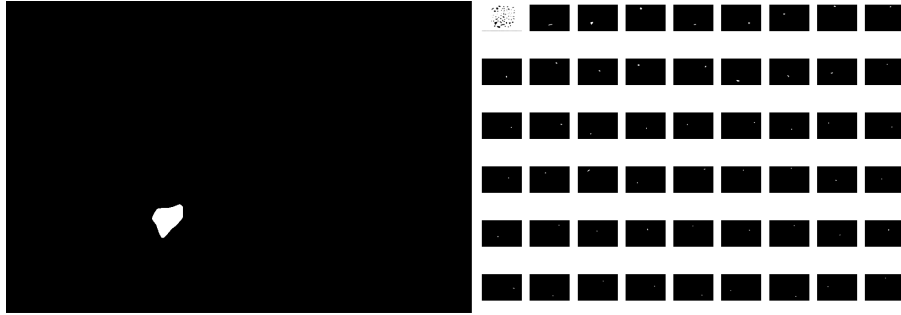


Figura 6.3: Máscara binaria extraída, y matriz con el resto de máscaras

Podemos observar en la Figura 6.2 la imagen demo original a la izquierda, y a la derecha, las áreas de segmentación detectadas superpuestas con distintos colores. Se puede apreciar a primera vista como ha detectado el fondo de papel como propia área de segmentación, y como además tiene un gran grado de acierto respecto a los microplásticos segmentados detectando prácticamente todos.

En la segunda imagen en la Figura 6.3, se muestra una única máscara binaria de uno de los microplásticos segmentados, junto a una matriz de máscaras binarias extraídas del conjunto de salida, como muestra de funcionamiento del modelo.

Con una correcta ejecución del modelo, y uso de los datos de salida, finaliza aquí la familiarización con el modelo de segmentación a usar.

6.1.2. Familiarización con LAVIS

Reflejando la primera parte, realizamos el mismo procedimiento de comprensión e instalación de LAVIS, y prueba del modelo BLIP para respuesta a preguntas visuales.



Figura 6.4: Estatua de merlion en Singapur (Wikipedia [29])

```
import torch
from lavis.models import load_model_and_preprocess
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model, vis_processors, txt_processors = load_model_and_preprocess(name="blip_vqa", model_type="vqav2", is_eval=True, device=device)
✓ 53s

raw_image = Image.open("C:/Users/Alberto/Downloads/merlion.png").convert("RGB")

# ask a random question.
question = "Which city is this photo taken?"
image = vis_processors["eval"](raw_image).unsqueeze(0).to(device)
question = txt_processors["eval"](question)
model.predict_answers(samples={"image": image, "text_input": question}, inference_method="generate")

# ['singapore']
✓ 1.0s

['singapore']
```

Figura 6.5: Celda de código de ejecución del modelo

```
import torch
from lavis.models import load_model_and_preprocess
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model, vis_processors, txt_processors = load_model_and_preprocess(name=
    'blip_vqa', model_type='vqav2', is_eval=True, device=device)

raw_image = Image.open('X').convert('RGB')
# realizamos una pregunta aleatoria ('¿Dónde se sacó la foto?')
question = 'Which city is this photo taken?'
image = vis_processors['eval'](raw_image).unsqueeze(0).to(device)
question = txt_processors['eval'](question)
model.predict_answers(samples={'image': image, 'text_input': question},
    inference_method='generate')

# ['singapore']
```

En la Figura 6.4 se muestra la imagen demo a usar, una foto de una estatua de un Merloin, en Singapur. Y seguidamente en la Figura 6.5 se muestra la celda de código donde se ejecuta el modelo, y se realiza la pregunta '¿Dónde se realizó la foto?' en inglés. En la figura podemos apreciar inmediatamente dos distinciones, la primera, que ha respondido correctamente a la pregunta, siendo la salida 'Singapore', la ciudad perteneciente a la estatua, y segundo, que el modelo se ha ejecutado en un solo segundo.

Con un correcto funcionamiento de ambos modelos, SAM y LAVIS BLIP, y un correcto manejo de los datos de salida, termina el proceso de prueba y aprendizaje del proyecto.

6.1.3. Conjuntos de datos

Un elemento importante en el desarrollo de redes neuronales son los conjuntos de datos, ya que la calidad y cantidad de datos de entrada que tengan estos impactan en la capacidad del modelo de ajustarse a la tarea en cuestión, o capacidad de generalización. En este proyecto, los modelos usados ya vienen entrenados, y por lo tanto los conjuntos de datos se usarán para la validación de los resultados de la aplicación, una etapa que será fundamental para evaluar el desempeño y capacidad de segmentación y clasificación de la aplicación en todas las etapas de esta. Es por ello, que, para tener una validación robusta, es necesario tener un conjunto de datos amplio que abarque tantos escenarios de segmentación de microplásticos como sea posible.

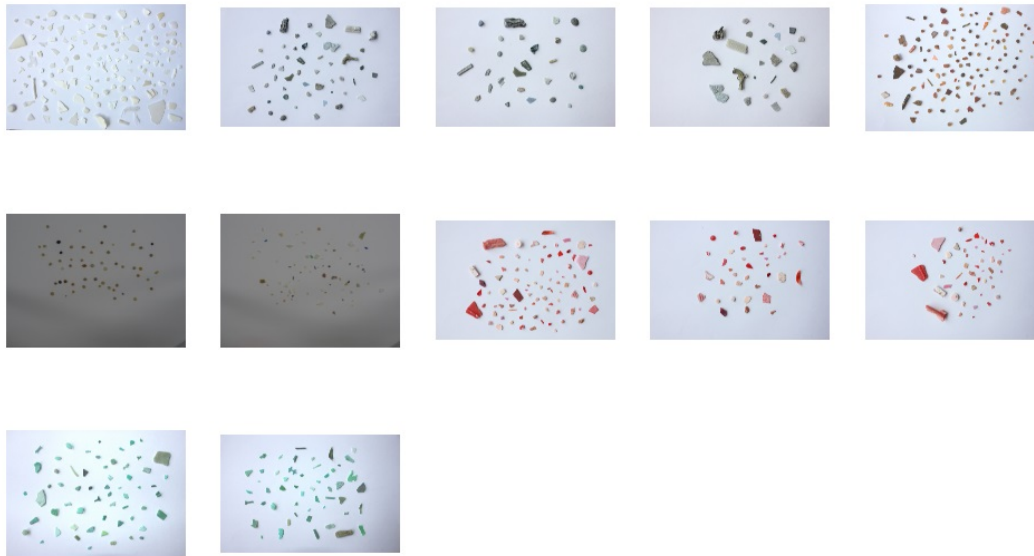


Figura 6.6: Conjunto de Datos proporcionado

El siguiente conjunto de imágenes mostrados en la Figura 6.6, proporcionadas por los tutoriales, son las muestras usadas en el recuento y clasificación de microplásticos para su introducción en bases de datos para uso científico, las que son parte del proceso de cuenta que este proyecto busca automatizar. Luego es el conjunto de datos de prueba que más se ajusta al entorno de trabajo real que buscamos. Se pueden observar diferentes escenarios que nos serán útiles tales como de iluminación y colores de microplásticos.

Parte del trabajo de validación consiste en establecer un 'ground truth', un etiquetado manual con la que comparar los resultados del programa, y validar su cercanía y precisión a los objetivos establecidos. En este caso, consiste en el recuento manual de los microplásticos en cada imagen y su clasificación, para poder obtener un valor real con el que comparar la salida de datos. Este proceso ayudara a valorar la evolución de la aplicación, identificando y cuantificando las mejoras o deterioros en el proceso de segmentación y clasificación de microplásticos en las distintas versiones de la aplicación.

6.2. Pipeline básico

Una vez familiarizados con los modelos y herramientas que componen el proyecto, se procede a desarrollar una primera fase de la aplicación, el cual compondrá un pipeline básico entre ambos modelos, en el cual la salida del segmentador conectará con la entrada del clasificador, obteniendo como resultado final un recuento de los microplásticos por colores. Un diagrama de tal modelo se muestra en la Figura 6.7.

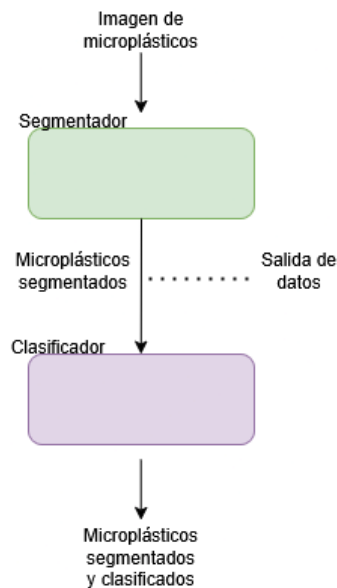


Figura 6.7: Diagrama del pipeline básico de la aplicación

A continuación se muestra una sección del código que enlaza la salida de datos de un modelo, con la entrada del siguiente, mostrado en la Figura 6.8.

```
array_of_plastics_crop = []
for i in range(0, len(masks_array)):
    image_to_mask = np.array(raw_image)

    mask = masks_array[i]
    bbox = masks[i]['bbox']
    mask_to_np = np.array(mask)
    image_to_mask[~mask,:] = [255,255,255]
    #bbox is XYWH, cropping is y:y+h , x:x+w
    cropped_image_final = image_to_mask[bbox[1]:bbox[1]+bbox[3] , bbox[0]:bbox[0]+bbox[2]]
    array_of_plastics_crop.append(cropped_image_final)
```

Figura 6.8: Celda de código de recorte de microplásticos

```
array_of_plastics_crop = []
for i in range(0, len(masks_array)):
    image_to_mask = np.array(raw_image)

    mask = masks_array[i]
```

```

bbox = masks[i]['bbox']
mask_to_np = np.array(mask)
image_to_mask[~mask,:] = [255,255,255]
#bbox is XYWH, cropping is y:y+h , x:x+w
cropped_image_final = image_to_mask[bbox[1]:bbox[1]+bbox[3]
    , bbox[0]:bbox[0]+bbox[2]]

array_of_plastics_crop.append(cropped_image_final)

sv.plot_images_grid(
    images=array_of_plastics_crop,
    grid_size=(8, int(len(array_of_plastics_crop) / 8)+1),
)

```

Usando los parámetros proporcionados por el objeto salida de la función de segmentado, realizamos recortes en la imagen original de entrada, mostrado en la celda de código de la Figura 6.8, obteniendo de esta forma una lista de imágenes de todos los microplásticos detectados y segmentados por el modelo, el cual posteriormente serán enviados al segundo modelo de la aplicación, en el cual la pregunta '¿De qué color es la piedra de la imagen?' ha sido establecida como parámetro de entrada, y las respuestas guardadas en una lista para su análisis. Se puede observar en la Figura 6.9, la lista de imágenes de los microplásticos detectados.

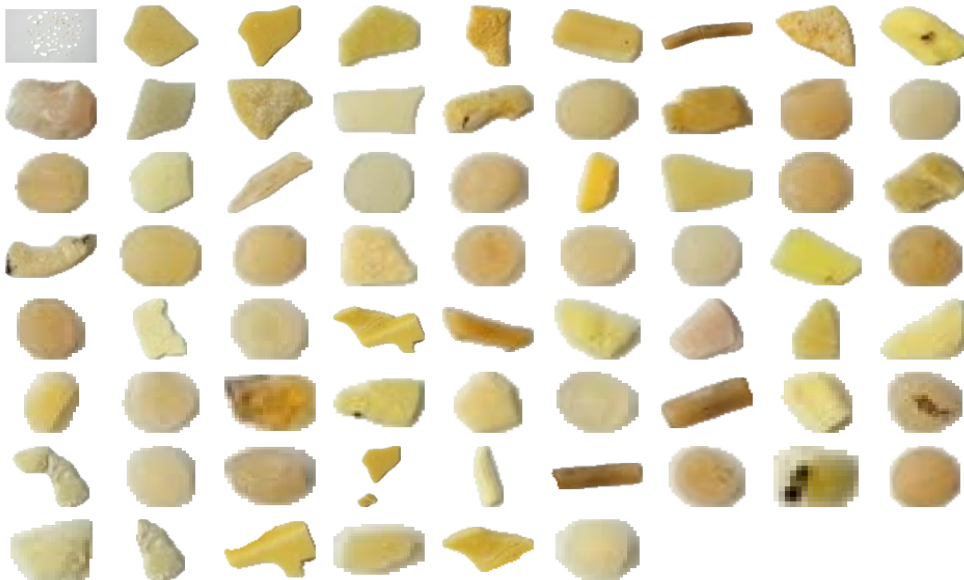


Figura 6.9: Imágenes de microplásticos de prueba

6.2.1. Resultados

Para esta primera fase, se han usado solo 3 imágenes a modo de valoración inicial, las 3 escogidas de tal forma que representan distintos posibles escenarios en un entorno de clasificación real, tanto de color, como de iluminación, cantidad de microplásticos presentes, y tamaño de imagen.

Los resultados a medir serán:

- Tiempo de ejecución: El período que tarda el modelo en ejecutarse desde el inicio hasta su finalización, utilizando el modelo a valorar en cada sección.
- Verdaderos positivos: Resultado en el que un modelo predice correctamente la característica a buscar. (Segmenta correctamente un microplástico)
- Falsos positivos: Resultado en el que un modelo predice incorrectamente la presencia de una característica a buscar. (Segmenta un área que carece de microplásticos)
- Falsos negativos: Resultado en el que un modelo no predice correctamente la presencia de una característica a buscar. (No segmenta un microplástico)
- Porcentaje de aciertos: Número de positivos verdaderos respecto al valor real.

Finalmente, los resultados se medirán individualmente por cada modelo, para poder obtener una valoración individual de cada segmento de la aplicación, y concluir que zonas del proceso de segmentación y clasificación requieren más tiempo de desarrollo. Cabe mencionar que el tiempo de ejecución medido solo tendrá en cuenta la ejecución del modelo, ya que compone la mayor parte de la ejecución, siendo el resto insustancial en comparación.



Figura 6.10: Microplásticos segmentados en primera fase



Figura 6.11: Primer resultado de segmentación de microplásticos

	Segmentación	Clasificación
Tiempo de ejecución	1m 39s	1m 0s
Verdaderos positivos	52	68
Falsos positivos	4	3
Falsos negativos	9	0
Porcentaje de acierto	85 %	97 %

Tabla 6.1. Resultados de segmentación de primera imagen

	Segmentación	Clasificación
Tiempo de ejecución	3m 6s	1m 47s
Verdaderos positivos	130	154
Falsos positivos	20	1
Falsos negativos	8	0
Porcentaje de acierto	94 %	100 %

Tabla 6.2. Resultados de segmentación de segunda imagen

	Segmentación	Clasificación
Tiempo de ejecución	1m 15s	30s
Verdaderos positivos	34	18
Falsos positivos	2	16
Falsos negativos	21	0
Porcentaje de acierto	61 %	55 %

Tabla 6.3. Resultados de segmentación de tercera imagen

Nótese el hecho que la sección de clasificación siempre da cero negativos falsos, esto es debido a que el resultado siempre o es una clasificación correcta del color, o no lo es, independientemente de si el microplástico fue un falso positivo segmentado.

En la Figura 6.10, se muestran las 3 imágenes usadas como validación, y en la Figura 6.11, los resultados de la segmentación en mismo orden, en donde se han postprocesado las imágenes originales, pintando de varios colores las distintas máscaras binarias superpuestas sobre los microplásticos e incluso el fondo. Seguidamente, en las Tablas 6.1, 6.2 y 6.3, se muestran las mediciones sobre estas tres segmentaciones, una vez mas en la misma secuencia.

De estos primeros resultados podemos obtener varias conclusiones. En primer lugar, con ayuda de las imágenes como guía de visualización, se puede comprobar que el fondo de papel siempre es segmentado y detectado por el modelo, detectándolo como falso positivo. Seguidamente, se puede comprobar una casi perfecta fiabilidad del clasificador por color en imágenes de iluminación correcta, y una casi igual alta fiabilidad para detectar todos los microplásticos del segmentador, en condiciones similares. Sin embargo, en un entorno de iluminación pobre, decrece significativamente.

Se puede además comprobar tanto en las imágenes segmentadas como en las tablas, la relativa gran cantidad de falsos positivos debido a que se realiza la segmentación sobre un mismo microplástico varias veces, detectándolo como distintos.

Finalmente, se puede apreciar el corto periodo de ejecución de los modelos, y como el primero tarda significativamente más que el resto debido al tamaño de la imagen y como el tiempo de clasificación de microplásticos viene lógicamente dada por la cantidad de microplásticos detectados.

Concluimos con esto la primera fase del proyecto, y con una visión de la aplicación al respecto. Problemas como la pobre iluminación de algunas imágenes, la gran cantidad de falsos positivos, y el aumento del porcentaje de acierto al máximo posible en la sección de segmentación serán los problemas por enfocar durante el resto del desarrollo.

6.3. Migración a diseño OOP

OOP (Del inglés - Object-Oriented Programming) o Programación Orientada a Objetos es un modelo de programación basado en la organización del código en objetos, que representan entidades del mundo real con propiedades (atributos) y comportamientos (métodos). A diferencia del modelo de programación procedimental usada hasta ahora en Jupyter Notebook, la programación orientada a objetos permite el uso de clases, lo que nos permitirá mejorar la organización y escalabilidad del programa.

En esta sección del desarrollo del proyecto, se realizará una migración del programa a una versión con programación orientada a objetos, y una vez terminada, se irán añadiendo los distintos módulos al algoritmo de segmentado y clasificación que busquen solventar los problemas. En la Figura 6.12 podemos apreciar un diagrama de la estructura a desarrollar.

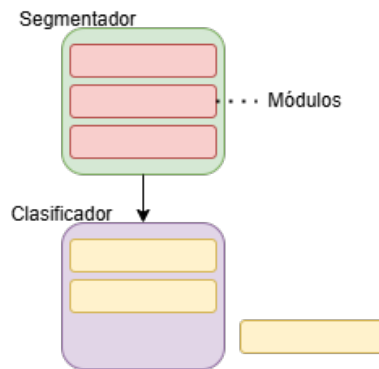


Figura 6.12: Diagrama del modelo OOP de la aplicación

Esta versión del programa está compuesta por dos principales secciones, la primera, el segmentador, contiene el modelo SAM de segmentación de microplásticos, y la segunda, el clasificador, contiene el modelo BLIP de respuestas a preguntas visuales. La salida del segmentador será usada como entrada para el clasificador, y una vez terminada la segmentación y clasificación de la imagen, se guardarán los resultados en un fichero de salida aparte. Dentro de cada sección del programa, se hallarán módulos que realicen cambios al flujo de datos, permutando el algoritmo de cada sección para su mejora. Usando las bases de la programación orientada a objetos, estos módulos pueden ser fácilmente sustituibles o accionables, para un control total del flujo del programa.

Los resultados finales del programa contendrán datos útiles tanto para los objetivos a cumplir del TFG como para el análisis y validación del programa. Dentro del fichero podremos encontrar:

- Número de microplásticos detectados, clasificados por colores
- Áreas en píxeles de los microplásticos detectados, clasificados por colores
- Tiempos de ejecución de las distintas secciones
- Archivo CSV de los datos para su fácil manipulación y uso
- Carpeta con imágenes de los distintos microplásticos detectados, y distintas imágenes editadas mostrando las detecciones

Finalmente, también se ha añadido la posibilidad de segmentar tanto imágenes individuales como ficheros enteros para el uso en volumen del programa.

6.4. Módulos implementados

Terminada la migración a un modelo de programación orientada a objetos, se proceden a describir los distintos módulos implementados en el flujo del programa.

6.4.1. Balance de blanco

El balance de blancos es un ajuste en fotografía y procesamiento de imágenes que busca corregir la gama de colores para que estos se vean más naturales y fieles a la realidad. Su principal objetivo es compensar las distintas iluminaciones que una escena pueda tener, y evitar así colores que dominen de fondo como tonos anaranjados o azules. Es una función presente en muchas cámaras hoy día, sin embargo, no siempre está habilitado.

En el contexto del proyecto como ya se vio con anterioridad, la pobre iluminación de la escena causa un gran deterioro en la calidad de segmentación y clasificación. Y aun teniendo una buena iluminación, si el balance de blanco no es el correcto y hay presentes colores alternos de fondo, es posible recibir una correcta segmentación, pero una muy pobre clasificación debido al cambio de los colores de los microplásticos por este ruido de fondo de iluminación. Es por esto, que se ha decidido añadir un módulo de balance de blanco que busque corregir estos posibles errores para mejorar la calidad de los objetivos del proyecto.

Para ello, se han implementado tres distintos algoritmos de balance de blanco ya existentes, desglosados y descritos a continuación:

```
class WhiteBalancer:

    def gray_world(self, image):
        wb = cv2.xphoto.createGrayworldWB()
        wb.setSaturationThreshold(self.wb_value)
        image = wb.balanceWhite(image, image)
        return image

    def gray_world_2(self, image):
        gray_world_image = skimage.util.img_as_ubyte(
            (image * (image.mean() / image.mean(axis=(0, 1))))
            .clip(0, 255).astype(int))
        return gray_world_image

    def white_patch(self, image, percentile=100):
        white_patch_image = skimage.util.img_as_ubyte(
            (image*1.0 / np.percentile(image,percentile, axis=(0, 1)))
            .clip(0, 1))
        return white_patch_image
```

El primer algoritmo es el algoritmo de mundo gris (Del inglés - Gray World Algorithm), el cual postula que, en promedio, los colores en una escena natural deben equilibrarse hasta formar un tono de gris neutro. Esto significa que la media de todos los valores de los canales rojo (R), verde (G) y azul (B) de una imagen deben ser aproximadamente igual. Este algoritmo viene implementado por defecto con la biblioteca cv2 de Python, y por lo tanto no hace falta gran desarrollo. Los resultados de aplicar este algoritmo a algunas de nuestras imágenes de muestra se encuentran en las Figuras 6.13, 6.14 y 6.15.



Figura 6.13: Ejemplo 1 del uso del algoritmo mundo gris

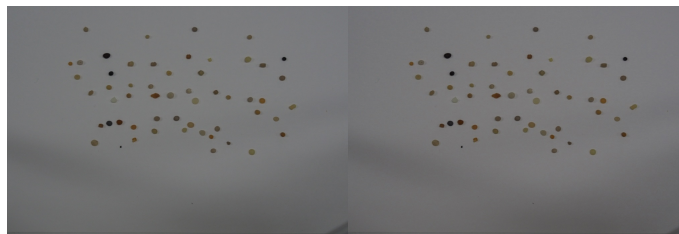


Figura 6.14: Ejemplo 2 del uso del algoritmo mundo gris

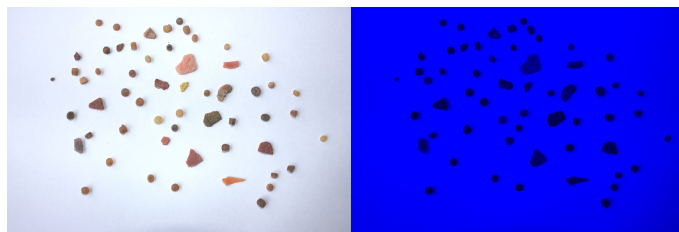


Figura 6.15: Ejemplo 3 del uso del algoritmo mundo gris

Como se puede observar a primera vista en la Figura 6.13, se eliminan los matices de amarillo presentes de fondo, y se aclara la imagen, mejorando la clasificación por colores de los microplásticos. Sin embargo, en la Figura 6.14, se observa que no hay cambios, debido a que, siendo la imagen principalmente ya gris, no hay balance en las capas necesario. Y finalmente, en la Figura 6.15, el balance de blanco de esta implementación falla por algún motivo desconocido, saturando la capa de azul de la imagen, empeorando significativamente el resultado, haciendo mucho más difícil de segmentar e imposible de clasificar.

El segundo algoritmo es otra implementación del algoritmo del mundo gris, programado manualmente. Matemáticamente, es supuestamente el mismo algoritmo, descrito ya con anterioridad, pero esta vez implementado usando la biblioteca skimage, y realizando el cálculo de media y transformación de la imagen en el mismo código. Los resultados se muestran en las Figuras 6.16, 6.17 y 6.18.

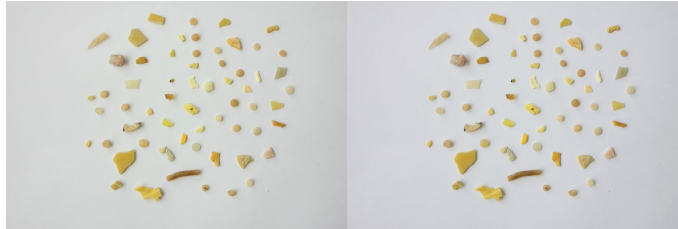


Figura 6.16: Ejemplo 1 del uso del algoritmo mundo gris 2



Figura 6.17: Ejemplo 2 del uso del algoritmo mundo gris 2

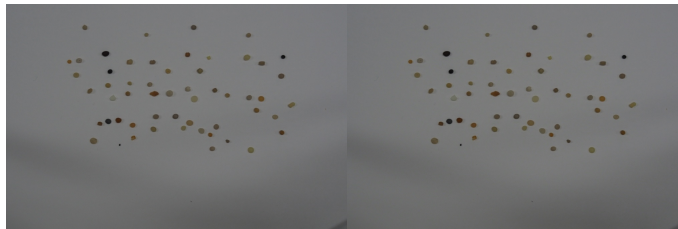


Figura 6.18: Ejemplo 3 del uso del algoritmo mundo gris 2

Como se puede comprobar a simple vista, se reproducen los resultados de la primera implementación en las Figuras 6.16 y 6.18, pero esta vez se elimina la imprevista saturación de capas en la Figura 6.17. Sin embargo, los resultados de la Figura 6.18 siguen siendo nulos, y siendo la principal razón del desarrollo de este módulo la eliminación de la pobre iluminación de las imágenes, se considera un resultado negativo.

El tercer algoritmo es el algoritmo de parche blanco (Del inglés - White Patch Algorithm) y se basa en la suposición de que la parte más brillante de la imagen debería ser blanca pura, es decir, tener valores máximos en los tres canales. Luego el algoritmo calcula el valor máximo en cada canal de color de la imagen y escala cada canal por su valor máximo, normalizando así la imagen, de tal forma que se garantiza que el punto de referencia aparezca como un blanco

puro, corrigiendo cualquier color dominante presente en la imagen original. El algoritmo tiene a su vez una variable percentil, la cual considera una mayor proporción de los píxeles más brillantes de la imagen como blancos, lo que resulta en una normalización más agresiva y brillante cuanto menor es este valor. Los resultados de aplicar este algoritmo se muestran en las Figuras 6.19, 6.20 y 6.21.



Figura 6.19: Ejemplo 1 del uso del algoritmo parche blanco

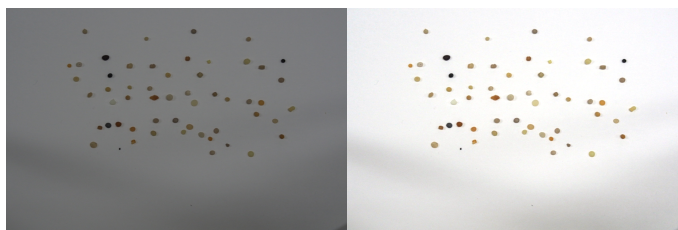


Figura 6.20: Ejemplo 2 del uso del algoritmo parche blanco



Figura 6.21: Ejemplo 3 del uso del algoritmo parche blanco

Podemos apreciar una gran mejora en la calidad de las Figuras 6.19 y 6.21, eliminando los colores de fondo amarillo y azul, y abrigantando significativamente la Figura 6.20. Hay que destacar que el hecho de que este algoritmo ejecute de esta forma se debe al hecho de que para la clasificación y cuenta de microplásticos siempre se use un fondo de papel blanco para su fácil identificación, lo que permite al algoritmo establecer fácilmente el blanco puro base de referencia.

Resultados de segmentación-clasificación de las imágenes en la Figura 6.20:

	Segm. sin	Segm. con	Clas. sin	Clas. con
Tiempo de ejecución	1m 15s	1m23s	30s	32s
Verdaderos positivos	34	35	18	26
Falsos positivos	2	2	16	9
Falsos negativos	21	20	0	0
Porcentaje de acierto	61 %	63 %	55 %	74 %

Tabla 6.4. Resultados de segmentación y clasificación con y sin balance de blanco

Se puede apreciar en la Tabla 6.4 una leve mejora en la segmentación, y una significativa mejora en la clasificación por colores, debido a que el modelo ya empieza a detectar los tonos de amarillo y marrón que antes detectaba como negro. A vista de estos resultados, se selecciona el algoritmo de parche blanco como valor por defecto, ya que muestra la mejor validación.

6.4.2. Función de superposición

Uno de los problemas de segmentación ya discutidos con anterioridad es el hecho de que a veces el mismo microplástico es enmascarado por el modelo varias veces, creando redundancias en la salida de datos, dando más microplásticos de los que realmente se han detectado. Estos errores se catalogaban anteriormente como positivos falsos, y un ejemplo de esto se puede comprobar en la Figura 6.22.



Figura 6.22: Ejemplo de superposición de máscaras

Para solucionar este problema se ha decidido crear un módulo de posprocesamiento de las máscaras, una vez el modelo las haya creado como salida de datos. El módulo comparará las máscaras entre sí y si en algún momento dos comparten el mismo espacio en la matriz binaria, significa que son parte del mismo microplástico, y se combinan en una sola.

```
def filter_masks(self, masks):
    masks_tuples = []
    for i in range(0, len(masks)-1):
        mask1 = masks[i]['segmentation'].copy()
        kernel = np.ones((3, 3), np.uint8)
        mask1 = np.multiply(mask1, 255)
        mask1 = mask1.astype('uint8')
        mask1 = cv2.dilate(mask1, kernel)
        for j in range(i+1, len(masks)):
            mask2 = masks[j]['segmentation']
            if np.logical_and(mask1, mask2).any():
                masks_tuples.append([i, j])
```

Este fragmento de código del módulo de superposición lidia con la identificación de las máscaras superpuestas, guardando las máscaras a combinar en la variable de 'tuplas de máscaras'. Antes de comprobar si se superponen, se realiza una dilatación de la máscara por si hay una pequeña separación de píxeles entre estas. Dado que las máscaras son matrices de valores binarios, la comprobación se puede realizar con un operando lógico AND, el cual solo devolverá Verdadero si dos valores de Verdadero se superponen, agilizando en gran parte el proceso de comparación.

Seguidamente, la combinación de las máscaras binarias se realiza con un operado lógico OR, y se cerciora de que el resto de las variables existentes dentro la variable 'máscara' como las coordenadas, área, o centro se transformen para reflejar las características de la nueva máscara binaria, con sus propias funciones.

Los resultados de este módulo son los siguientes:



Figura 6.23: Resultado de modulo de superposición de máscaras

Se pueden apreciar dos características principales en la Figura 6.23, y los resultados en la Tabla 6.5. La primera, la completa eliminación de los negativos falsos en las máscaras, es decir ya no hay máscaras duplicadas presentes. La segunda, e igual de importante, es el hecho de que la sección de segmentación ha multiplicado en tiempo de ejecución. Esto es debido a la naturaleza exponencial de la función de superposición, el cual crece significativamente a mayor es la cantidad de microplásticos detectados. Con un ejemplo de 150 microplásticos iniciales, el tiempo de ejecución de este módulo superó los 7 minutos.

	Segm. sin	Segm. con	Clas. sin	Clas. con
Tiempo de ejecución	3m 6s	10m20s	1m47s	1m55s
Verdaderos positivos	130	130	154	130
Falsos positivos	20	0	1	0
Falsos negativos	8	8	0	0
Porcentaje de acierto	94 %	94 %	100 %	100 %

Tabla 6.5. Resultados de segmentación y clasificación con y sin modulo de superposición

6.4.3. Escalado de imagen

El escalado de imágenes es el proceso de transformar el tamaño de una imagen, ya sea aumentando (upscaling) o reduciendo (downscaling) sus dimensiones in píxeles. Este proceso es fundamental en aplicaciones de visión por computador, ya que afecta directamente a la calidad, precisión y eficiencia de los algoritmos de procesamiento de imágenes en los modelos de redes neuronales. Además, es corriente la necesidad de tener un tamaño específico de imagen para que puedan ser procesadas como datos de entrada, debido al tamaño de la primera capa de una red neuronal.

Mientras que en ambos modelos usados un tamaño fijo no es necesario, ya que algoritmos interiores lidian con su tamaño, sí que pueden afectar a la calidad del resultado, tanto de mejor como de peor forma. Este módulo, parte de la sección de segmentación y mostrado en la Figura 6.24, lidiará con el escalado de imágenes previa a la segmentación con el objetivo de buscar una mejora de calidad de segmentación tanto en tiempo de ejecución como precisión.

```
def upscale(self, raw_image):
    if self.algorithm == 1:
        return raw_image
    if self.algorithm == 2:
        return self.DNNSS(raw_image)
    if self.algorithm == 3:
        return self.downscale(raw_image)

def DNNSS(self, raw_image):
    sr = cv2.dnn_superres.DnnSuperResImpl_create()
    sr.readModel(self.model_path)
    sr.setModel("edsr",4)
    raw_image = sr.upsample(raw_image)
    return raw_image

def downscale(self, raw_image):
    h, w, l = raw_image.shape
    ratio = self.downscale_size / w
    if ratio < 1:
        raw_image = cv2.resize(raw_image, None, fx=ratio, fy=ratio)
    return raw_image
```

Figura 6.24: Código de modulo de escalado

```
def DNNSS(self, raw_image):
    sr = cv2.dnn_superres.DnnSuperResImpl_create()
    sr.readModel(self.model_path)
    sr.setModel(self.model, self.model_num)
    raw_image = sr.upsample(raw_image)
    return raw_image
```

```

def downscale(self, raw_image):
    h, w, l = raw_image.shape
    ratio = self.downscale_size / w
    if ratio < 1:
        raw_image = cv2.resize(raw_image, None, fx=ratio, fy=ratio)
    return raw_image

```

En el escalado de imágenes existen varios algoritmos de interpolación que buscan aumentar la resolución de una imagen existente, manteniendo la calidad de visualización lo máximo posible. Sin embargo, en este módulo se ha decidido usar EDSR (Del inglés - Enhanced-Deep-Super-Resolution), modelos entrenados de redes neuronales que buscan superescalar imágenes con la mínima pérdida de resolución [4].

Resultados de imagen demo:

	Seg(1024x683)	Seg(2048x1366)	Clas(1024x683)	Clas(2048x1366)
T. de ejecución	1m 39s	4m 1s	1m 0s	1m 0s
Ver. posit.	52	52	68	68
Fal. posit.	4	4	3	3
Fal. negat.	9	9	0	0
P. de acierto	85 %	85 %	97 %	97 %

Tabla 6.6. Resultados de segmentación y clasificación con aumento de escalado

Se puede comprobar que los resultados obtenidos en la Tabla 6.6 son exactamente iguales, con un cambio en un aumento en el tiempo de ejecución. Esto es debido a que los algoritmos internos del modelo de segmentación normalizan la imagen antes de ser usadas, obteniendo resultados prácticamente iguales.

Veamos ahora los resultados obtenidos al reducir la resolución de la imagen, usando un sencillo algoritmo de escalado de la biblioteca cv2:

	Seg(4080x2868)	Seg(1024x720)	Clas(4080x2868)	Clas(1024x720)
T. de ejecución	3m 6s	1m 13s	1m 47s	1m 49s
Ver. posit.	130	130	154	94
Fal. posit.	20	20	1	54
Fal. negat.	8	8	0	0
P. de acierto	94 %	94 %	100 %	62 %

Tabla 6.7. Resultados de segmentación y clasificación con reducción de escalado

Se pueden observar dos cambios importantes al reducir el escalado de la imagen a un cuarto de su tamaño, mostrados en la Tabla 6.7. El primero, que el tiempo de segmentación se ha reducido casi a la mitad, debido a que el posprocesamiento interno de máscaras de SAM es mucho más lento cuanto mayor son las imágenes. Y lo segundo, que la clasificación por colores de esta imagen es mucho más imprecisa, debido a la pérdida de resolución de los microplásticos usados como datos de entrada.

Ante estos resultados, se ha decidido implementar una reducción de escala (por defecto a 1024 de ancho), obteniendo así una rápida segmentación, e implementar un posprocesamiento adicional que extraiga los microplásticos de la imagen original, usando transformaciones de las máscaras y coordenadas dadas, para poder mantener un grado elevado de precisión en la clasificación.

6.4.4. Filtrado de máscaras

Otro problema de segmentación que se encontró durante el desarrollo de este TFG fue el hecho de que algunos conjuntos de microplásticos eran segmentados como parte de la misma máscara por el modelo SAM. Esto causaba una subestimación en el conteo de microplásticos individuales, ya que varios microplásticos que deberían haber sido identificados como objetos separados eran agrupados en una sola región, además dificultando la clasificación de estos al haber varios microplásticos en una sola imagen. Como consecuencia, la precisión de la aplicación se veía afectada. Este fallo se puede comprobar en el ejemplo de máscara binaria mostrada en la Figura 6.25.

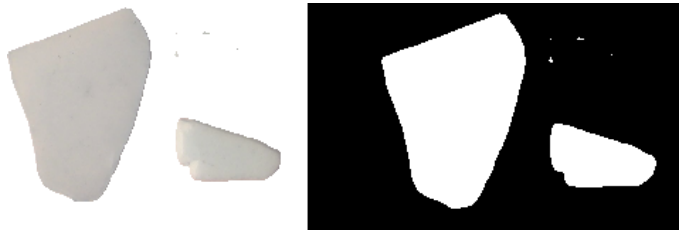


Figura 6.25: Múltiples microplásticos sobre una sola máscara

Como solución, se implementó un simple módulo de postprocesado que daba uso a una función en la biblioteca `cv2`, `'connectedComponentsWithStats()'` la cual detecta algorítmicamente zonas sin conectar en las máscaras binarias, y las separa. Una vez obtenidas las máscaras separadas, se añaden al conjunto, asegurándose además de que el resto de los valores como coordenadas y áreas también se ajusten a las nuevas máscaras.

6.4.5. Uso de parámetros

Como paso final, atenderemos a reducir la cantidad de negativos falsos, esto es la cantidad de microplásticos que no fueron detectados y segmentados en sus máscaras. Para ello, se indaga más profundamente en la interfaz de la aplicación de SAM, para poder manejar los parámetros de filtrado manualmente.

```

class SamAutomaticMaskGenerator:
    def __init__(
        self,
        model: Sam,
        points_per_side: Optional[int] = 32,
        points_per_batch: int = 64,

        #ORIGINAL PRED IOU THRESH = 0.88
        pred_iou_thresh: float = 0.88,

        #ORIGINAL STABILITY SCORE THRESH = 0.95
        #ORIGINAL STABILITY SCORE OFFSET = 1.0
        stability_score_thresh: float = 0.95,
        stability_score_offset: float = 1.0,
        box_nms_thresh: float = 0.7,
        crop_n_layers: int = 0,
        crop_nms_thresh: float = 0.7,
    ):

```

Figura 6.26: Parámetros de la interfaz SAM

De los parámetros que se encontraron, mostrados en la Figura 6.26 algunos de los de mayor interés eran los siguientes:

- Puntos por lado (points_per_side): La cantidad de puntos de muestreo realizados por cada eje de la imagen. A mayor cantidad de puntos, mayor resolución de búsqueda de regiones, y mayor posibilidad de detectar microplásticos, pero aumenta el tiempo de ejecución exponencialmente.
- Umbral de predicción IoU (pred_iou_thresh): En la visión por computador, IoU - Intersección sobre Union (Del inglés - Intersection Over Union), es una métrica de rendimiento que mide la superposición entre la predicción del modelo y el etiquetado manual. El modelo SAM, junto con una máscara, produce además una predicción de esta métrica, y si no es lo suficientemente elevada la máscara es filtrada y descartada. Reducir el umbral de predicción IoU significa generar más segmentaciones positivas, tanto verdaderas como falsas.
- Umbral de filtrado de estabilidad (stability_score_thresh): El modelo SAM genera máscaras de probabilidad, donde cada píxel de la imagen tiene un valor continuo entre 0 y 1, las cuales son umbralizadas para generar máscaras binarias. Dependiendo del valor de umbralizado, la máscara será distinta, y si la cantidad de cambio presente en la máscara dependiendo de su umbralizado se conoce como estabilidad. Este parámetro filtra en base a la estabilidad de las máscaras, reducir su valor filtrará menos máscaras, y por lo tanto aumentará el número de positivos tanto verdaderos como falsos.

Tras realizar cambios a los parámetros de filtrado y observar los resultados, se encontró que la mejor solución era aumentar la cantidad de puntos de muestreo en la imagen, dado que la posible razón de que algunos microplásticos no eran detectados se debe a que, con un valor por defecto de 32 puntos de muestreo por lado, algunos de los puntos no caen sobre los microplásticos al detectar. Luego aumentar este valor causaba que se detectaran, pero aumentaba también el tiempo de ejecución exponencialmente. Por defecto, se decidió usar 64 puntos de muestreo por lado, pero este valor se puede cambiar manualmente.

Con todos los módulos activos, los resultados son los siguientes:

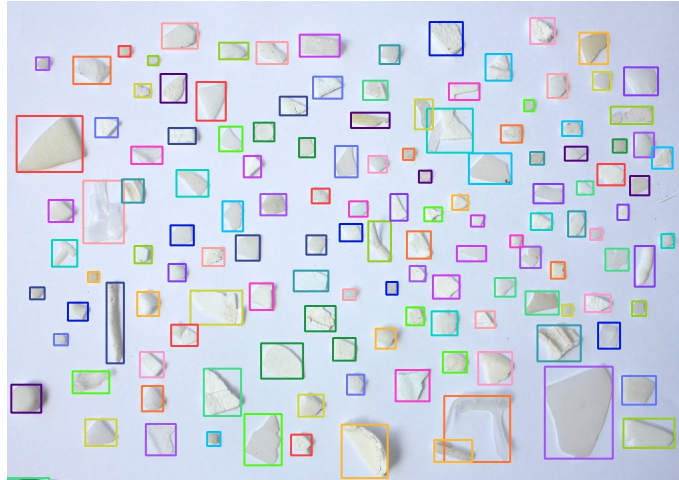


Figura 6.27: Segmentación con todos los módulos activos 1

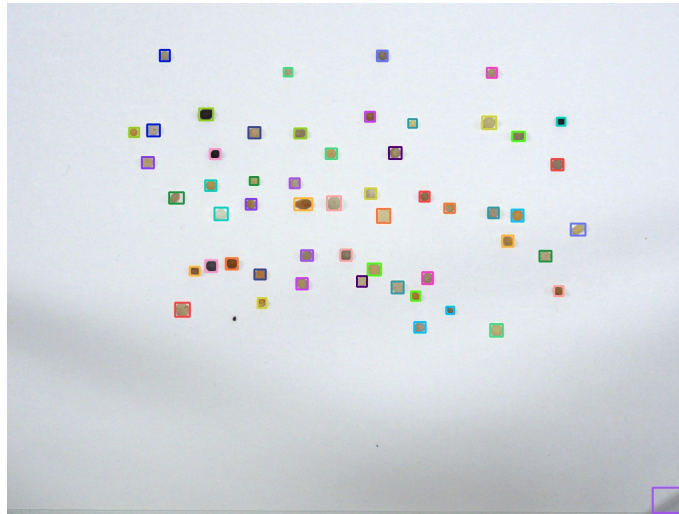


Figura 6.28: Segmentación con todos los módulos activos 2

Resultados de la Figura 6.27:

	Seg sin	Seg con	Clas sin	Clas con
T. de ejecución	3m 6s	4m 10s	1m 47s	1m 49s
Ver. posit.	130	135	154	136
Fal. posit..	20	1	1	1
Fal. negat.	8	0	0	0
P. de acierto	94 %	99 %	100 %	100 %

Tabla 6.8. Resultados de segmentación y clasificación con y sin todos los módulos

En las Figuras 6.27 y 6.28 podemos observar la segmentación de microplásticos usando cajas contenedoras en el postprocesado en vez de máscaras para su fácil visualización, y en la Tabla 6.8 la validación de los resultados de la Figura 6.27.

Se ha conseguido aumentar el porcentaje de acierto en segmentación y clasificación al máximo posible, eliminar casi por completo la presencia de positivos falsos, y eliminar problemas como la múltiple segmentación de microplásticos sobre una misma máscara, con un ligero aumento en el tiempo de segmentación. A la vista de estos resultados, se consideran cumplidos los objetivos propuestos de la aplicación.

6.5. Ajuste fino del modelo

El ajuste fino de un modelo, o 'fine-tuning' en inglés, es una técnica de aprendizaje en el entrenamiento de redes neuronales en el que se toma un modelo previamente entrenado y se vuelve a entrenar con un nuevo conjunto de datos para mejorar su rendimiento con una tarea específica en mente. Esto agiliza la producción de un modelo entrenado para una tarea específica ya que no requiere un entrenamiento y desarrollo desde cero, empleando otro como base.

Previamente, se desglosaron los varios modelos accesibles para el uso de Segment Anything Model, y como el modelo 'H', el más pesado y lento de todos, pero con mejor precisión, fue el usado por defecto durante el desarrollo de el TFG. Como objetivo adicional tanto desde el contexto de investigación como del de desarrollo de la aplicación, se decidió realizar un ajuste fino del modelo de segmentación base de SAM, el modelo 'B', el cual tenía una precisión pobre, pero menor peso. Realizar un ajuste fino tendrá por lo tanto dos objetivos para el proyecto, el primero, el de validar la capacidad del modelo base para ser afinado a una tarea y comprobar su cualidad de entrenamiento, y la segunda, la posibilidad de usar un modelo personalizado para el proyecto, de menor peso al estar basado en el modelo 'B' de SAM, y por lo tanto mucho más rápido y portátil.

6.5.1. Preparado de datos

Uno de los aspectos clave de todo entrenamiento de redes neuronales es el conjunto de datos usado, y garantizar que sea representativo de la tarea objetivo, variado en la diversidad de casos presentes y sobre todo volumen para un entrenamiento pleno, cuidando de no tener demasiados ejemplos similares lo que puede causar un sobreajuste que invalide la capacidad del modelo de usarse con casos nuevos. Ninguna de estas finalidades cambia con el ajuste fino de modelos, y por lo tanto la primera tarea a realizar es la producción de un conjunto de datos eficaz y capacitado para el ajuste fino del modelo preentrenado.

Para ello, se dio uso al conjunto de datos ya proporcionado y se seleccionaron la mitad de las imágenes que mejor representaban al conjunto entero con la mejor variedad de formas y colores. Posteriormente, con un editor de imagen, se dibujaron manualmente las máscaras binarias de cada microplástico que se usaran de 'ground truth', a modo de entrenamiento, en la Figura 6.29 se muestra un ejemplo de este preparado.



Figura 6.29: Ejemplo de preparado de máscaras de entrenamiento

6.5.2. Entrenamiento

Con el conjunto de datos establecido, se procedió al desarrollo del código de entrenamiento del modelo SAM. Usando un repositorio ya existente [12] y plenamente público como base inicial, y tras numerosos cambios y reestructuraciones para la adaptación a nuestra tarea específica de segmentación de microplásticos, se finalizó el código de entrenamiento.

```

EXTRACTING BOUNDING BOXES AND POINTS FOR MASKS

filtered_masks = np.array([large_mask])

parsed_masks = []
parsed_bbox = []
parsed_points = []

for iteration in range(0, len(filtered_masks)):

    #display(im_pil)
    #print(image.shape)

    output = cv2.connectedComponentsWithStats(
        filtered_masks[iteration], 8, cv2.CV_32S)
    (numLabels, labels, stats, centroids) = output

    # loop over the number of unique connected component labels
    > for i in range(0, numLabels):...

        #print(parsed_bbox)
        #print(parsed_images[0].shape)
        print("NEW SET: ", str(len(parsed_bbox)))

[5] ✓ 0.3s
... NEW SET: 112

```

Figura 6.30: Código de ajuste fino 1

En la primera sección del código mostrado en la Figura 6.30, realizamos una preparación del conjunto de datos para el entrenamiento propio. Se empieza dando nuevamente uso a la función de cv2 connectedComponentsWithStats() para separar todas las segmentaciones de microplásticos individuales, lo que nos devuelve un conjunto de datos de una sola imagen, para ser usada como entrenamiento junto con el resto.

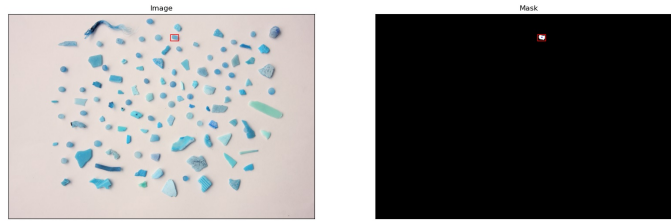


Figura 6.31: Ejemplo de una entrada de entrenamiento

Se puede ver en el ejemplo de la Figura 6.31, como se va a dar uso al prompt, un punto de coordenadas, con el ground truth como salida, la máscara resultado como expectativa.

```
# Set up the optimizer, hyperparameter tuning will improve performance here
lr = 1e-4
wd = 0
optimizer = torch.optim.Adam(sam_model.mask_decoder.parameters(), lr=lr, weight_decay=wd)

loss_fn = torch.nn.MSELoss()
# loss_fn = torch.nn.BCELoss()

[17] ✓ 0.0s
```

Figura 6.32: Código de ajuste fino 2

```
num_epochs = 5
losses = []

for epoch in range(num_epochs):
    epoch_losses = []
    # Just train on the first 20 examples
    for k in range(0, len(final_bbox)):
        input_image = input_image_raw.to(device)

        # No grad here as we don't want to optimise the encoders
        with torch.no_grad():
            image_embedding = sam_model.image_encoder(input_image)

        prompt_box = np.array(final_bbox[k])
        box_torch = torch.as_tensor(prompt_box, dtype=torch.float, device=device)
        box_torch = box_torch[None, :]

        prompt_points = np.array(final_points[k])
        points_torch = torch.as_tensor(prompt_points, dtype=torch.float, device=device)
        labels_torch = torch.as_tensor(np.array([1]), dtype=torch.float, device=device)
```

Figura 6.33: Código de ajuste fino 3

Con un conjunto de datos de entrenamiento preparado y posteriormente procesado, mostrado en las celdas de código de las Figuras 6.32 y 6.33 se prosigue a la sección de entrenamiento en la que primero se seleccionan las variables de entrenamiento importantes como:

- Función de pérdida (Loss function): La métrica que cuantifica la diferencia entre las predicciones del modelo y los valores reales (ground truth). Proporciona una medida del error, que el algoritmo de optimización utilizará para actualizar los pesos del modelo y mejorar su desempeño.
- Épocas (Epochs): Una iteración completa sobre el conjunto de datos de entrenamiento. En muchos casos, se requieren múltiples iteraciones sobre los datos para que el modelo aprenda correctamente y actualice sus pesos.

Seguidamente, se deja entrenar al modelo, actualizar sus pesos, y cerciorarse de que la función de pérdida actualiza de manera lógica. Los resultados fueron los siguientes:

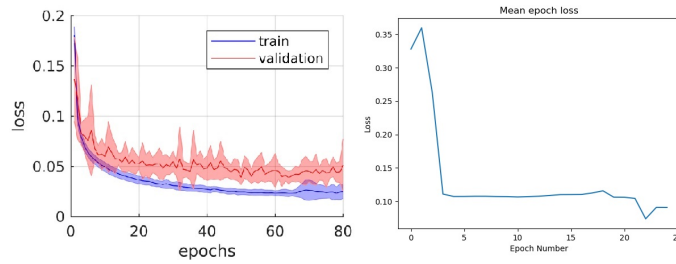


Figura 6.34: Comparativa de función de pérdida esperada (izq.) con función de pérdida resultante (der.) (ResearchGate [17])

Nótese que la función de pérdida resultante decrece inmediatamente, a diferencia de la de entrenamiento esperada que decrece suavemente con cada época, además de nivelarse anormalmente.

Desgraciadamente tras el entrenamiento nos encontramos que aunque la función de pérdida se asemeja a la apariencia esperada, mostrado en la Figura 6.34, tras realizar pruebas con el modelo personalizado nos encontramos que no hay diferencia respecto al modelo base, posiblemente debido a la poca cantidad de datos de entrenamiento accesible.

Como conclusión a esta sección del proyecto dedicada al ajuste fino del modelo base de SAM, terminamos entendiendo que no tenemos un conjunto de datos lo suficientemente grande para entrenar el modelo en base a la segmentación de microplásticos.

6.6. Interfaz de usuario

Una visión inicial del TFG contenía el despliegue de la aplicación en un entorno web para ser usada por cualquier usuario a través de un navegador, por medio de algún servidor o página web. Tras estudiar varias posibilidades, se descartó la idea ya que requeriría demasiados recursos o no eran del todo implementables. En su lugar, se decidió implementar un programa descargable a través de alguna plataforma para ser usado por los posibles usuarios.

La interfaz de usuario es el medio a través del cual un usuario interactúa con un sistema software. Comprende los elementos visuales y prácticos que permiten al usuario interactuar con dicho sistema, como botones, menús o formularios. La utilidad de la interfaz de usuario yace en la capacidad de permitir a un mayor grupo de gente ser usuarios de un sistema software.

Terminado por completo el desarrollo de la funcionalidad base de la aplicación de segmentación y clasificación, se procedió al desarrollo de la interfaz de usuario.

6.6.1. Línea de comando

Como primera fase de una interfaz de usuario, se decidió desarrollar una implementación de la aplicación por la línea de comando, en la que el usuario ejecuta el fichero Python por consola, introduciendo los argumentos necesarios como archivo de imágenes, además de otros parámetros de filtrado, dando uso así a la funcionalidad de la aplicación solo con el uso de una consola, la cual se puede apreciar en la Figura 6.35.

```
C:\Users\Alberto\Desktop\MicroplasticSegmenter>python segment_command.py -h
usage: segment_command.py [-h] [-o O] [--pps PPS] dir

segmentador/clasificador de microplasticos

positional arguments:
  dir                directory to examine

options:
  -h, --help        show this help message and exit
  -o O              directory to output, default is input dir
  --pps PPS         points per side, default is 64
```

Figura 6.35: Ejecución del comando ayuda de la implementación del programa por línea de comando

6.6.2. Interfaz gráfica de usuario

Como segunda interfaz, se decidió desarrollar una interfaz gráfica usando el ya descrito framework QTDesigner, el cual permite fácilmente la creación de un entorno grafico con el que un usuario no avanzado pueda interactuar cómoda e intuitivamente.

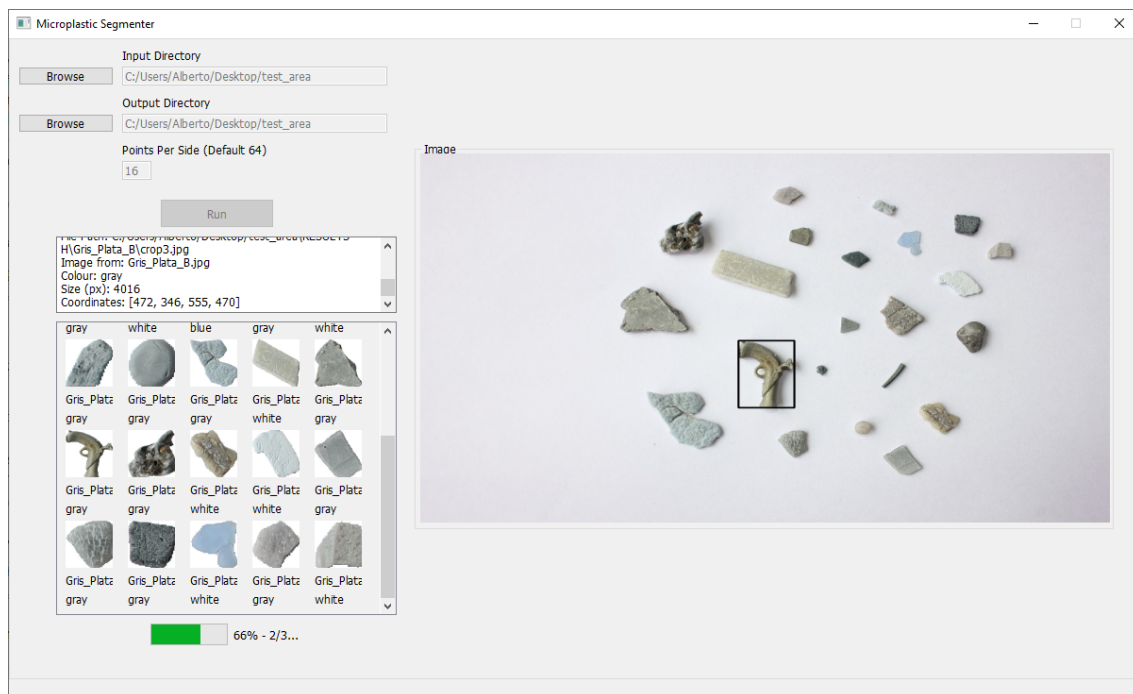


Figura 6.36: Interfaz gráfica de usuario

Además de tener todas las funcionalidades básicas de la aplicación por defecto, como el guardado de datos de salida procesados en una carpeta, la interfaz gráfica tiene además añadidos que facilitan su uso y mantienen al usuario informado del proceso de segmentación y clasificación en todo momento, lo cual se ve en la Figura 6.36

El primero, una reproducción de una consola de comando, el cual imprime constantemente los distintos pasos y módulos por el cual el programa está ejecutando, informando al usuario del estado del proceso. El segundo, un listado con las imágenes de todos los microplásticos detectados hasta el momento, el cual es interactivo y muestra por pantalla datos recopilados del microplástico en los que se hacen clic. además, se incluye una imagen de la fotografía que ha sido previamente analizada, mostrando con un recuadro la posición del microplástico en pantalla. Finalmente, se incluyó una barra de progreso la cual informa del estado completo del análisis, dictando cuantas imágenes se han segmentado y clasificado, y cuantas faltan.

Capítulo 7

Conclusiones

Este TFG fue propuesto con dos objetivos principales, el de actuar como corta investigación sobre las crecientes tecnologías en redes neuronales, explorando sus potenciales usos y capacidades para abordar tareas en el ámbito propuesto, de segmentación y clasificación, además de realizar una rápida valoración de los distintos modelos accesibles. Y el de desarrollar una aplicación que ayudara en gran medida y automatizara la laboriosa tarea de hacer un conteo y clasificación de los distintos microplásticos existentes, siendo el propuesto usuario la Facultad de Ciencias del Mar de la Universidad de Las Palmas de Gran Canaria en su ayuda contra la contaminación de microplásticos en los litorales canarios. Siendo el contexto de esta aplicación las crecientes preocupaciones respecto a la contaminación por microplásticos en todo el mundo, presentes en las nuevas directivas europeas de normativas de manipulación de plástico.

Durante el transcurso de este proyecto se han realizado numerosas indagaciones en tecnologías y materias de las cuales solo se comprendían superficialmente, elaborando exploraciones en una temática exponencialmente creciente y con multitud de usos, no solo teórica si no también práctica, poniendo dichos modelos y tecnologías a prueba, desarrollando comparativas y valoraciones entre ellas. Al mismo tiempo, se ha podido poco a poco desarrollar una aplicación fácil de usar que integraba los objetivos propuestos al principio del proyecto, incrementado en precisión y tiempo de ejecución a cada interacción de la aplicación. Como conclusión al respecto, se puede decir que los objetivos han sido cumplidos exitosamente, tanto los objetivos del TFG como objetivos personales de desarrollo de aplicaciones y aprendizaje sobre temas tecnológicos novedosos.

Es imposible concluir sin mencionar que este TFG hubiera sido imposible sin la ayuda de mis tutores, y sus numerosas sugerencias y recomendaciones. Aun así, en su estado final la aplicación deja muchas posibles mejoras abiertas por si se sigue desarrollando en un futuro. Mejoras tanto en el apartado visual y estético, con mejores menús, interactividad y decorativo para una más agradable experiencia de usuario, como en funcionalidad ya que siempre se puede mejorar la precisión de los modelos, añadido de más módulos, y mejora en tiempo de ejecución. La arquitectura modular de la aplicación deja además abierta la puerta a la fácil introducción de otros modelos distintos que surjan en un futuro.

Como conclusión, se considera que se han alcanzado todos los objetivos descritos al inicio del proyecto, se deja la puerta abierta a posibles mejoras en el futuro de la aplicación, incluida la implementación originalmente propuesta de despliegue por plataforma web y se espera que se le de uso en la lucha contra la contaminación de los entornos naturales.

Bibliografía

- [1] Fon Mathuros. «More plastic than fish in the ocean by 2050: report offers blueprint for change». En: *World Economic Forum*. Vol. 19. Fecha acceso: 14/04/2025. 2016. URL: <https://www.weforum.org/press/2016/01/more-plastic-than-fish-in-the-ocean-by-2050-report-offers-blueprint-for-change/>.
- [2] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. Fecha acceso: 14/04/2025. 2016. arXiv: 1506.02640 [cs.CV]. URL: <https://arxiv.org/abs/1506.02640>.
- [3] Joseph Redmon et al. «You only look once: Unified, real-time object detection». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 779-788.
- [4] Bee Lim et al. «Enhanced deep residual networks for single image super-resolution». En: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2017, págs. 136-144.
- [5] Kaiming He et al. *Mask R-CNN*. Fecha acceso: 14/04/2025. 2018. arXiv: 1703.06870 [cs.CV]. URL: <https://arxiv.org/abs/1703.06870>.
- [6] Alicia Herrera et al. «Novel methodology to isolate microplastics from vegetal-rich samples». En: *Marine pollution bulletin* 129.1 (2018), págs. 61-69.
- [7] Junnan Li et al. *BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation*. 2022. DOI: 10.48550/ARXIV.2201.12086. URL: <https://arxiv.org/abs/2201.12086>.
- [8] Alexander Kirillov et al. «Segment anything». En: *Proceedings of the IEEE/CVF international conference on computer vision*. 2023, págs. 4015-4026.
- [9] Vedant Palit et al. «Towards vision-language mechanistic interpretability: A causal tracing tool for blip». En: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, págs. 2856-2861.
- [10] Fiamma Ainhoa Mesa Santana. *Detección De Microplásticos En Playas Utilizando Técnicas De Visión Por Computador*. 2023.
- [11] Edwin Ortiz Campos. *Exploración de técnicas avanzadas de visión por computador para la segmentación eficiente de microplásticos*. 2024.
- [12] Dr. Sreenivas Bhattiprolu. *Python for Microscopists*. Fecha acceso: 14/04/2025. URL: https://github.com/bnsreenu/python_for_microscopists/blob/master/331_fine_tune_SAM_mito.ipynb.
- [13] Google Colab. *What is Markdown?* Fecha acceso: 14/04/2025. URL: https://colab.research.google.com/notebooks/markdown_guide.ipynb.

- [14] European Commission. *Microplastics - The EU aims to address the growing volume of microplastics in the environment by tackling the main sources of releases*. Fecha acceso: 14/04/2025. URL: https://environment.ec.europa.eu/topics/plastics/microplastics_en.
- [15] Directorate-General for Environment. *Proposal for a Regulation on preventing pellet losses to reduce microplastic pollution*. Fecha acceso: 14/04/2025. URL: https://environment.ec.europa.eu/publications/proposal-regulation-preventing-pellet-losses_en.
- [16] FacebookResearch. *Segment Anything Model (SAM)*. Fecha acceso: 14/04/2025. URL: <https://github.com/facebookresearch/segment-anything>.
- [17] Oliver Faust. *Automated detection of atrial fibrillation using long short-term memory network with RR interval signals*. Fecha acceso: 14/04/2025. URL: https://www.researchgate.net/figure/Training-and-testing-loss-function-over-the-epochs-The-solid-line-is-the-mean-and-the_fig3_326471505.
- [18] Matthieu Guillaumin. *Closed-Form Training of Conditional Random Fields for Large Scale Image Segmentation*. Fecha acceso: 14/04/2025. URL: https://www.researchgate.net/figure/Example-images-and-segmentation-masks-from-HorseSeg-left-and-DogSeg-right-datasets_fig1_261182364.
- [19] Jupyter. *Project Jupyter Documentation*. Fecha acceso: 14/04/2025. URL: <https://docs.jupyter.org/en/latest/>.
- [20] LAVIS. *What is LAVIS?* Fecha acceso: 14/04/2025. URL: <https://opensource.salesforce.com/LAVIS/latest/intro.html>.
- [21] Francisco David Meneses-Bautista. *Pronóstico del tipo de cambio USD/MXN con redes neuronales de retropropagación*. Fecha acceso: 14/04/2025. URL: https://www.researchgate.net/figure/Red-neuronal-artificial-de-cuatro-capas_fig1_323985249.
- [22] Python. *Python Home Page*. Fecha acceso: 14/04/2025. URL: <https://www.python.org/>.
- [23] Pytorch. *PyTorch documentation*. Fecha acceso: 14/04/2025. URL: <https://pytorch.org/docs/stable/index.html>.
- [24] QtDesigner. *Qt Widgets Designer Manual*. Fecha acceso: 14/04/2025. URL: <https://doc.qt.io/qt-6/qtdesigner-manual.html>.
- [25] Supervision. *Supervision - Roboflow Home Page*. Fecha acceso: 14/04/2025. URL: <https://supervision.roboflow.com/>.
- [26] TensorFlow. *Introduction to TensorFlow*. Fecha acceso: 14/04/2025. URL: <https://www.tensorflow.org/learn>.
- [27] Ultralytics. *Ultralytics, YOLO*. Fecha acceso: 14/04/2025. URL: <https://github.com/ultralytics/ultralytics>.
- [28] VSCode. *Visual Studio Code documentation*. Fecha acceso: 14/04/2025. URL: <https://code.visualstudio.com/docs>.
- [29] Wikipedia. *Merlion*. Fecha acceso: 14/04/2025. URL: <https://en.wikipedia.org/wiki/Merlion>.

Anexo - Manual de usuario

Instalación

Todos los enlaces se hayan al final del documento de instalacion.

Instalar Python, version mínima 3.9 requerida.

Instalar Pytorch y Torchvision.

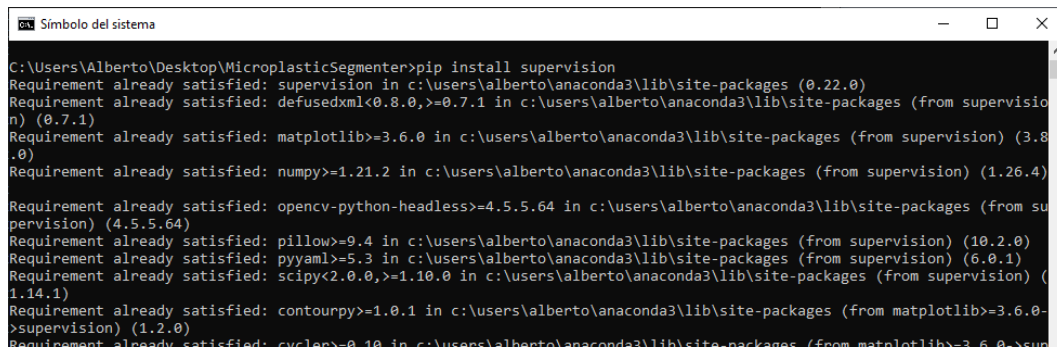
NOTE: Latest PyTorch requires Python 3.9 or later.

PyTorch Build	Stable (2.6.0)			Preview (Nightly)	
Your OS	Linux		Mac		Windows
Package	Conda		Pip	LibTorch	Source
Language	Python			C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.4	CUDA 12.6	ROCm 6.2.4	CPU
Run this Command:	pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118				

Figura 7.1: Instalación de Pytorch y Torchvision - Hay que asegurarse que CUDA sea compatible con su GPU, si no, usar CPU

Usando el entorno python de su conveniencia, por ejemplo, el CMD de Windows, instalar las siguientes biblioteca:

- segment-anything - Desde Enlace
- salesforce-lavis - Desde Enlace
- supervision - 'pip install supervision'
- webcolors - 'pip install webcolors'
- skimage - 'pip install skimage'
- PyQt5 - Desde enlace QtDesigner



```
C:\Users\Alberto\Desktop\MicroplasticSegmenter>pip install supervision
Requirement already satisfied: supervision in c:\users\alberto\anaconda3\lib\site-packages (0.22.0)
Requirement already satisfied: defusedxml<0.8.0,>=0.7.1 in c:\users\alberto\anaconda3\lib\site-packages (from supervision) (0.7.1)
Requirement already satisfied: matplotlib>=3.6.0 in c:\users\alberto\anaconda3\lib\site-packages (from supervision) (3.8.0)
Requirement already satisfied: numpy>=1.21.2 in c:\users\alberto\anaconda3\lib\site-packages (from supervision) (1.26.4)
Requirement already satisfied: opencv-python-headless>=4.5.5.64 in c:\users\alberto\anaconda3\lib\site-packages (from supervision) (4.5.5.64)
Requirement already satisfied: pillow>=9.4 in c:\users\alberto\anaconda3\lib\site-packages (from supervision) (10.2.0)
Requirement already satisfied: pyyaml>=5.3 in c:\users\alberto\anaconda3\lib\site-packages (from supervision) (6.0.1)
Requirement already satisfied: scipy<2.0.0,>=1.10.0 in c:\users\alberto\anaconda3\lib\site-packages (from supervision) (1.14.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\alberto\anaconda3\lib\site-packages (from matplotlib>=3.6.0->supervision) (1.2.0)
Requirement already satisfied: cv2>=0.10 in c:\users\alberto\anaconda3\lib\site-packages (from matplotlib>=3.6.0->supervision) (4.12.0)
```

Figura 7.2: Instalación desde CMD de Windows - Supervision

Descargar los siguientes modelos que se encuentran en el git de segment-anything:

- vit_h (REQUERIDO)
- vit_l (OPCIONAL)
- vit_b (OPCIONAL)

Descargar la aplicación y descomprimir.

Colocar modelos en la carpeta 'models'

Para ejecutar el programa en línea de comando por CMD de Windows, usar 'python segment_command.py'

Para ejecutar el programa con interfaz gráfica por CMD de Windows, usar el comando 'python segment_ui.py'

ENLACES:

Python: <https://www.python.org/downloads/>

Pytorch y Torchvision: <https://pytorch.org/get-started/locally/>

Segment Anything: <https://github.com/facebookresearch/segment-anything>

Modelos:

vit_h: https://dl.fbaipublicfiles.com/segment_anything/sam_vit_h_4b8939.pth

vit_l: https://dl.fbaipublicfiles.com/segment_anything/sam_vit_l_0b3195.pth

vit_b: https://dl.fbaipublicfiles.com/segment_anything/sam_vit_b_01ec64.pth

LAVIS: <https://github.com/salesforce/LAVIS>

Supervision: <https://github.com/roboflow/supervision>

QtDesigner: <https://build-system.fman.io/qt-designer-download>

Segmentador de Microplásticos:

<https://github.com/AlbertoFidalgo/Microplastic-Segmenter>

Uso

FUNCIONAMIENTO:

El programa está diseñado para la búsqueda y clasificación de microplásticos en una imagen de fondo blanco usando las bibliotecas de segment-anything y LAVIS. Localizará en la imagen los microplásticos, y clasificará por color, guardando los resultados en una carpeta previamente seleccionada.

Línea de Comando

EJECUCIÓN:

`segment_command.py:`

Programa de ejecucion por terminal.

Ejecutar usando comando 'python' por terminal, o interfaz de desarrollo preferida.

Usar opcion '-h' para ver menu de ayuda.

Ejemplo (Ejecutando por CMD de Windows):

Primero, acceder a la carpeta contenedora por línea de comando.

'python segment_command.py -h' - Menu de ayuda del programa

'python segment_command.py

'C:\Users\Ejemplo\Desktop\Carpeta_De_Microplasticos'

Ejecutara el programa usando la carpeta 'Carpeta de microplasticos'

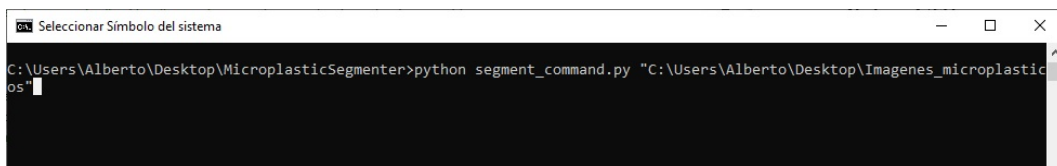


Figura 7.3: Uso de aplicación por CMD de Windows

Interfaz Gráfica

EJECUCION:

segment_ui.py:

Programa de interfaz gráfica.

Primero, acceder a la carpeta contenedora por línea de comando.

Ejecutar usando comando 'python' por terminal, o interfaz de desarrollo preferida.

Tener en cuenta que desde el CMD de Windows tenemos que estar dentro de la carpeta contenedora del programa.

Ejemplo: 'python segment_ui.py'

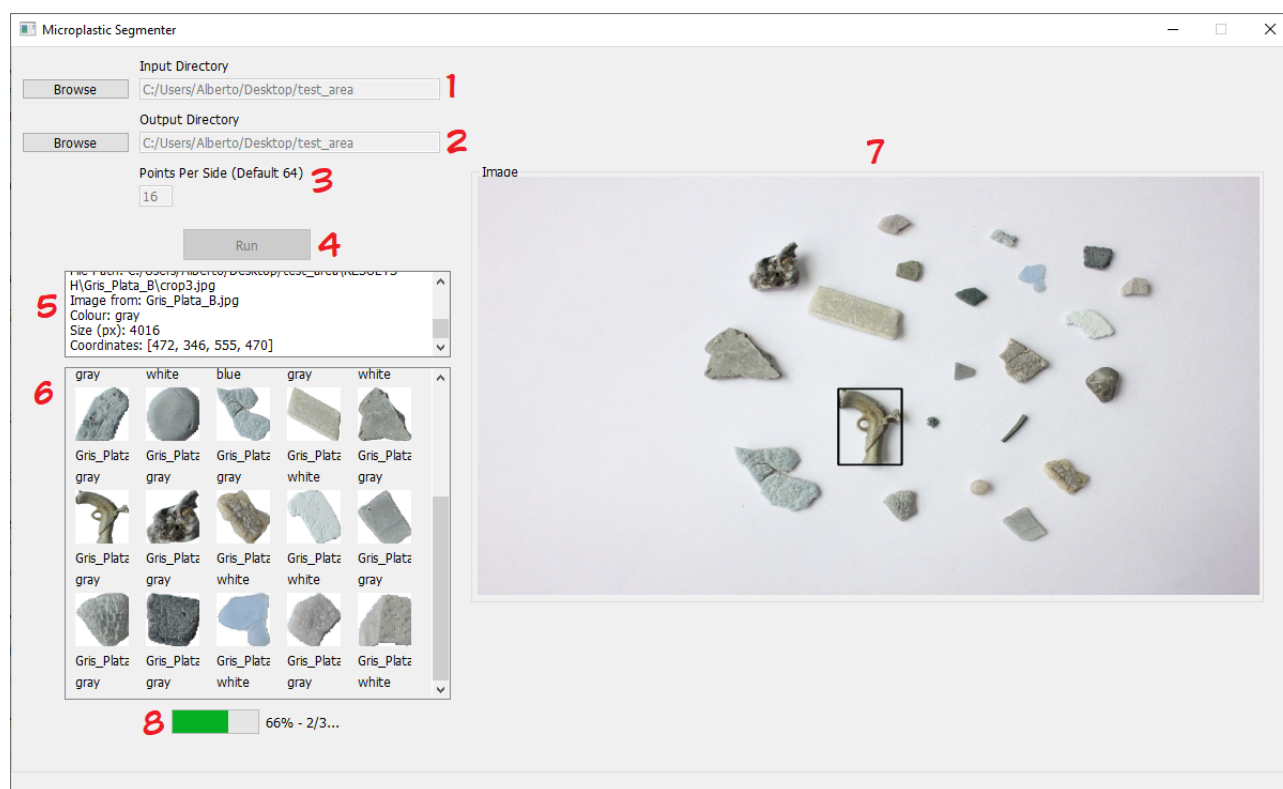


Figura 7.4: Uso de aplicación por interfaz gráfica

1. Input Directory: Carpeta donde se encuentran todas las imagenes a escanear. REQUERIDO
2. Output Directory: Carpeta donde se creara la carpeta de salida de datos. REQUERIDO
3. Points Per Side: Cantidad de puntos de segmentacion de segment-anything, se recomienda y usa 64 por defecto. REQUERIDO
4. Run: Inicializa la ejecucion del programa.
5. Terminal: Muestra el proceso de ejecucion del segmentador-clasificador. Además de datos adicionales.
6. Ventana de Microplasticos: Muestra los microplasticos detectados hasta el momento. Realizar Clic en alguno para mostrar datos sobre este, y mostrarlo en la ventana de la imagen.
7. Ventana de Imagen: Al hacer Clic en un microplastico, muestra la imagen origen.
8. Barra de progreso: Progreso para la finalizacion del programa.