



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería en Informática



TFG del Grado en Ingeniería Informática

**Sistema de reconocimiento  
automático en arqueobotánica**



Presentado por Jaime Sagüillo Revilla  
en Universidad de Burgos — 2 de junio de 2017  
Tutores: D. Álvar Arnaiz González, Dr.  
José Francisco Díez Pastor y **Dr.** Virginia Ahedo  
García





UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería en Informática



Álvar Arnaiz González, José Francisco Díez Pastor profesores del departamento de Departamento de Ingeniería Civil, Área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Jaime Sagüillo Revilla, con DNI 12782524-K, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Sistema de reconocimiento automático en arqueobotánica.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 2 de junio de 2017

Vº. Bº. del Tutor:

Vº. Bº. del co-tutor:

D. **nombre tutor**

D. **nombre co-tutor**



## Resumen

La arqueobotánica es la ciencia que estudia las interrelaciones de las poblaciones humanas antiguas con el mundo vegetal [3]. Obteniendo información de las actividades del ser humano relacionadas con las plantas como: técnicas de agricultura, consumo de vegetales, utilización de las plantas o información sobre los entornos de vegetación antiguos.

En concreto, nuestro objetivo se centra en la rama de los micro-restos y, más en concreto, en los fitolitos. Los fitolitos son partículas microscópicas generadas por las plantas, a causa de un proceso metabólico vital para ellas. La identificación de los fitolitos nos permite obtener, entre otra, la información anteriormente citada.

En este proyecto tratamos de solucionar la problemática de analizar cada una de las muestras microscópicas manualmente. Mediante la creación de un sistema que sea capaz de reconocer de forma automática los fitolitos en cada una de estas muestras.

Para ello, se comenzó mediante técnicas de segmentación, las cuales no aportaban grandes resultados, por su enorme ineficiencia en tiempos. Se continuó con la utilización de técnicas tradicionales, en el ámbito de la visión artificial, como es la ventana deslizante. Y, se finalizó, estudiando técnicas avanzadas de *deep learning*, o aprendizaje profundo, junto a técnicas de *data augmentation*.

Además, en la utilización de este tipo de técnicas se plantea un problema fundamental que es la necesidad de poseer imágenes con los fitolitos previamente identificados, por lo que se realizó un etiquetador de imágenes que permite a los expertos etiquetar los diversos tipos de fitolitos en las muestras.

## Descriptores

Arqueobotánica, Fitolito, Reconocimiento automático de objetos, Inteligencia artificial



**Abstract**

**Keywords**

---

# Índice general

---

|  |            |
|--|------------|
| <b>Índice general</b>  | <b>III</b> |
| <b>Índice de figuras</b>                                       | <b>V</b>   |
| <b>Índice de tablas</b>  | <b>VI</b>  |
| <b>Introducción</b>  | <b>1</b>   |
| <b>Objetivos del proyecto</b>                                  | <b>3</b>   |
| 2.1. Objetivos generales . . . . .                             | 3          |
| 2.2. Objetivos técnicos . . . . .                              | 3          |
| <b>Conceptos teóricos</b>                                      | <b>5</b>   |
| 3.1. Inteligencia artificial . . . . .                         | 5          |
| 3.2. Minería de datos . . . . .                                | 6          |
| 3.3. Procesamiento de imágenes . . . . .                       | 7          |
| 3.4. Detección de objetos . . . . .                            | 8          |
| 3.5. Aprendizaje profundo . . . . .                            | 9          |
| <b>Técnicas y herramientas</b>                                 | <b>12</b>  |
| 4.1. Lenguajes de programación . . . . .                       | 12         |
| 4.2. <i>Anaconda</i> . . . . .                                 | 13         |
| 4.3. Librerías auxiliares . . . . .                            | 14         |
| 4.4. Control de versiones: <i>Git</i> . . . . .                | 15         |
| 4.5. Repositorio: <i>Github</i> . . . . .                      | 15         |
| 4.6. Documentación: $\text{\LaTeX}$ . . . . .                  | 16         |
| 4.7. Entorno de desarrollo: <i>JetBrains PyCharm</i> . . . . . | 16         |
| 4.8. Herramienta de prototipado: <i>NinjaMock</i> . . . . .    | 16         |
| <b>Aspectos relevantes del desarrollo del proyecto</b>         | <b>17</b>  |

|   |           |
|---|-----------|
| <i>ÍNDICE GENERAL</i>   | IV        |
| 5.1. Procesamiento de imágenes . . . . .                                      | 17        |
| 5.2. Clasificadores . . . . .   | 18        |
| 5.3. Conclusiones . . . . .   | 22        |
| 5.4. Problema fundamental: falta de imágenes . . . . .                        | 23        |
| 5.5. Detección de objetos mediante técnicas de <i>deep learning</i> . . . . . | 23        |
| <b>Trabajos relacionados</b>  | <b>25</b> |
| <b>Conclusiones y Líneas de trabajo futuras</b>                               | <b>27</b> |
| 7.1. Conclusiones . . . . .   | 27        |
| 7.2. Líneas futuras de trabajo . . . . .                                      | 27        |
| <b>Bibliografía</b>   | <b>28</b> |

---

# Índice de figuras

---

|   |    |
|---|----|
| 3.1. Ejemplo de imagen tras aplicar <i>thresholding</i> . . . . .       | 8  |
| 3.2. Imagen predicha por <i>YOLO</i> . . . . .                          | 10 |
| 4.3. Ejemplo de diagrama <i>burndown</i> de un <i>sprint</i> . . . . .  | 15 |
| 5.4. Conversión de la imagen original a escala de grises . . . . .      | 18 |
| 5.5. Distintos ejemplos de una imagen segmentada . . . . .              | 19 |
| 5.6. Resultados obtenidos mediante transformación divisoria . . . . .   | 20 |
| 5.7. Esquema del funcionamiento de la ventana deslizante . . . . .      | 20 |
| 5.8. Resultados tras aplicar el clasificador sobre una imagen . . . . . | 22 |

---

# Índice de tablas

---

|   |    |
|---|----|
| 3.1. Comparativa entre los distintos sistemas de detección de objetos | 11 |
|---|----|

---

# Introducción

---

Este trabajo esta desarrollado en colaboración con investigadores del CSIC, **especialmente** con Débora Zurro y Virginia Ahedo, los cuales serán los principales usuarios de los productos *software* desarrollados en este proyecto.



**Está compuesto de un conjunto de herramientas, que tienen el fin de desarrollar un sistema capaz de reconocer automáticamente fitolitos.** Crear un sistema de este tipo es una tarea compleja, ya que lleva consigo un gran número de problemáticas a resolver, más allá de los problemas implícitos que tiene un sistema de visión artificial, entre las cuales se encuentran las siguientes:

- No poseemos un conjunto de imágenes de fitolitos etiquetadas, con los tipos de fitolitos que las componen y otra información necesaria. Base fundamental para la construcción de un sistema de aprendizaje automático.
- Los fitolitos son de distintos tamaños y tridimensionales, pero las fotografías son planas, en 2D.
- Las imágenes microscópicas de fitolitos, no solo contienen fitolitos, sino que contienen otros materiales, como restos de materia inorgánica.
- Los fitolitos pueden estar superpuestos entre sí. Debido a que se fotografían fitolitos en disolución.



Debido a que no poseemos dichas imágenes **desde un principio**, muchas de las tareas que se podrán ver en este proyecto se realizarán con **caras**. Se utilizarán caras puesto que las bases de datos de caras son mucho más comunes y nos permitirán tener una primera aproximación al problema.



Debido al problema de la falta de imágenes de estas características, nos veremos obligados a crear un etiquetador de imágenes<sup>1</sup>. El cual, nos permitirá obtener toda la información necesaria para tener un conjunto de imágenes que nos permitan llevar a cabo el sistema automático de reconocimiento de fitolitos.

Como más tarde iremos viendo, la mayoría de los productos generados en este proyecto son *Jupyter Notebooks*, los cuales nos permiten interactuar fácilmente con el código. Se explican en detalle en el capítulo **de técnicas y herramientas**. Cada uno de estos *notebooks* contendrán estudios sobre algunas herramientas o técnicas utilizadas o investigadas.

Finalmente, y a modo de aclaración, para llevar a cabo este sistema se irán estudiando diferentes técnicas, como previamente he comentado en el resumen. Comenzando por la técnica de segmentación, continuando con la ventana deslizante y avanzando hasta técnicas más avanzadas, como *deep learning*[24].



---

<sup>1</sup>Un etiquetador de imágenes es una herramienta que permite identificar donde se encuentran los diferentes objetos en una imagen.

---

# Objetivos del proyecto

---

En este apartado se explican los distintos objetivos identificados en este proyecto. Distinguiendo entre los objetivos generales del proyecto y los objetivos técnicos.

## 2.1. Objetivos generales

Los objetivos generales que plantea este proyecto son los siguientes:

- Realizar un estudio de las técnicas del estado del arte que solucionen el reconocimiento automático de fitolitos con la mejor precisión y eficiencia posible.
- Crear una aplicación para el etiquetado de fitolitos, mediante la cual se extraiga toda la información necesaria.
- Crear un sistema de reconocimiento automático de fitolitos. Mediante el cual un usuario sea capaz de introducir cualquier imagen que desee para realizar este reconocimiento automático.

## 2.2. Objetivos técnicos

Los objetivos técnicos que plantea este proyecto son los siguientes:

- Utilizar *Python* para crear todo lo que involucra este sistema, como lenguaje de programación principal.
- Usar librerías para *Python*, como *scikit-image*[\[12\]](#), que nos permitan llevar a cabo las tareas más complejas del proyecto.

- Crear una aplicación, basada en los *Jupyter Notebooks*, que permita una fácil instalación y uso a los usuarios. Y que sea escalable como aplicación web.
- Utilizar un sistema de control de versiones, en nuestro caso *Git*, junto a un servicio central, *GitHub*. Para mantener un correcto control de los productos generados.
- Utilizar alguna herramienta para la gestión del proyecto, en nuestro caso *ZenHub*. Que nos permita hacer un seguimiento de la metodología Scrum.
- Utilizar herramientas de prototipado para llevar a cabo las interfaces de usuario.

---

# Conceptos teóricos

---



Para la comprensión de este proyecto es necesaria la explicación de algunos conceptos teóricos que introduciré en este apartado. Para ello este apartado se **verá** organizado en cinco bloques principales: inteligencia artificial, procesamiento de imágenes, minería de datos, *deep learning* y detección de objetos.

## 3.1. Inteligencia artificial

Inteligencia artificial, que comúnmente aparece como las siglas IA o AI **del inglés**, consiste en otorgar a las máquinas la capacidad de realizar acciones que habitualmente son realizadas por humanos, imitando la inteligencia cognitiva humana [2]. Esta podría ser una posible definición de Inteligencia artificial, pero existen multitud [11]. Algunos posibles ejemplos conocidos por todos en los que se aplica la IA son el coche autónomo, videojuegos, asistentes personales, reconocimiento facial en imágenes, entre otros.

### Aprendizaje automático

Aprendizaje automático, o en inglés *machine learning*, es un campo de la informática cuyo objetivo es dar a los computadores la capacidad de aprender sin explícitamente haber sido programado [19]. **Por lo tanto, se encarga de construir modelos para problemas en los que un algoritmo programado no puede conseguir buenos resultados o es muy complejo hacer que lo sean.**

### Aprendizaje supervisado y no supervisado

En esta sección vamos a distinguir entre aprendizaje supervisado y no supervisado, ambas técnicas pertenecientes al campo de **machine learning**.

Aprendizaje supervisado es una técnica que consiste en la inferencia de una función partiendo de un conjunto de datos de entrenamiento etiquetado [22].

Dos de las tareas más comunes del aprendizaje supervisado son clasificación y regresión. Es decir, para tareas en las que el resultado deseado sea obtener una variable que nos informe a que clase pertenece una determinada instancia, lo cual es un valor discreto, o para tareas en las que se desea obtener un valor continuo. Dos posibles ejemplos, a modo de aclaración, podrían ser: la clasificación de *emails* en *spam* o *no spam* y la predicción del valor de un inmueble.



En cuanto al otro tipo de aprendizaje, el aprendizaje no supervisado consiste en la obtención de la estructura oculta en los datos. Pero al contrario que en el aprendizaje supervisado, partiendo de un conjunto de datos sin etiquetar, es decir, datos para los cuales no poseemos su valor deseado [23]. Una tarea muy común en aprendizaje no supervisado es el **clustering** o agrupamiento que consiste en la agrupación de los datos **en un número de grupos determinado**. Un posible ejemplo de la utilización de este tipo de aprendizaje podría ser la categorización de clientes en varios grupos con fines comerciales.

## 3.2. Minería de datos

### Clasificación

Clasificación, como ya se ha introducido previamente, es la tarea de identificar, entre un conjunto de categorías o clases<sup>2</sup>, a que categoría o clase pertenece una determinada instancia.



En el caso de este proyecto, estamos tratando de clasificar los distintos tipos de fitolitos. Por lo tanto, tendremos tantas clases como tipos de fitolitos.

### Reconocimiento de patrones

El reconocimiento de patrones consiste en la extracción de propiedades similares entre las distintas instancias de una clase [21]. De manera que podamos identificar un determinado objeto en función de los patrones que contiene este. Este concepto, como podemos observar, está íntimamente relacionado con el concepto de clasificación y *machine learning*.

### *Bag of Words*

*Bag of Words* (BoW), o en español bolsa de palabras, es una técnica comúnmente utilizada en la clasificación de documentos [16]. En la que se describe un texto mediante el conjunto de palabras que componen dicho documento.

---

<sup>2</sup>Se entiende por clase a cada uno de los tipos, o categorías, en las que se desean clasificar las instancias. Por ejemplo, en un problema en el que deseemos clasificar imágenes como personas y perros. Las clases serán: persona y perro.

Pero esta técnica también ha sido aplicada al reconocimiento de objetos en imágenes. Siendo, la técnica que mejores resultados conseguía, hasta la llegada del *Deep Learning*. En este caso, se tratan las características de una imagen como palabras.



### 3.3. Procesamiento de imágenes

#### Segmentación

La segmentación en el campo de la visión artificial, como se indica en la [wikipedia](#), consiste en subdividir una imagen en varios [píxeles](#) u objetos [18]. Cuando segmentamos una imagen, lo que pretendemos hacer es cambiar su representación para poder obtener de esta una mayor utilidad o cantidad de información.

En nuestro caso, segmentamos la imagen para eliminar el fondo de ella y obtener así una imagen con solo su parte delantera. De esta manera, eliminamos el ruido que existe en la imagen y, a su vez, la simplificamos reteniendo la parte de la imagen en la que se encuentran los objetos que nos interesan.



Posteriormente a este paso, nos interesa, como es obvio, dividir la parte delantera de la imagen resultante en objetos. De este modo, obtendremos cada uno de los objetos por separado de forma idónea.

#### Binarización

La binarización de una imagen consiste en la simplificación de los valores de cada píxel a 2 posibles valores, blanco o negro, representando el fondo y el frente de la imagen cada uno de ellos. Esta técnica nos permite conservar únicamente la información que nos interesa, eliminando el resto.

#### Thresholding

Es el método más simple para la segmentación de una imagen, pudiéndose utilizar para la binarización de una imagen, como es nuestro caso. Consiste en reemplazar los píxeles por debajo de una determinada constante a píxeles negros, y los que se encuentran por encima a píxeles blancos o viceversa. Un ejemplo de una imagen a la que se le ha aplicado esta técnica se muestra en la figura 3.1.

Existen distintas maneras de llevar a cabo este proceso, siendo uno de los más conocidos el método de Otsu [20].

#### Descriptores visuales

Los descriptores visuales, o descriptores de características, son descripciones de las características visuales de los contenidos en imágenes o videos, en

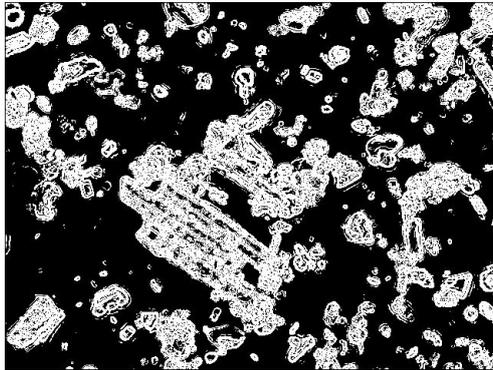


Figura 3.1: Ejemplo de imagen tras aplicar *thresholding*.

nuestro caso de imágenes, con el propósito de la detección de objetos [15]. El objetivo de los descriptores visuales es obtener la información que resulta significativa, eliminando a su vez la que no lo es. Así, utilizaremos la información que el descriptor nos proporciona para detectar los objetos que nos interesan en una imagen. Algunos ejemplos de características son la forma, el color o la textura.

Como se puede imaginar, obtener las características a mano es una tarea complicada y que usualmente no funciona correctamente. Por ello, utilizamos un método de extracción automática de características como es *Histogram of Oriented Gradients*, el cual se basa en los gradientes de la imagen para detectar los distintos objetos que se encuentran en la imagen [17].



### 3.4. Detección de objetos

#### Reconocimiento de objetos contra Detección de objetos

Antes de abordar el siguiente concepto, debemos de comprender la diferencia entre el término reconocimiento de objetos y detección de objetos.

Reconocimiento de objetos es el término utilizado cuando se desea detectar todos los objetos para los que ha sido entrenado el clasificador. Proporcionándonos el tipo de objeto y las coordenadas de la caja que rodea ese objeto. Dependiendo del algoritmo de clasificación empleado, también puede devolver las probabilidades de que ese objeto sea un verdadero positivo o un falso positivo.

En cuanto a la detección de objetos, en este solo se desea obtener si es objeto o no [redacted]. Simplificando el problema anterior, de manera que, pasamos de un problema multiclase a un problema con dos clases, o binario. Objeto o no objeto.

### 3.5. Aprendizaje profundo

Aprendizaje profundo, o más comunmente mencionado en inglés mediante las palabras *deep learning*, es la técnica más avanzada, catalogada como el estado del arte, para ámbitos como la visión artificial, el reconocimiento de voz automático, el procesamiento del lenguaje natural, la bioinformática y el reconocimiento de audio, entre otros campos [4].

*Deep learning* es un conjunto de algoritmos de la rama del aprendizaje automático que se caracteriza por los siguientes aspectos, principalmente [4]:

1. Consisten en modelos con varias capas en las que la información sufre a menudo operaciones no lineales.
2. Consisten en representaciones de características que parten desde un nivel bajo de abstracción hasta llegar a un nivel alto. Por ejemplo, desde los píxeles de la imagen hasta las clases de objetos y sus coordenadas dentro de la imagen.

Este conjunto de técnicas es la intersección entre las redes neuronales, la inteligencia artificial, el modelado de grafos, la optimización, el reconocimiento de patrones y el procesado de señales [4].

Pero no todo son ventajas en esta aproximación. Y es que existe un problema principal: el cual es el volumen de imágenes que necesita un modelo de este tipo para ser adecuadamente entrenado, del orden de miles o cientos de miles. Y, por otro lado, el tiempo necesario para entrenar un modelo de este tipo. Más adelante indagaremos en posibles soluciones.



#### **YOLO**

*YOLO*, o *You Only Look Once*, es una aproximación innovadora en la detección de objetos, considerado actualmente, como el estado del arte en esta materia, junto a *Faster RCNN* [10]<sup>3</sup>. Tratando de crear una arquitectura que funciona en tiempo real pero, a su vez, siendo capaz de soportar un número de clases mayor a 9000 [9].

#### **Versiones**

*YOLO* actualmente tiene dos versiones. Y, en cada una de estas versiones, se ha desarrollado una versión pequeña y una normal. En la primera versión de *YOLO* se consiguió llegar a resultados muy positivos, como la capacidad de

---

<sup>3</sup>Faster RCNN es un detector de objetos en tiempo real con una precisión similar a la última versión de *YOLO*, pero con menor rendimiento en tiempos que *este*.

procesar 45 imágenes por segundo con una precisión de 63.4  $mAP$ <sup>4</sup>. Y, con la versión pequeña de este, se llegaron a procesar 155 imágenes por segundo con una precisión de 52.7  $mAP$  [8]. Precisiones muy positivas, pero que todavía no estaban a la altura del estado del arte a nivel de precisión.

En la segunda versión, se considera a *YOLO* como el estado del arte, junto a *Faster RCNN*. En esta versión, *YOLO* consigue llegar a una precisión de 76,8  $mAP$ , procesando 67 imágenes por segundo, y 78,6  $mAP$ , procesando 40 imágenes por segundo. Lo cual, supera a *Faster RCNN* en precisión y rapidez, siendo esta última ampliamente superada [9].<sup>5</sup>

### Característica diferenciadora de *YOLO*

*YOLO* otorga un enfoque distinto respecto a sus competidores. Mientras que en otros casos, como *Faster RCNN*, se separan las distintas etapas del procesamiento de una imagen. *YOLO*, he de aquí su nombre, solo necesita de un único «vistazo» para predecir la imagen y obtener las coordenadas, o cajas que rodean a cada uno de los objetos predichos [8]. Véase la figura 3.2.

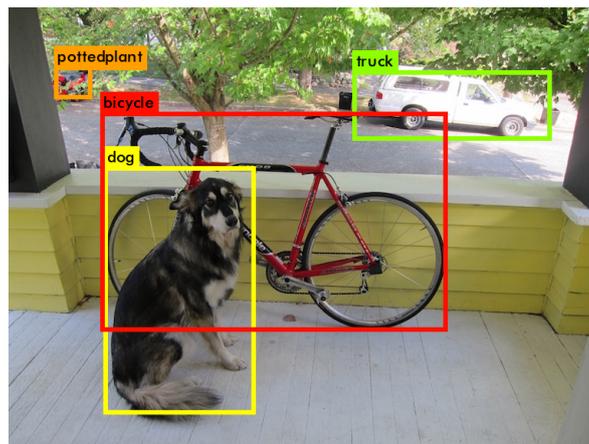


Figura 3.2: Imagen predicha por *YOLO*

Por supuesto, esta perspectiva no es tan simple como un giro en el enfoque. Sino que, además, implementa otras características, como la normalización en lotes o la obtención automática del tamaño de las cajas, que la permiten ser la mejor aproximación actual en este campo [9].

Podemos ver una comparativa entre los mejores detectores de objetos actuales con *YOLO* en la tabla 3.1. Indicando para cada uno de ellos la precisión

<sup>4</sup> *Mean average precision* ( $mAP$ ), o Media promedio de precisión, es comúnmente utilizada como una medida de evaluación de la precisión en la detección de objetos.

<sup>5</sup> *Faster RCNN* consigue clasificar una imagen cada dos segundos (*FPS*, *frames per second*). En cambio, *YOLO* es capaz de procesar más de 40 imágenes.

Tabla 3.1: Comparativa entre los distintos sistemas de detección de objetos sobre el mismo conjunto de entrenamiento y sobre la misma configuración [9].

| <b>Detection frameworks</b> | <b>Train</b> | <i>mAP</i> | <b>FPS</b> |
|-----------------------------|--------------|------------|------------|
| Fast R-CNN                  | 2007+2012    | 70.0       | 0.5        |
| Faster R-CNN VGG-16         | 2007+2012    | 73.2       | 7          |
| Faster R-CNN Resnet         | 2007+2012    | 76.4       | 5          |
| YOLO                        | 2007+2012    | 63.4       | 45         |
| SDD300                      | 2007+2012    | 74.3       | 46         |
| SDD500                      | 2007+2012    | 76.8       | 19         |
| YOLOv2 288 x 288            | 2007+2012    | 69         | 91         |
| YOLOv2 352 x 352            | 2007+2012    | 73.7       | 81         |
| YOLOv2 416 x 416            | 2007+2012    | 76.8       | 67         |
| YOLOv2 480 x 480            | 2007+2012    | 77.8       | 59         |
| YOLOv2 544 x 544            | 2007+2012    | 78.6       | 40         |

y la capacidad de procesamiento de imágenes, en las unidades de media promedio de precisión e imágenes por segundo, respectivamente.

---

# Técnicas y herramientas

---

En esta sección se explican las distintas técnicas y herramientas utilizadas para llevar a cabo el proyecto.

## 4.1. Lenguajes de programación

### *Python*

*Python* es el lenguaje de programación principal utilizado para el desarrollo de este proyecto. *Python* es un lenguaje de alto nivel, interpretado, estructurado y de código abierto que puede ser usado para tareas de muchos tipos [14].

*Python* se caracteriza por muchos aspectos. Y tiene múltiples ventajas respecto a otros lenguajes, como la fácil legibilidad de su código, las potentes funciones que incorpora por defecto, la utilización de tipado dinámico y las múltiples librerías de código abierto disponibles para distintas tareas, entre otras. Las ventajas que presenta respecto a otros lenguajes, como pueden ser C o Java, nos llevan a usar este lenguaje como el núcleo de nuestro proyecto.

### *JavaScript*

*JavaScript* es el lenguaje de programación de *HTML* y de la *Web* [13]. En nuestro caso, este es utilizado para la realización del etiquetador de imágenes, puesto que está basado en una aplicación *Web*. No existen alternativas estándar a él. Pero si existen librerías para facilitar el uso del mismo, como *JQuery*, la cual es utilizada en la medida de lo posible.

### *JSON*

*JSON*, siglas que denotan *JavaScript Object Notation*, es un formato ligero para el intercambio de datos [5]. Se basa en la notación de objetos de



*JavaScript*, de ahí su nombre. Este es utilizado para el almacenamiento de información en nuestra aplicación, como más tarde veremos.

Existe una posible alternativa a *JSON*, la cual es *XML*. Este es un lenguaje de marcado que puede ser utilizado con el mismo objetivo que *JSON*. Pero *XML* tiene distintas desventajas respecto *JSON*:

- *JSON* es más corto.
- *JSON* es más fácil de leer.
- *JSON* se integra fácilmente con *Python*<sup>6</sup>.

## 4.2. *Anaconda*

*Anaconda* es una distribución de *Python* y *R* que facilita las tareas de gestión de paquetes, de entornos y de versiones de lenguajes de programación. Nos permite crear varios entornos con distintas versiones de paquetes o lenguajes de programación. Y nos permite la instalación, desinstalación y actualización de distintos paquetes. Facilitándonos estas tareas en la medida de lo posible.

*Anaconda* nos provee de muchas ventajas respecto a la utilización de *Python* de manera aislada. La utilización de distintas versiones del mismo lenguaje en un mismo sistema operativo es una problemática que *Anaconda* nos soluciona aislándonos de todos estos problemas. Además, como previamente se ha comentado sobre la gestión de paquetes, *Anaconda* nos provee de una herramienta, «*conda*», para la gestión de paquetes.

En la realización de este proyecto se han utilizado múltiples paquetes. Entre ellos:

- *matplotlib*: usado para la representación de imágenes.
- *numpy*: usado para facilitar la manipulación de vectores de números.

Adicionalmente, a continuación explico brevemente algunos otros paquetes con mayor influencia en el proyecto.

---

<sup>6</sup>Python es capaz de traducir, o *parsear*, *JSON* a sus propias estructuras sin que tengamos que realizar ningún paso intermedio.

### ***scikit-image***

*scikit-image* nos proporciona un conjunto de herramientas para el procesamiento de imágenes. En las primeras fases, fue utilizada para distintos aspectos de investigación sobre la problemática a acometer, como la segmentación de imágenes. Pero finalmente, se utilizó únicamente para la lectura y guardado de imágenes.

### ***scikit-learn***

*scikit-learn* nos proporciona un conjunto de herramientas para la minería de datos y análisis de estos. En concreto esta se vio utilizada para la creación de clasificadores y para la realización de *clustering*.

### ***Jupyter Notebook***

Es una herramienta que nos permite crear *webs* interactivas con código, texto, representaciones de los datos o distintas visualizaciones, como imágenes. Muchos de los productos generados por este proyecto serán de este tipo, como el etiquetador de imágenes.

## **4.3. Librerías auxiliares**

Para llevar a cabo este proyecto, además de las librerías ya citadas, se han utilizado algunos repositorios auxiliares para llevar a cabo algunas de las tareas. Indico a continuación los repositorios con una breve descripción:

- *Jupyter Dashboards*: es una extensión que nos permitirá mostrar un *Jupyter Notebook* con un estilo más personalizado.
- *IPython File Upload*: es una extensión que nos permitirá subir imágenes desde un *Jupyter Notebook*.
- *darkflow*: es una implementación de *YOLO* utilizado para el reconocimiento automático de fitolitos.



### **Gestor de tareas: *ZenHub***

*ZenHub* es una herramienta para la gestión de proyectos totalmente integrada con *GitHub*. Esta herramienta nos permite obtener diagramas *burn-down*<sup>7</sup> utilizados en el anexo de planificación del proyecto para el seguimiento del proyecto, véase la figura 4.3. También nos permite obtener un gráfico en

---

<sup>7</sup>Un diagrama *burndown* es un gráfico que representa la cantidad de trabajo restante.

el que podemos ver los puntos de historia<sup>8</sup> en los distintos *sprints*<sup>9</sup>. Y, finalmente, nos proporciona un tablero *kanban* para mejorar el flujo de trabajo.



Figura 4.3: Ejemplo de diagrama *burndown* de un *sprint*

#### 4.4. Control de versiones: *Git*

El control de versiones es una parte fundamental para la realización de un proyecto. Este nos permite acometer distintas acciones, como comprobar el cambio realizado entre versiones, trabajar en equipo fácilmente o volver a un punto anterior, entre otras cosas. En nuestro caso, utilizamos *Git* como control de versiones.

*Git* nos proporciona las características que necesitamos. Además de algunas ventajas respecto a otras herramientas, como ser un sistema distribuido o proporcionar una interfaz de comandos muy potente.

#### 4.5. Repositorio: *Github*

Otra herramienta principal, a seleccionar, es el repositorio central, en nuestro caso hemos elegido *Github*. *Github* nos proporciona muchas utilidades y una forma simple de llevar a cabo el proyecto con todas las características necesarias. Es por todas sus características y una previa experiencia con la herramienta por lo que se eligió como nuestro servicio de repositorio central.

<sup>8</sup>Los puntos de historia es una medida del esfuerzo a realizar para completar una tarea.

<sup>9</sup>*Sprint* es un periodo de tiempo en el que el conjunto de las tareas, definidas al principio de este, deben ser finalizadas y revisadas.

## 4.6. Documentación: $\LaTeX$

La memoria y anexos de este proyecto han sido escritos en  $\LaTeX$ . Este nos proporciona muchas ventajas en contraste con otros editores de documentos, como *Word* o sus alternativas, a la hora de realizar un documento de estas características.

$\LaTeX$  nos facilita concentrarnos simplemente en el contenido. Supeditando a este la problemática de como debe formatearse el contenido. Por lo tanto,  $\LaTeX$  automatiza muchas de las típicas tareas que llevaríamos a cabo con otro editor de documentos y, además, permite obtener documentos con una altísima calidad.

## 4.7. Entorno de desarrollo: *JetBrains PyCharm*

El entorno de desarrollo o, por sus siglas *IDE*, es la aplicación utilizada para el desarrollo de la aplicación. Proporcionando múltiples herramientas, desde el auto-completar, hasta *plugins* que permiten comprobar el recubrimiento del código mediante tests, entre otras funcionalidades.

Para esta utilidad se valoraron dos herramientas principalmente: *PyDev* y *JetBrains PyCharm*. Escogiéndose esta última por poseer una mayor cantidad de herramientas integradas, como el control de versiones, distintas opciones para refactorizar el código y un muy buen auto-completar, entre otras características.

## 4.8. Herramienta de prototipado: *NinjaMock*

La herramienta de prototipado es la aplicación utilizada para un primer diseño de la interfaz de una aplicación. Facilitando la demostración, evaluación y agilizando el proceso de llevar a cabo una interfaz. En nuestro caso se utilizó *NinjaMock*, la cual es una aplicación web que nos permite llevar a cabo esta tarea fácilmente.

---

# Aspectos relevantes del desarrollo del proyecto

---



## 5.1. Procesamiento de imágenes

Como primera aproximación al problema que nos concierne, nos hemos enfrentado al procesamiento de imágenes mediante la librería *Scikit-image* de *Python*. Mediante esta herramienta trataremos de dar solución a nuestro problema siguiendo los siguientes pasos:

1. Convertimos la imagen a escala de grises
2. Segmentamos los objetos del fondo de la imagen
3. Obtenemos los distintos objetos de la imagen

### **Convertimos** la imagen a escala de grises

La conversión de la imagen original (RGB) a escala de grises viene motivada con el mero objetivo de poder segmentar los objetos del fondo de la imagen mediante el método de *Thresholding*. Solo pudiéndose partir de una imagen en escala de grises. En la figura 5.4 podemos ver los resultados.

### Segmentamos los objetos del fondo de la imagen

Una vez tenemos la imagen en escala de grises, procedemos a transformar nuestra imagen en una imagen en blanco y negro o binarizada. Los motivos por los que binarizamos la imagen es para obtener una imagen que sea más significativa para nosotros y además este simplificada. Lo cual nos será útil para facilitarnos su procesamiento.

*Scikit-image* nos proporciona distintos métodos mediante los cuales podemos segmentar una imagen. En la figura 5.5 podemos observar el resultado

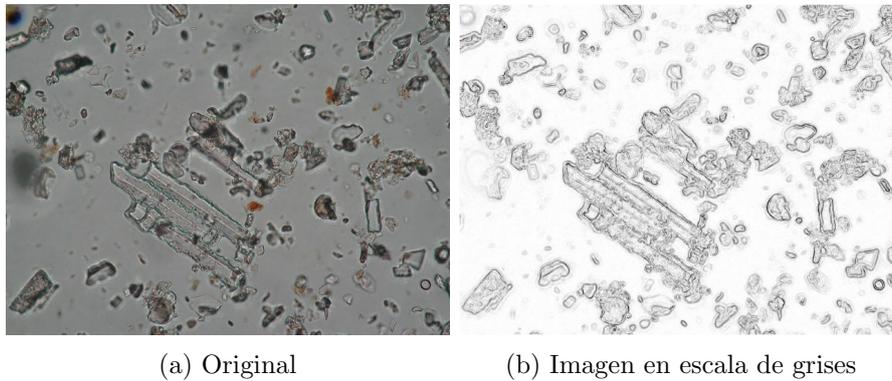


Figura 5.4: Conversión de la imagen original a escala de grises

aplicando distintos métodos, los cuales se van indicando en cada una de las figuras.

## Obtenemos los distintos objetos de la imagen

Después de tener la imagen binarizada de la forma más apropiada posible probamos a segmentar los distintos objetos de nuestra imagen.

## Transformación divisoria

Transformación divisoria, o en inglés *Watershed segmentation*, es un algoritmo clásico para la segmentación de objetos en una imagen.



Durante las primeras pruebas, la segmentación más interesante hasta el momento ha sido la que se muestra en la figura 5.6. A partir de *Watershed segmentation* con marcado, representando cada color un objeto distinto. Más allá de esta segmentación no se ha conseguido nada mejor.

## 5.2. Clasificadores

La aproximación mediante procesamiento de imágenes no parece la más adecuada visto los resultados obtenidos. Por ello vamos a realizar el estudio sobre una segunda aproximación mediante clasificadores, junto a descriptores visuales y la técnica de la ventana deslizante, o en inglés *sliding window*.

*Sliding window* consiste en la subdivisión de una imagen en distintos fragmentos, estableciendo previamente el tamaño de los fragmentos, tanto en ancho como en alto, y el tamaño del salto en cada eje, tras una subdivisión.

Como podemos apreciar en la figura 5.7, primero, la ventana deslizante recorre la imagen en el eje horizontal. Comenzando por la esquina superior izquierda hasta llegar a la esquina superior derecha. Generando una imagen

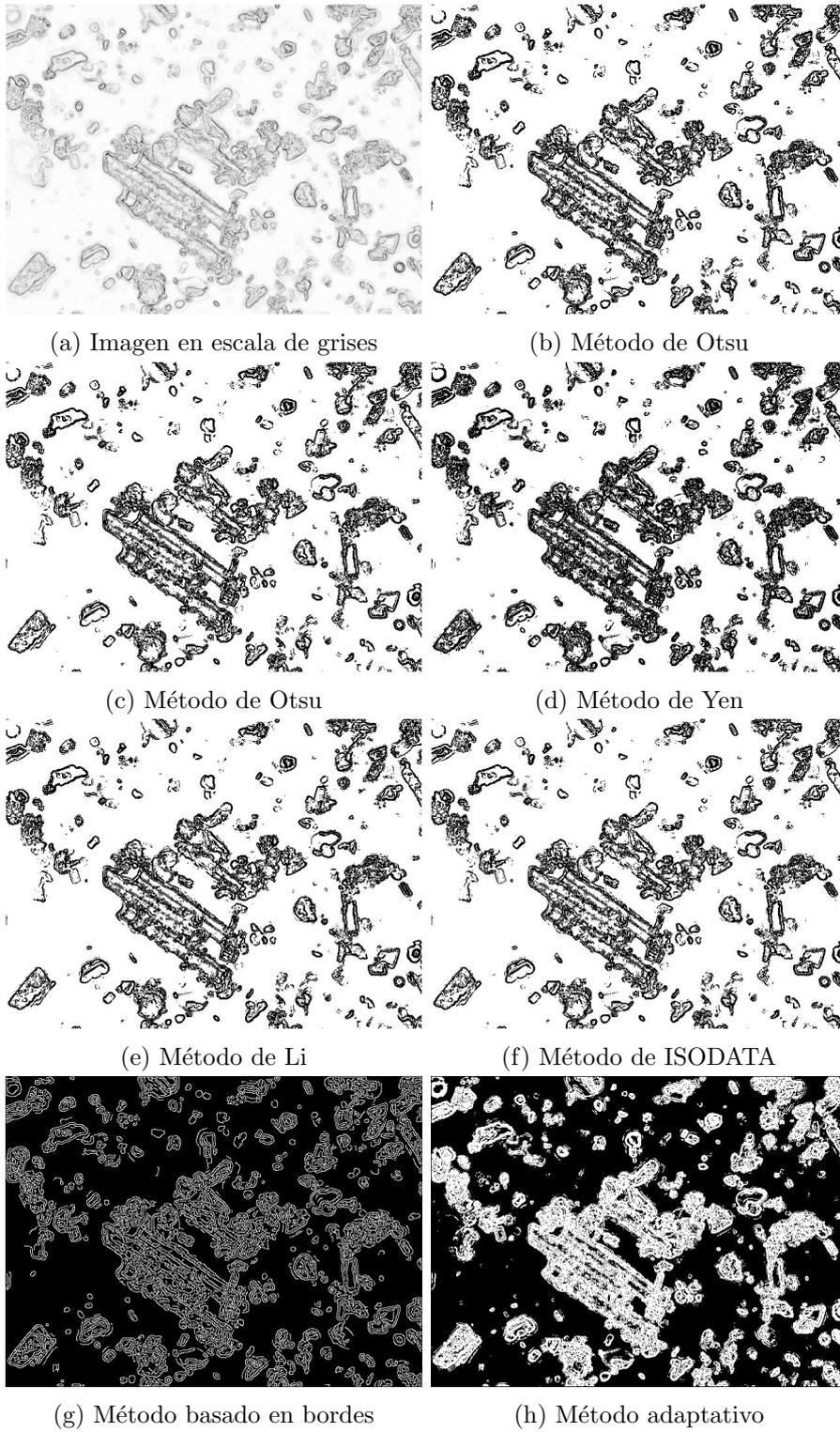


Figura 5.5: Distintos ejemplos de una imagen segmentada

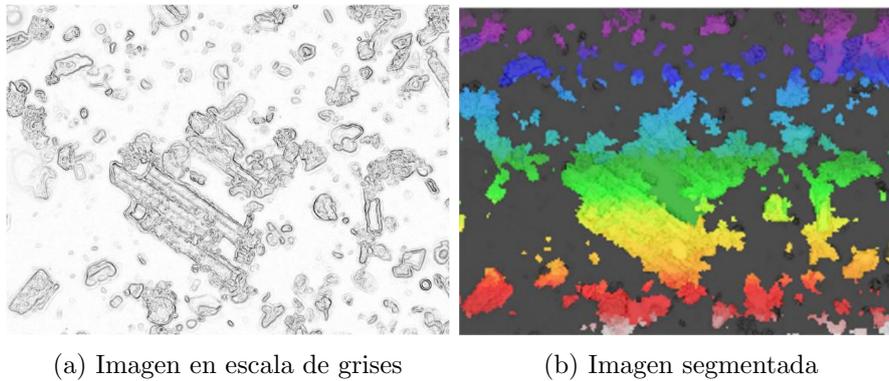


Figura 5.6: Resultados obtenidos mediante transformación divisoria

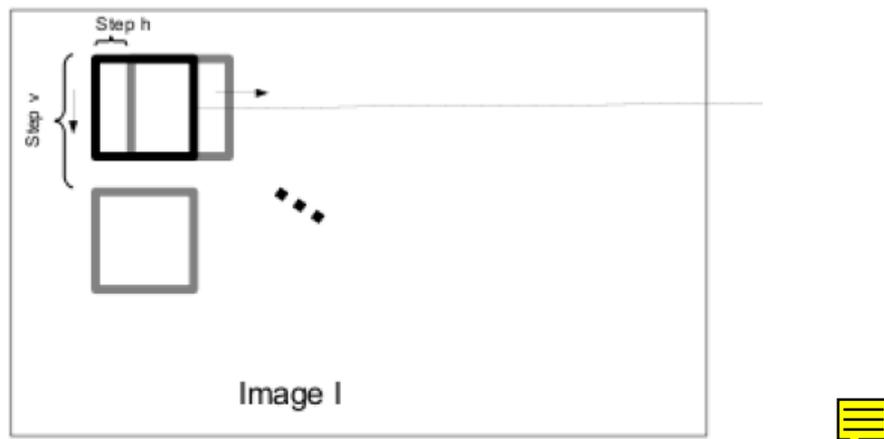


Figura 5.7: Esquema del funcionamiento de la ventana deslizante

por cada subdivisión y realizando saltos horizontales según el tamaño que hayamos definido. Una vez esta llega a la esquina superior derecha, la ventana deslizante lleva a cabo el mismo proceso pero realizando un salto en el eje vertical. Y así sucesivamente hasta completar su recorrido por toda la imagen.

El primer conjunto de técnicas escogidas ha sido una Máquina de Vector Soporte junto al Histograma de los gradientes orientados, la cual es una técnica para la extracción automática de características. Sin embargo, iremos utilizando distintas técnicas para adoptar la que mejor se adapte a nuestra problemática.

El procedimiento para obtener el clasificador es el siguiente:

1. Crear un conjunto de entrenamiento de imágenes de caras que consideramos que son elementos positivos.

2. Crear un conjunto de entrenamiento de imágenes de no-caras que consideramos que son negativos.
3. Extraer las características del conjunto de entrenamiento mediante un descriptor visual.
4. Entrenar<sup>10</sup> el clasificador.

Finalizado el entrenamiento, ya tenemos nuestro clasificador listo para enviarle nuevas imágenes y que sean clasificadas.

### Reconocimiento de imágenes en nuevas imágenes

Para el reconocimiento de objetos en nuevas imágenes, deberemos llevar a cabo los tres siguientes pasos:

1. Dividir la imagen en múltiples fragmentos.
2. Comprobar si cada uno de los fragmentos contiene el objeto.
3. Si existe solapamiento en la detección de objetos, muy común en el uso de este tipo de clasificadores, se deben de combinar dichos solapamientos en uno único.

Cada uno de los fragmentos anteriores se solapa en gran medida. Por lo que origina un problema de sobre-reconocimiento de objetos, reconociendo donde existe un posible positivo, más de uno, en la mayoría de casos. Por ello, se aplica el tercer paso sobre los objetos reconocidos, que es la eliminación del solapamiento de objetos mediante la técnica de *Non-maximum suppression*.

### Aplicación sobre el reconocimiento de caras

Como previa experimentación con esta metodología, vamos a entrenar el clasificador para el reconocimiento de caras. Y en función de la efectividad del método sobre las caras tomaremos una serie de conclusiones sobre las que decidiremos si llevar a cabo esta solución sobre nuestro problema.

Como explicábamos anteriormente, una vez tenemos nuestro clasificador le enviamos una nueva imagen, como podría ser la presentada en la figura 5.8a. A partir de esta imagen el clasificador nos permitirá obtener las ventanas <sup>11</sup> en las que detecta una cara, como vemos en la figura 5.8b. Podemos apreciar que

---

<sup>10</sup>Nos referimos por entrenar, en este ámbito, a enviar al clasificador las distintas imágenes con la clase a la que pertenecen (fitolito de tipo 1, fitolito de tipos 2, etc.).

<sup>11</sup>Se entiende por ventana, en este contexto, a la caja o cuadrilátero que etiqueta un positivo en una imagen.

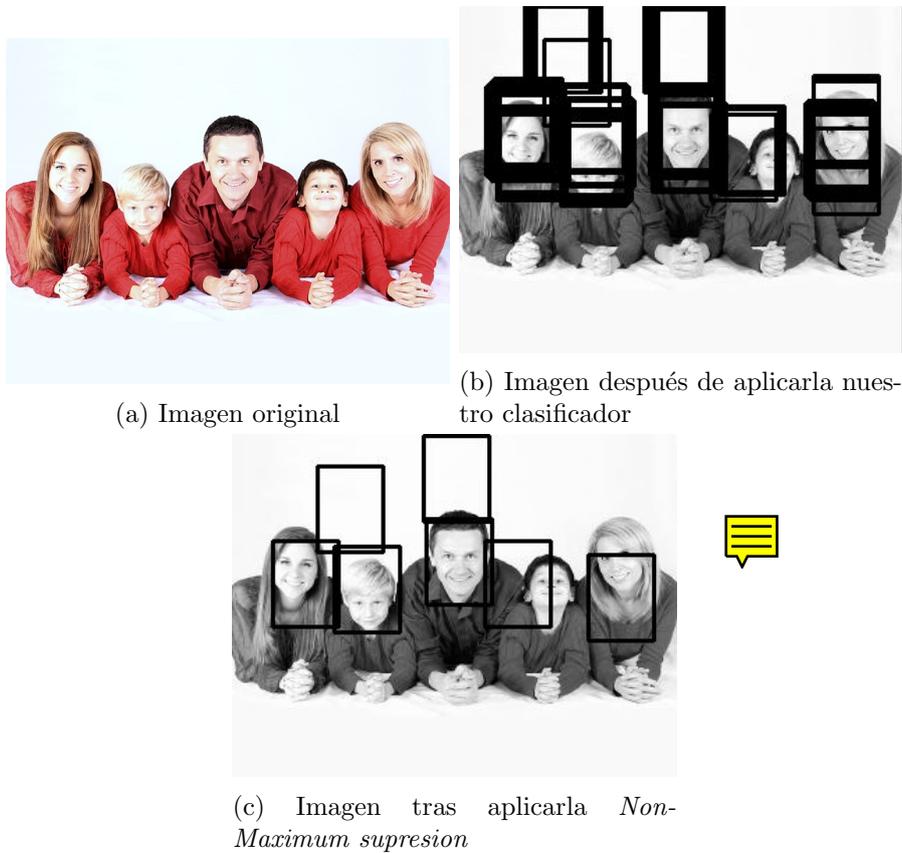


Figura 5.8: Resultados tras aplicar el clasificador sobre una imagen

existe más de una ventana alrededor de cada cara. Y finalmente, tras aplicar el método *Non-Maximum supresion* obtenemos el resultado final mostrado en la figura 5.8c.

### 5.3. Conclusiones

El método utilizado en esta sección se encuentra muy limitado por los siguientes aspectos:

- Es una técnica que presenta problemas de rendimiento. Puesto que requiere de 2 o 3 segundos, como mínimo, para clasificar una nueva imagen.
- Es una técnica que no tiene en cuenta el contexto de la imagen para su clasificación. Sino, que solo tiene en cuenta cada uno de los fragmentos de la imagen de manera aislada al resto. Lo cual dificulta una adecuada precisión del clasificador.

- Comete muchos errores en la detección de falsos positivos. Como se puede observar en la figura 5.8c.
- El tamaño de la subdivisión de la imagen es constante. Por lo tanto, complica las posibilidades en la detección de distintos tamaños de fitolitos.

## 5.4. Problema fundamental: falta de imágenes

Llegados a este punto, y teniendo una primera aproximación, con mayor o menor precisión, de como afrontar el reconocimiento automático de fitolitos, se plantea un problema fundamental en el desarrollo de este proyecto. El problema, al que me refiero, es que no poseemos o conocemos ningún conjunto de entrenamiento de imágenes etiquetadas<sup>12</sup> de fitolitos. Y por lo que hemos podido observar, hasta el momento, es el requisito básico en el aprendizaje supervisado.

Debido a esta problemática, desarrollamos un etiquetador de fitolitos que permita a nuestros usuarios crear un conjunto de imágenes de fitolitos etiquetadas y así, solucionar este problema fundamental en el desarrollo del proyecto.

En concreto, este etiquetador es una aplicación web desarrollada sobre un conjunto de tecnologías *Python* y *JavaScript*. Con el objetivo de poder realizar un futuro despliegue en un servicio *web* que facilite lo máximo las tareas a nuestros usuarios. La información más detallada para un futuro programador, o para el uso por parte del usuario, se encuentra en los anexos del manual del programador y el manual del usuario, respectivamente.



## 5.5. Detección de objetos mediante técnicas de *deep learning*

Una vez observados los resultados mediante una técnica clásica, como es *sliding window*. Vamos a ir un paso más allá, con las técnicas de *deep learning*, las cuales son las que, con diferencia, mayor rendimiento aportan en la actualidad, como previamente he explicado en los conceptos teóricos.

Para ello, vamos a entrenar una implementación de *YOLO* en *Python*, todavía en desarrollo, para tratar de llevar a cabo el detector automático de fitolitos. Aunque, existen otras alternativas que se podrían adoptar en un futuro de no conseguir los resultados esperados con esta implementación.

Previamente, en los conceptos teóricos, se ha explicado que este tipo de modelos también presentan algunas problemáticas, por el volumen de imágenes

---

<sup>12</sup>Me refiero por imágenes etiquetadas, a imágenes con las coordenadas de donde se encuentran los distintos fitolitos en estas.

necesario y el tiempo necesario para ser entrenados, principalmente. Existen soluciones, como *data augmentation*, transferencia de conocimiento o partir de modelos previamente entrenados. Que iremos valorando en este desarrollo.

---

## Trabajos relacionados

---

Actualmente, no existe ningún sistema popular que intente solucionar el problema acometido en este trabajo. Pero si existen trabajos similares en los que se estudian posibles técnicas para abordar la realización de sistemas automáticos para materiales microscópicos [1].

En el caso del artículo *Automatic recognition of complete palynomorphs in digital images* [1], escrito por J.J. Charles, se crea un sistema consistente en las siguientes tres fases:



1. Preprocesar la imagen, segmentando la parte de atrás de la imagen de la parte de delante.
2. Segmentar las distintas regiones de la imagen.
3. Clasifica las regiones de la imagen.

Esta aproximación genera muy buenos resultados para la aplicación estudiada, pero presenta varios problemas para nuestro caso. Los problemas a los que me refiero son los siguientes:

- Es un sistema aplicado a una única clase de objetos. En nuestro caso, queremos que sea un sistema escalable debido a la gran variedad de tipos de fitolito.
- No existen grandes diferencias en los tamaños de palinofaceos. Fue entrenado para objetos de tamaño 30, 50 y 70. Pero en nuestro caso los fitolitos son de distintos tamaños, estrechos y altos, o viceversa. Es decir, teniendo tamaños muy variados y siendo tridimensionales, por lo cual las formas de los materiales varían sustancialmente. Lo cual complica nuestro proyecto un gran paso más allá.

- En ningún momento se hace referencia al tiempo necesario para clasificar una nueva imagen. Pero estas técnicas suelen ser bastante ineficientes, debido al gran espacio de exploración que plantean.

Por lo tanto, las técnicas de *deep learning* actuales presentan grandes mejoras en cuanto a eficiencia, escalabilidad, y flexibilidad en cuanto al aprendizaje de nuevos tipos de fitolitos, en nuestro caso. Aunque también presentan otros problemas a cambio de sus grandes posibilidades, como he comentado previamente en otras secciones.

---

# Conclusiones y Líneas de trabajo futuras

---

## 7.1. Conclusiones

## 7.2. Lineas futuras de trabajo

Los fitolitos poseen una complejidad enorme por la multitud de formas de estos, incluso perteneciendo a un mismo tipo, y la multitud de tipos de fitolitos existentes. Por lo tanto, existe un gran margen de mejora en el reconocimiento automático de estos.

El problema fundamental, junto al ya expuesto sobre la complejidad de las formas de los fitolitos, es la inexistencia de un conjunto de imágenes etiquetadas de fitolitos. Actualmente, las técnicas más avanzadas para el reconocimiento de objetos necesitan de conjuntos de entrenamiento muy grandes. Por ello, en futuros desarrollos se podrían mejorar las técnicas de *data augmentation* e incorporar nuevas técnicas con el objetivo de obtener un conjunto de entrenamiento sustancialmente mayor.

Una de las nuevas técnicas a las que me refiero es el aprendizaje mediante modelos 3D, el cual nos permitiría crear un conjunto de imágenes significativamente mayor [6] [7]. Estas técnicas consiguen recrear un modelo 3D de un objeto a partir de diferentes imágenes. Y a partir del modelo generado obtener imágenes desde las distintas perspectivas.

Otra posible mejora sería la realización del entrenamiento de *YOLO* en un entorno con mayores recursos, sobre todo con la utilización de una mejor tarjeta gráfica, como la *Nvidia Titan X*. O incluso con la utilización de varias tarjetas gráficas concurrentemente, lo cual aceleraría el proceso de entrenamiento significativamente.

---

## Bibliografía

---

- [1] J.J. Charles. Automatic recognition of complete palynomorphs in digital images. 2009. [Online; accedido 7-Mayo-2017].
- [2] Jack Copeland. What is artificial intelligence?, 2000. [Online; accedido 16-Marzo-2017].
- [3] Universidad Complutense de Madrid. Arqueobotánica, 2016. [Online; accedido 7-Mayo-2017].
- [4] Li Deng and Dong Yu. *Deep Learning: Methods and Applications*. 2013. [Online; accedido 3-Mayo-2017].
- [5] JSON. Json, 2017. [Online; accedido 1-Mayo-2017].
- [6] Fred Nicolls. Structure and motion from sem: a case studysem: a case study. 2016. [Online; accedido 27-Mayo-2017].
- [7] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Exploring invariances in deep convolutional neural networks using synthetic images. 2014. [Online; accedido 27-Mayo-2017].
- [8] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 2015. [Online; accedido 28-Abril-2017].
- [9] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. 2016. [Online; accedido 2-Mayo-2017].
- [10] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. 2015. [Online; accedido 28-Mayo-2017].
- [11] Stuart Russell and Peter Norvig. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, page 5, 1995.

- [12] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. *scikit-image: image processing in Python*. 2014. [Online; accedido 28-Mayo-2017].
- [13] w3schools. *Javascript tutorial* — w3schools, 2017. [Online; accedido 1-Mayo-2017].
- [14] Wikibooks. *Python programming* — wikibooks, the free textbook project, 2017. [Online; accedido 1-Mayo-2017].
- [15] Wikipedia. *Visual descriptor* — Wikipedia, the free encyclopedia, 2016. [Online; accedido 22-Febrero-2017].
- [16] Wikipedia. *Bag-of-words model* — wikipedia, the free encyclopedia, 2017. [Online; accedido 2-Mayo-2017].
- [17] Wikipedia. *Histogram of oriented gradients* — Wikipedia, the free encyclopedia, 2017. [Online; accedido 22-Febrero-2017].
- [18] Wikipedia. *Image segmentation* — Wikipedia, the free encyclopedia, 2017. [Online; accedido 10-Febrero-2017].
- [19] Wikipedia. *Machine learning* — Wikipedia, the free encyclopedia, 2017. [Online; accedido 9-Abril-2017].
- [20] Wikipedia. *Otsu's method* — Wikipedia, the free encyclopedia, 2017. [Online; accedido 8-Febrero-2017].
- [21] Wikipedia. *Pattern recognition* — Wikipedia, the free encyclopedia, 2017. [Online; accedido 9-Abril-2017].
- [22] Wikipedia. *Supervised learning* — Wikipedia, the free encyclopedia, 2017. [Online; accedido 9-Abril-2017].
- [23] Wikipedia. *Unsupervised learning* — Wikipedia, the free encyclopedia, 2017. [Online; accedido 9-Abril-2017].
- [24] Yoshua Bengio Yann LeCun and Geoffrey Hinton. *Deep learning*. 2015. [Online; accedido 28-Mayo-2017].