UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del grado en Ingeniería Informática

Sistema de Etiquetado de Fitolitos Online

Presentado por Marta Monje Blanco
en Universidad de Burgos — 17 de septiembre de 2018
Tutores: Dr. Álvar Arnaiz González y Dr. José Francisco Díez Pastor

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

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

Exponen:

Que la alumna D.ª Marta Monje Blanco, con DNI 71312200X, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

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, 17 de septiembre de 2018

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
Dr. Álvar Arnaiz González	Dr. José Francisco Díez Pastor

Resumen

La arqueobotánica es la sub-rama de la arqueología que estudia la relación entre plantas y humanos en el pasado mediante el análisis de restos vegetales hayados en un contexto arqueológico. Su objetivo el es de dar a conocer qué técnicas de agricultura y de cosecha se usaban, en qué condiciones paleoambientales subsistían y la finalidad para las que las cultivaban (fines médicos, alimenticios...) en función de la antigüedad y el estado de las muestras.

En este proyecto se va a trabajar con la recogida de datos y en el prototipado de modelos para la futura detección y clasificación de micro-restos, más concretamente de fitolitos. Los fitolitos son células de origen vegetal, las cuales han sufrido un proceso de biomineralización debido a la actividad metabólica.

La finalidad de este proyecto es la de proporcionar un conjunto de herramientas y de prototipos que permitan la futura integración de un sistema de clasificación automática de fitolitos, clasificándolos en siete clases diferentes.

Con el fin de facilitar la recogida de datos para el entrenamiento del modelo se ha utilizado un sistema de cliente servidor en el cual se proporciona un etiquetador especializado con el cual los expertos podrán subir sus propias imágenes y eiquetarlas. Una vez se tuviera un conjunto razonablemente grande, se emplearía para entrenar el clasificador.

Actualmente dicho servicio está desplegado en un servidor para ofrecer una mejor comunicación entre los expertos en arqueobotánica y la gestión del entrenamiento del clasificador.

Descriptores

Servidor web, Reconocimiento y clasificación de imágenes, Arqueobotánica, Fitolito, Inteligencia artificial, tratamiento y recogida de datos.

Abstract

Archaeobotany is the sub-branch of archeology that studies the relation between plants and humans in the past by analyzing plant residues in an archaelogical context. Its aim is to reveal what agriculture and harvesting techniques were used, in what paleoenvironmental conditions subsisted and how they were used based on the age of the samples.

In this project we will work with data collection and prototyping of models for the future detection and classification of micro-residues, more specifically phytoliths. Phytoliths are cells of plant origin which have undergone a process of biomineralization due to metabolic activity.

The objective of this project is to approach the classification of different photographic samples of micro-residues taken in the laboratory, trying to recognize phytoliths and classifying them into seven different classes.

In order to facilitate the collection of training data of the model, a client-server system has been developed in which a specialized labelling tool is provided with which experts can upload their own images and label them. Once a reasonably large set was available, it would be used to train the classifier.

Currently, this service is deployed on a server to offer better communication among experts in archaeobotany and the management of classifier training.

Keywords

Web server, Images recognition and classification, Archaeobotany, Phytolite, Artificial intelligence, treatment and data collection.

Índice general

Índice de figuras

Índice de tablas

Introducción

Los fitolitos son un tipo de células vegetales cuyos procesos metabólicos han creado estructuras que se han fosilizado. Este estado de fosilización permite que se pueda extraer mucha información de ellas.

El proceso de identificación de fitolitos es compleja y requiere de expertos que la lleven a cabo.

Los fitolitos son células tridimensionales, que al ser tratadas como imágenes de dos dimensiones pueden variar la forma en la que los percibimos. Esta ambigüedad hace que el coste de tener a un ser humano etiquetando no sea beneficioso.

Hay ocasiones en las que se necesita precisión y los sistemas de detección automáticos. Actualmente se utilizan este tipo de sistemas para diagnóstico médico, detección de alimentos en mal estado, piezas defectuosas en una cadena de producción.

En este proyecto se abordan dos frentes diferenciados: La recogida e interpretación de datos y el despliegue de esta herramienta en un servidor de producción.

- **Etiquetador:** se trata de una herramienta especializada mediante la cual se pueden etiquetar siete tipos predefinidos de fitolitos. Su finalidad es la de facilitar la recogida de datos de entrenamiento, haciéndola más transparente para los expertos que van a generar el conjunto de datos de entrenamiento y más accesible para recoger dichos datos y manipularlos posteriormente.

- **Despliegue:** Las herramientas que se creen en este proyecto serán

desplegadas en un servidor de producción[1]. De esta forma hacemos que la aplicación sea más accesible para los usuarios evitando instalaciones intermedias además de facilitar el paso de la información desde el etiquetador a la posterior interpretación y uso de los datos.

Parte del objetivo de este proyecto era la de crear un clasificador que detectase los fitolitos de forma automática y que los clasificase según su tipo, aunque por diversas razones, explicadas en el apartado de 4.3, no ha sido posible.

La idea principal de este proyecto empieza en el etiquetado. El etiquetador genera conjuntos de etiquetas seguros gracias a la limitación en la nomenclatura de las etiquetas. Estas etiquetas, junto a las imágenes que son usadas para el etiquetado, son recogidos por el servidor y se usan para generar la información necesaria para el entrenamiento de un modelo.

[1]Enlace de acceso: http://martamonjeblanco.pythonanywhere.com/

Objetivos del proyecto

En este apartado se van a presentar los objetivos del proyecto, tanto a nivel global, como a nivel técnico.

2.1. Objetivos globales

Como conjunto de proyecto se presentan los siguientes objetivos:

- Crear un etiquetador que genere un conjunto de entrenamiento recogiendo la información necesaria y estructurándola de forma funcional para su posterior lectura, interpretación y manipulación por parte de un modelo.
- Desplegar la aplicación en un servidor de producción desde el que se pueda trabajar cómodamente y generando el mínimo conflicto entre los usuarios que etiquetan y los que entrenan el modelo. Se realizará un estudio comparativo entre las distintas opciones de despliegue que han sido probadas.
- Conocer y comprender el funcionamiento de librerías para realizar el aprendizaje automático como *Tensorflow* y su relación con CUDA.
- Probar y estudiar diferentes prototipos para la futura detección y clasificación de fitolitos.

2.2. Objetivos técnicos

El uso de distintos elementos técnicos tiene como objetivo dentro de este proyecto los siguientes puntos:

- Utilizar *HTML* y *Javascript* para elaborar la parte del cliente, en este caso, los elementos que conforman el entorno web del proyecto.
- El uso de *Python* y de *Flask* para operar en la parte del servidor. *Flask* va a actuar como un intermediario entre las partes de la lógica escrita en *Python* y la parte del cliente explicada en el punto anterior.
- Generar un control de acceso de usuarios a la aplicación mediante un login. Se utilizará un acceso gestionado por Google por medio de su API [2].
- Despliegue en *Pythonanywhere*, para el cual se ha necesita conocimientos de *git*. Este tipo de servicio solo permite hostear aplicaciones escritas en *Python*.
- Proporcionar un servidor seguro que permita una interacción transparente por parte del usuario. Básicamente que el usuario no tenga que instalarse. Que todo los que se haga.
- Uso de control de versiones mediante *git* y y un servicio central, en este caso *GitHub*.
- Se va a seguir la metodología de Scrum, haciendo reuniones semanales correspondiente a los sprints que se planteen.
- La organización de las *issues* dentro del proyecto se hará mediante *ZenHub* que nos proporcionará un tablero para ordenar las actividades en función de su estado y prioridad.

[2]Enlace a APIs y servicios de *Google*: https://console.developers.google.com/apis/dashboard?project=corded-cortex-201715

Conceptos teóricos

Para la comprensión del proyecto es necesario introducir aquellos conceptos inherentes al desarrollo, como el tipo de arquitectura cliente servidor y aquellos términos referentes a la inteligencia artificial y por otro lado, a la arqueobotánica y los fitolitos.

3.1. Arqueobotánica y Fitolitos

Como ya se explicó en la introducción, la arqueobotánica [10] es una rama de la arqueología que se encarga de estudiar la relación que existía entre las plantas y los humanos en diferentes momentos de la historia. Pretende conocer qué técnicas se usaban para el cultivo, recolección, y que usos le daban después, si lo usaban de alimento, para fines médicos y curativos, para rituales y ofrendas...

Toda esta información se extrae de restos de plantas hallados en contextos arqueológicos y pueden enriquecer nuestro conocimiento sobre culturas antiguas.

Fitolitos

Los fitolitos [14] forman parte de los restos de plantas que son analizados, más concretamente pertenecen al tipo de micro-restos, es decir, aquellos que solo pueden ser apreciados con la ayuda de un microscopio [7].

Se trata de células de origen vegetal que han sufrido una biomineralización debido a procesos propios del ciclo de vida de estas. Gracias a las características físicas de la célula, este proceso de micro-fosilización es fácil

de lograr y se conserva muy bien a lo largo del tiempo, además se produce en abundancia y existe una gran variedad.

Existen muchas formas de clasificar fitolitos, pero la que proporciona más uniformidad es la que lo hace basándose en su morfología.

Dependiendo de en qué zona se hallen o a qué especie de planta pertenezcan, los fitolitos pueden adquirir distintas formas, colores o tamaños debido a la diversidad vegetal propia de la zona.

Para simplificar el proyecto, tanto al nivel del etiquetador como el planteamiento del clasificador, vamos a distinguir entre siete de las formas más comunes que adoptan los fitolitos:

1. *Bilobate*
2. *Bulliform*
3. *Cyperaceae*
4. *Rondel*
5. *Saddle*
6. *Spherical*
7. *Trichomas*

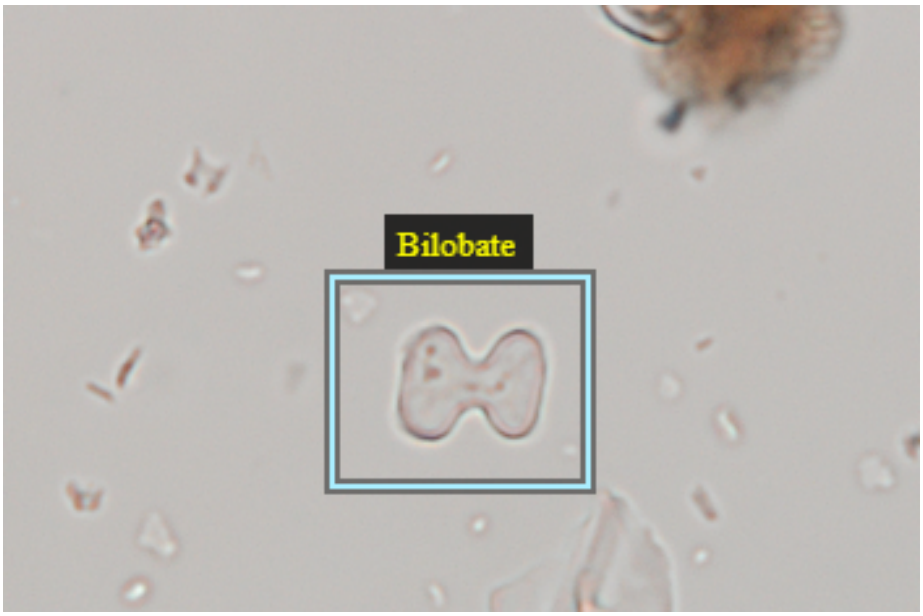

Figura 3.1: Ejemplo de Bilobate etiquetado mediante el etiquetador

Aunque hay más tipos de fitolitos, las formas que presentan los que hemos presentado son lo suficientemente diferentes entre sí como para que la tarea del etiquetado no sea excesivamente complicada si se tienen los conocimientos necesarios para identificarlos, aun así se requieren de expertos para ello.

En muchas ocasiones las imágenes con los fitolitos vienen con un exceso de *ruido* de fondo como se puede apreciar en la imagen 3.2. Esto quiere decir que en las imágenes que se van a usar para el etiquetado no solo hay fitolitos, hay más elementos microscópicos que pueden obstaculizar la tarea del etiquetado.

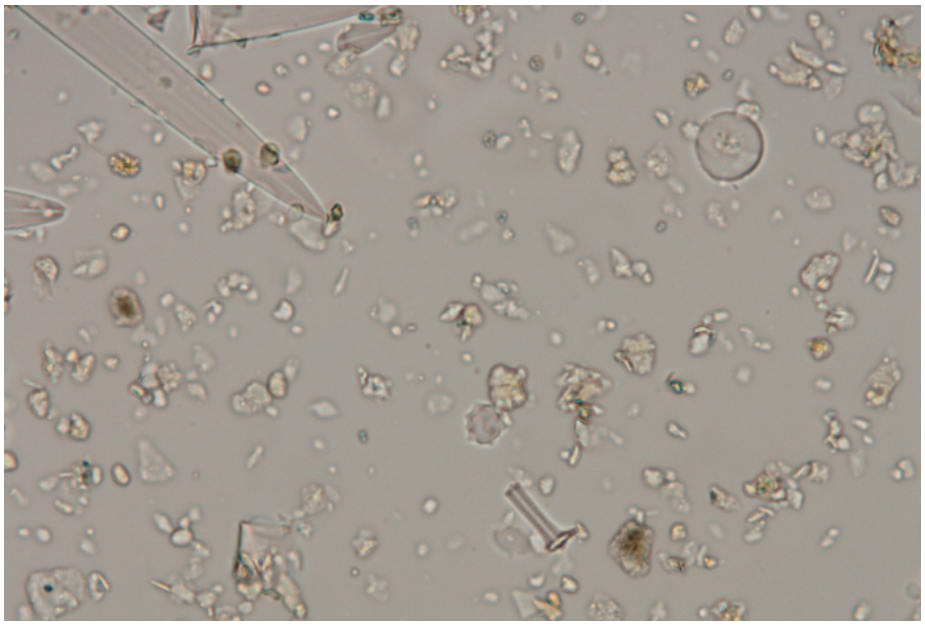

Figura 3.2: Imagen que presenta mucho ruido, es decir, elementos que no son fitolitos y pueden llevar a error.

3.2. Arquitectura cliente-servidor

La aplicación está alojada en un servidor para que se pueda trabajar con ella de forma más sencilla.

La arquitectura que sigue es la de cliente-servidor (Ver imagen 3.3). Se trata de una arquitectura software que separa los recursos y/o servicios que se van a proporcionar desde el servidor, de los clientes que van a hacer uso de la aplicación bajo demanda [13].

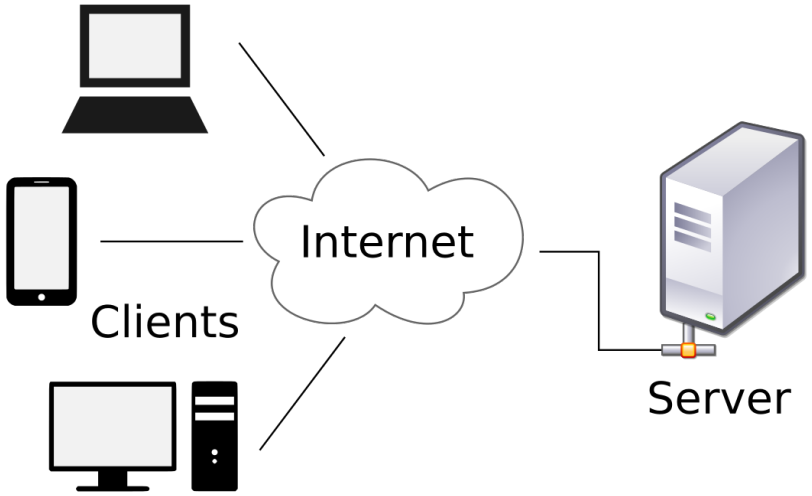

Figura 3.3: Esquema básico de la arquitectura cliente-servidor.

Imagen extraída de https://es.wikipedia.org/wiki/Cliente-servidor

La aplicación se ha desplegado en un Pythonanywhere, un servidor especializado en aplicaciones escritas en *Python* [16].

En este servidor se alojará la aplicación y las imágenes con las que se desee trabajar dentro de ella. Hablaremos más adelante de esta herramienta y de por qué se escogió trabajar con ella en el capítulo de Técnicas y herramientas 3.6.

3.3. Inteligencia Artificial (AI)

La Inteligencia artificial se define como la técnica de simulación de los procesos cognitivos propios de los humanos reproducidos en máquinas. Hacen uso del aprendizaje y la autocorreción para emular el razonamiento tal y como lo entendemos y resolver problemas [11].

Actualmente la definición de inteligencia artificial cobra diferentes significados en función del autor que la defina. Si bien por una parte se busca lograr que una máquina razone o resuelva problemas como un ser humano, en otras ocasiones lo que se busca es que las capacidades cognitivas desarrolladas por la máquina sean superiores a las de un ser humano.

En el caso de esta aplicación la idea era que la el mediante un modelo se

pudiesen **reconocer objetos** dentro de una imagen, en este caso de fitolitos y devolver la clase a la que pertenecen junto a un porcentaje de confianza.

El **porcentaje de confianza** es un indicador de la precisión de la predicción que ha realizado el modelo. Por ejemplo el modelo puede etiquetar un bilobate al 98 % de confianza, lo que nos indica como de seguro es el clasificador cuando hace una predicción.

3.4. Detección y clasificación de objetos

La **detección de objetos** dentro de una imagen consiste en localizar las coordenadas en donde se encuentra cualquiera de los tipos de objetos que se estén tratando de extraer. Esto llevado a la aplicación, consistiría en identificar y señalar todos los fitolitos, independientemente del tipo, y devolver las coordenadas de estos.

Por otra parte, la **clasificación de objetos** se encarga de identificar qué tipo de objetos, en este caso que tipo de fitolito se a detectado. Como anteriormente ya se había comentado, dentro de la clasificación es donde obtendremos el porcentaje de confianza de la etiqueta.

3.5. Redes convolucionales

Las redes convolucionales son un tipo de redes neuronales artificiales que emulan el comportamiento de las neuronas de la corteza visual primaria en un cerebro biológico [2] (Ver imagen 3.4).

La diferencia entre una red neuronal ordinaria y una convolucional, es que esta última está diseñada para trabajar con imágenes.

¿Por qué redes convolucionales y no convencionales?

El objetivo era el de reconocer y clasificar objetos dentro de imágenes de una complejidad razonablemente alta. Se trata de imágenes grandes y tienen mucha definición.

Las redes neuronales convencionales pueden trabajar con imágenes, pero no escalan bien cuando las imágenes son más grandes o de una resolución mayor.

Las redes convolucionales [5] trabajan extrayendo poco a poco la información en sus capas superiores, va creando patrones que poco a poco se van

simplificando y en las capas más profundas se intenta que los patrones que la red haya deducido coincidan con la foto original.

Tendremos acceso a este tipo de redes gracias a Tensorflow, del cual se hablará en el capítulo de técnicas y herramientas 3.6.

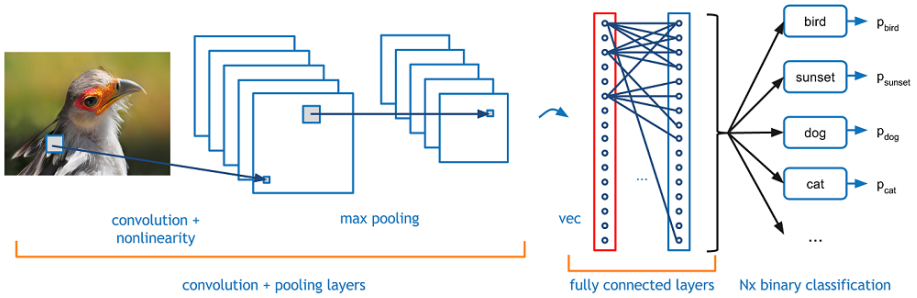

Figura 3.4: Esquema de funcionamiento de una red neuronal convolucional en la extracción de información para formar patrones.

Imagen extraída de: https://goo.gl/bJfXr5

3.6. *mAP* en detección de objetos

mAp, cuyas siglas en inglés significan *mean Average Precision*, es la medida del promedio de la precisión de un modelo.

La **precisión** es el porcentaje de predicciones positivas que son correctamente clasificadas con un modelo.

Se calcula haciendo la media entre los valores de precisión tomados para cada consulta.

Tenemos que tener en cuenta este valor a la hora de escoger el modelo con el que se va a entrenar.Los modelos más rápidos o que menos recursos necesitan son los que suelen tener este valor más bajo.

A menudo para tener una precisión alta hay que sacrificar tiempo, es decir, el modelo va a tardar más en entrenar.

Uno de los modelos con los que se ha intentado construir el clasificador se llama *ssd mobilenet v1 coco* el cual tiene un tiempo bajo en comparación con otros modelos más precisos pero por contra su precisión de menor.

Técnicas y herramientas

A lo largo del desarrollo del proyecto se han hecho uso de técnicas y de herramientas que han ayudado en las tareas de desarrollo y diseño de la aplicación.

4.1. Etiquetador

Para poder generar el conjunto de entrenamiento necesitamos un etiquetador que nos proporcione una salida interpretable. Preferiblemente se quería que el formato de salida fuese legible para los humanos.

Al principio del proyecto se pensó en encontrar un etiquetador que estuviese a parte del proyecto, es decir, que fuese una aplicación distinta y que el usuario que quisiera etiquetar imágenes usase dicha aplicación, para que más adelante pudiésemos entrenar el modelo y que este fuese lo único que hubiese en el servidor.

Más tarde se optó por embeber un etiquetador dentro de la aplicación para facilitar al usuario el etiquetado y evitar posibles problemas con la posterior lectura.

Previamente a la elección del etiquetador que se ba a usar se exploraron tras opciones:

- **LabelID**[3]:Es una aplicación de escritorio. En un principio parecía atractiva, pero debido a la poca documentación que tenía y a la forma en la que recogía los datos, se descartó.

[3]Enlace a la páina de LabelID: https://sweppner.github.io/labeld/

- **LabelImg:** [4] Personalmente, a pesar de no haber escogido este, es probablemente el que mejor devuelva los datos. Si bien es cierto que es difícil de integrar en el proyecto, Hay mucha documentación a cerca de como entrenar modelos de Tensorflow con este etiquetador, lo cual es un punto a favor porque no tienes que manipular los datos de salida demasiado, asi que es más difícil corromperlos.

- **VGG Image Annotator (VIA):**[5] este etiquetador es de software libre, por lo que es fácil de integrar y de modificar dentro del proyecto. Devuelve los datos en *.csv* y en *.json* y la recogida de estos se puede manipular para amoldarla a la que necesite el modelo para ser entrenado.

  Al final fue la herramienta que se escogió y en base a la que se ha construido el etiquetador (Ver imagen 4.5). El resultado final del etiquetador el funcional y seguro para que no se manipulen los daos de forma externa.

Figura 4.5: Vista del etiquetador una vez a sido integrado en el proyecto.

[4] Enlace a la página de LabelImg: https://github.com/tzutalin/labelImg

[5] Enlace para acceder a VGG Image Annotator (VIA)http://www.robots.ox.ac.uk/%7Evgg/software/via/

4.2. Despliegue

Framework : *Flask*

Para desarrollar una aplicación web necesitamos antes de nada un framework que nos provea de las utilidades propias y un esquema de trabajo. Existen diferentes alternativas, en este caso desarrolladas sobre *Python*, como podrían ser *Flask*, *Pyramid* o *Django*. Fuera de contexto ninguna es mejor que otra, pero desde el punto de vista de la aplicación y el mío propio es adecuado hacer un pequeño análisis de elección.

La decisión está entre *Django* y *Flask*, por la documentación y los tutoriales existentes, no obstante, aunque *Django* sea más completo, también es menos flexible. Por otro lado *Flask* te permite instalar diferentes utilidades a medida que las vayas necesitando, además de que aunque tiene una documentación más reducida que *Django*, esto la hace más práctica y útil dado que es lo suficientemente completa.

Flask también es más rápida que *Django* y más sencilla de aprender, por lo tanto emplearemos *Flask*[6].

Nanobox

Nanobox[7] es un servicio que emplea Virtual Box y Docker para crear entornos de desarrollo virtuales dentro de la máquina local. Se configura a través de un fichero yaml llamado *boxfile.yml*, en el cual especificaremos la configuración de los recursos que vayamos a necesitar tanto localmente como en producción.

Tiene una interfaz muy amigable pero la instalación de las dependencias es bastante pesada. Permite desplegar nuevas versiones de forma muy sencilla y transparente para el programador. Tiene muchas funcionalidades de control de memoria, consola para monitorizar los procesos y aunque es de pago, con el paquete de estudiantes se puede probar gratis durante unos meses.

Se descartó porque no me permitía acceder a los permisos de las carpetas que creaba en el servidor, más adelante, hablaremos de ello y de las soluciones que se dieron.

[6] Ver documentación de Flask : http://flask.pocoo.org/
[7] Documentación de Nanobox : https://docs.nanobox.io/

Pythonanywhere

Pythonanywhere [8] es un servidor web, dedicado exclusiamente a alojar aplicaciones escritas en *Python*. Su interfaz es bastante fácil de entender. Es exclusiva para *Python*, por lo que resulta muy práctico ya que *Flask* es *Python*.

Este hosting nos ofrece dos consolas de *Python* con las que poder interactuar con el proyecto. Se necesita tener el proyecto en un repositorio público y hay que utilizar git para llevarse dicho proyecto al servidor. Una vez clonado el proyecto solo habrá que sincronizarlo y relanzarlo cada vez que queramos un cambio en la aplicación.

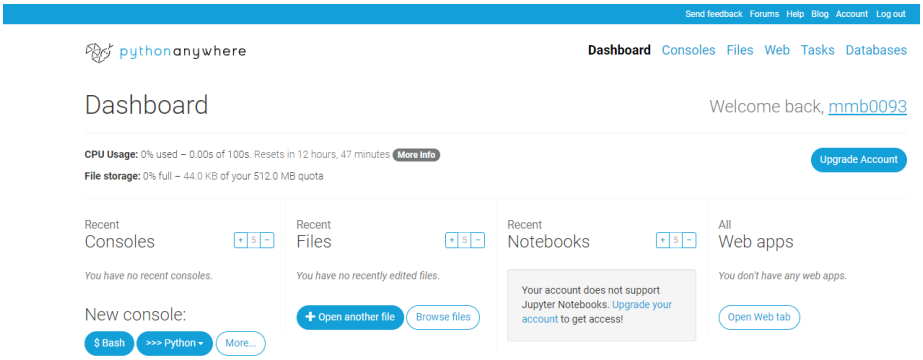

Figura 4.6: Interfaz de PYthonanywhere

Comparación entre *Pythonanywhere* y *Nanobox*

En este apartado se va a hacer una comparativa entre *Pythonanywhere* y *Nanobox* con el fin de argumentar mi decisión final de decantarme por *Pythonanywhere*:

- **Interfaz:** Ambas son bastante intuitivas, pero la de *Nanobox* tiene una interfaz más limpia y bonita a pesar de tener muchas más funcionalidades.
- **Precio:** *Nanobox* es de pago y es caro y la preba gratis dura poco. *Pythonanywhere* es gratuito para aplicaciones de poco tránsito.
- **Documentación:** *Nanobox* tiene una extensa documentación plagada de ejemplos y tutoriales muy fáciles de seguir. *Pythonanywhere* tiene

[8] Página principal de Pythonanywhere: https://www.pythonanywhere.com/

mucha documentación propia pero su verdadero valor reside en la comunidad que lo usa.

- **Información externa a la aplicación:** *Nonobox* no tiene a penas tutoriales externos ni una comunidad que le de un mínimo soporte y por otra parte *Pythonanywhere* tiene una extensa comunidad que comenta dudas, hace tutoriales y por lo general mantienen con vida estas prácticas.
- **Funcionamiento:** Con*Pythonanywhere* no tuve a penas problemas, y los que tuve fueron triviales y de fácil solución. Por otra parte *Nanobox* no llegó a funcionarme completamente del todo.

Mirando los puntos expuestos de forma global, el único que importa es el de **funcionamiento**, los demás son en mayor o menor medida complementarios.

Por esta razón se ha escogido *Pythonanywhere*.

4.3. Lenguaje de programación

Python

Se ha usado *Python* en su versión más reciente, la 3.7.0 [1].

Es un lenguaje muy extendido por ser de código abierto, fácil de leer e interpretar. Es de propósito general, lo que significa que puede abordar muchos frentes en el desarrollo. Es un lenguaje interpretado, multiparadigma (orientación a objetos, programación funcional y programación imperativa), es de tipado dinámico y es independiente de la plataforma en la que se ejecute.

Presenta múltiples ventajas frente a otros lenguajes, entre ellos la gran cantidad de librerías de licencia gratuita que tiene, las funcionalidades que ofrece y la versatilidad de sus funcionalidades.

En este caso es interesante para el proyecto por dos aspectos:

- El despliegue de la aplicación, para el que utilizaremos *Flask* como framework
- *Tensorflow*[9] y lectura de datos. *Python* facilita la lectura de archivos con los que vamos a trabajar en este caso: csv y json, los cuales son

[9]Enlace para el repositorio de Tensorflow: https://github.com/tensorflow

usados frecuentemente con *Tensorflow*, la librería que se va a usar para tratar de entrenar un modelo.

Anaconda

Anaconda es una distribución de licencia libre de *Python* y R [3].

Las ventajas del uso de *Anaconda* empiezan desde la gestión de paquetes y librerías para *Python*, gracias a su sistema de administración de paquetes **Conda** [10] lo cual nos facilitará mucho usar librerías enfocadas al procesamiento de datos y en el aprendizaje automático, como Tensorflow, Scikit-team y SciPy.

Desarrollo web: *HTML y bootstrap*

HTML es un lenguaje de marcado, cuyas siglas en inglés son *HyperText Markup Language* [15], es decir, lenguaje de marcas de hypertexto.

Se utiliza para confeccionar paginas web. Es un lenguaje basado en la diferenciación, es decir, separar aquellos elementos externos (*Javasecript, CSS*, imágenes, enlaces...) llamándolos a través de referencias a su ubicación.

Con *HTML* se busca que una página escrita en este lenguaje tenga el mismo aspecto y funcionalidad en cualquier buscador en el que se ejecute.

Por otro lado **Bootstrap**, es un framework web multiplataforma que contiene plantillas para diseñar aplicaciones más funcionales de forma sencilla [12].

Gracias a estas librerías la página se verá más limpia y profesional sin la necesidad de añadir muchas líneas de *CSS* o *Javascript*.

Gestión del proyecto

Control de versiones, repositorio de GitHub

Debido a que a lo largo de la carrera nos hemos decantado por el uso de esta herramienta y ya estaba familiarizada con ella será lo que vayamos a utilizar para alojar el proyecto en un repositorio y poder acceder a él de forma controlada por medio de un cliente de *git*.

[10] Para conocer un poco más conda: https://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_paquetes

ZenHub

Es una extensión del navegador que se integra con GitHub para poder controlar las tareas que hay que llevar a cabo en el proyecto de forma ágil y rápida de ver.

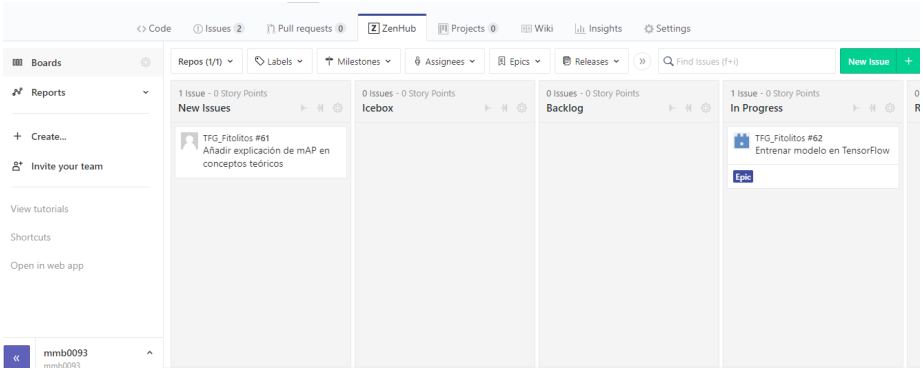

Figura 4.7: Vista del tablero que nos ofrece la herramienta de ZenHub

Documntación con LaTeX

LaTeX [6] es un lenguaje de marcado mediante el cual escribiremos la memoria y los anexos de la documentación del proyecto.

Es fácil de usar y nos ofrece bastantes ventajas frente a otros editores de texto en cuanto a la personalización.

Por otro lado la curvatura de aprendizaje de LaTeX puede ser pronunciada al principio.

El entorno en el que se va a editar con LaTeX es *Texstudio*. [11]

Otras librerías

Flask Dance

Para hacer el login se ha usado una librería de Flask que te permite generar un control de acceso mediante otras aplicaciones como *GitHub*, *Twitter* o *Facebook* [4]. Se genera un cliente en la API[12] de Google y se añade

[11] Enlace a la página de Texstudio https://www.texstudio.org/

[12] La API de Google con la que se ha generado el usuario para la aplicación: https://www.googleapis.com/auth/userinfo.profile

para qué direcciones se quiere restringir el acceso, es decir, hay que asociarle las rutas del proyecto.

Aspectos relevantes del desarrollo del proyecto

5.1. Elección del etiquetador

Al principio del proyecto se pensó en dividir la funcionalidad del etiquetador de la del clasificador, por lo que una de las primeras tareas fue la de buscar uno que fuese fácil de documentar, fácil de entender por el usuario que vaya a generar las etiquetas, fácil de instalar y que devolviese el conjunto de datos de tal forma que la lectura sea sencilla de hacer desde un script de python.

Después de investigar qe etiquetaro era e más decuado para esta tarea, se optó por una llamada *Alp's Labeling Tools for Deep Learning*[13] Al final se tomó la decisión de preparar una máquina virtual con el etiquetador instalado y hacer un tutorial por escrito para que el usuario aprendiese a generar las etiquetas como se puede apreciar en la imagen 5.8

[13]Enlace para ver la documentación del etiquetador https://alpslabel.wordpress.com/

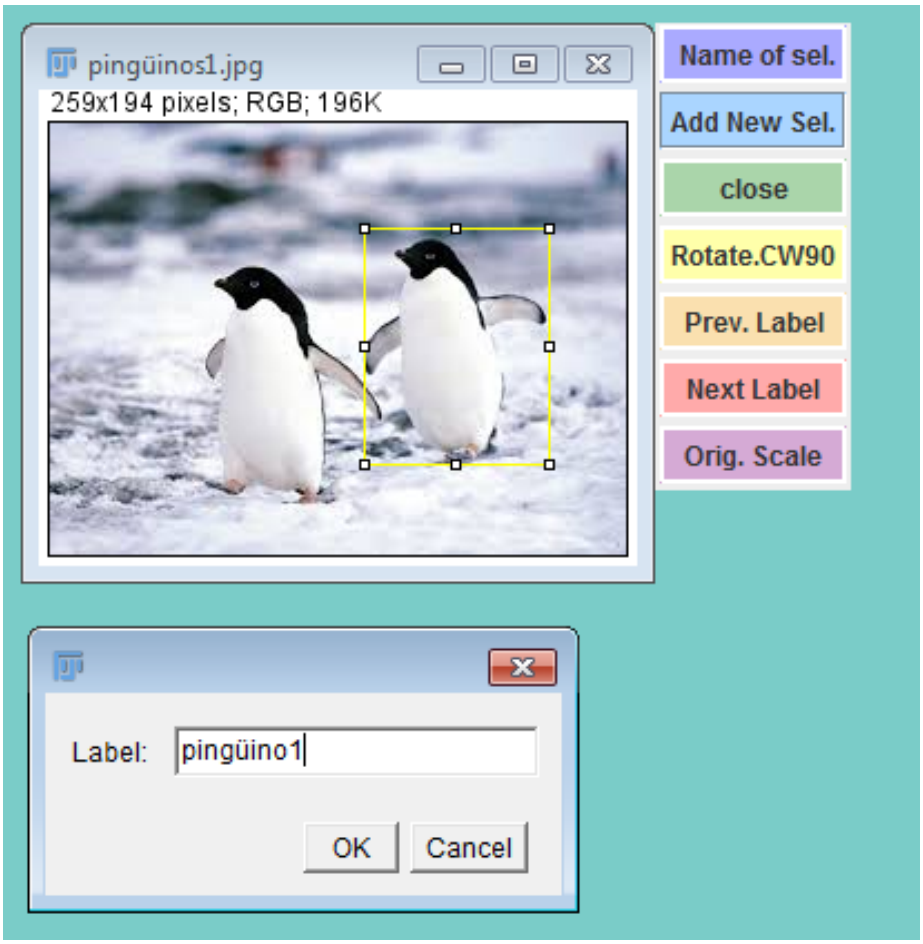

Figura 5.8: Captura del primer prototipo del etiquetador

La idea terminó desechándose una vez estuvo finalizada porque la entrada de datos y la nomenclatura de las etiquetas podría ser propensa a errores debido a que depende mucho de que el usuario que etiquete no se equivoque escribiendo el nombre de la etiqueta, además de que la labor llegaba a ser muy pesada.

Esto podría traer problemas con la lectura de datos en el futuro además de pérdida de información.

Llegado a este punto se optó por ofrecer un etiquetador personalizado dentro de la aplicación el cual se puede ver en la imagen 5.9

Figura 5.9: Vista del etiquetador personalizado.

Problemas con el etiquetador

El etiquetador que se proporciona en la aplcación está basado en otro de código abierto llamado *VGG Image Annotator (VIA)*.

Se escogió porque ofrecía muchas opciones de carga y descarga y modificar la salida de datos para que el *csv* fuese compatible con el clasificador y porque estaba escrito en *JavaScript* completamente. Esto de que estuviese en *JavaScript* era importante porque ha facilitado su integración en el proyecto, no obstante hubo una serie de obstáculos a mencionar:

1. En primer lugar, se trata de una aplicación muy grande y con muchas funcionalidades que a priori no se necesitaban, lo cual complicaba la comprensión del código.

2. No había a penas comentarios en todo el código, había muchas funciones que no se utilizaban y los nombres de las variables, en muchas ocasiones, eran poco o nada descriptivas.

El código del etiquetador es correcto, pero tiene lagunas funcionales que se han tenido que solucionar.

3. A la hora de introducir las etiquetas, el etiquetador original necesita que las introduzcas a mano, pero es precisamente esto de lo que huíamos la última vez, así que se optó por ponerle botones con el nombre ya predefinido de los fitolitos de interés para ahorrarnos erores futuros.

4. Redimensionamiento de las etiquetas. Para recoger los datos en el *csv* existía una problemática porque si cambiabas una etiqueta de tamaño el valor no se actualizaba correctamente el archivo donde se recogían las coordenadas de la etiqueta.

   Para solucionar esto se detecto el problema mediante pruebas de etiquetado y se arregló permitiendo que se actualizasen los valores.

5. La falta de experiencia por mi parte en lenguajes como *JavaScript* y *HTML* hizo mella en los tiempos dedicados al etiquetador. Por suerte la curva de aprendizaje es ligera.

6. El diseño de la interfaz. Fue un problema porque al principio hubo problemas con el despliegue, como más adelante explicaremos, y al no poder acceder a los permisos de las carpetas, no podíamos guardar los datos en el servidor, así que para que esto no fuese un impedimento a la hora de obtener las etiquetas, hicimos que no se guardasen en el servidor ni las etiquetas ni las imágenes, pero estas no las descargábamos porque el usuario las subía desde su propio ordenador, por lo que es innecesario.

Al final el etiquetador funciona como se esperaba que lo hiciese.

5.2. Problemas en el depliegue

La primera opción que se usó para alojar la aplicación fue *Nanobox*. La conforman un conjunto de herramientas que facilitan el mantenimiento de la aplicación. En mi caso generó problemas desde el principio.

Nanobox: Instalación de *Docker*

Nanobox requiere de un grupo de docker al que se le otorgan privilegios para manipular carpetas. En mi caso no hubo forma de otorgárselos a la primera. Traté de hacerlo por consola pero tampoco los aceptaba.

Borré toda la instalación que había hecha hasta el momento de *Nanobox* y la reinstalé, y conseguí alojar la aplicación, aun así, los permisos de las carpetas en los que se querían guardar las imágenes y etiquetas seguían siendo insuficientes y no hubo forma de cambiarlo.

La siguiente opción fue la dejarlo en local para la presentación, así al menos se verán sus funcionalidades al completo sn depender de permisos en carpetas que no podemos controlar.

En retrospectiva esto fue bueno en varios sentidos:

- El primero es que este servidor solo era gratuito porque contaba con un paquete de ayuda para usuarios estudiantes lo que significaba que en octubre caducaba y empezaría a ser de pago sin previo aviso, por lo que en el fondo lo considero una preocupación menos.

- El Hosting por el que opté después es mucho más cómodo y rápido y aunque tenga menos opciones de mantenimiento y depuración, la prefiero por la simpleza de su mecanismo basado en *Git*.

- He tenido la oportunidad de probar dos opciones distintas para el despliegue lo cual me ha ayudado a ser más crítica con qué herramientas he de tener en consideración y hsta qué punto es necesario gastar tiempo en ellas.

- Evito las incompatibilidades del fichero *yml* y los ficheros de configuración del despliegue, que se quedan en el repositorio.

También ha tenido sus inconvenientes:

- Se ha perdido mucho tiempo en algo que sí que aporta valor al proyecto pero no a costa de perder tanto el tiempo en ello. Lo mejor huvise sido haber dejado de lado esta forma de intentar desplegar y haber pasado a la sigiente, para no obcecarme.

- Se han perdido todas las opciones de mantenimiento, depuración y control de recursos que ofrecía *nanobox*, que no son necesarios a priori, pero no estaba de más cuando había errores o cuando se quería gestionar los recursos que consumía la aplicación.

Solución: Cambio a *Pythonanywhere*

Después de mucho tiempo perdido en el despliegue, se decidió que de momento trabajaría en local para poder avanzar.

Estando más avanzado el proyecto, y con el fin de dejarlo alojado en un servidor, se tomó la decisión de probar otra forma de despliegue.

Pythonanywhere ofreció un despliegue sencillo mediante una consola de bash que hace uso de git.

5.3. El clasificador

Uno de los frentes abiertos del proyecto es la futura integración de un clasificador que permita al usuario subir imágenes que contengan fitolitos y que el sistema de devuelva las imágenes con las etiquetas correspondientes en los fitolitos.

Se ha explorado la opción de hacerlo usando modelos ya hechos proporcionados por *Tensorflow*.

5.4. Problemas con las versiones: CUDA, cuDNN y TensorFlow

Un requisito indispensable es instalar la librería *TensorFlow-GPU*, debido a que vamos a tratar con imágenes, irá más rápido que con CPU.

Análogamente a esto hay que tener instalado CUDA y cuDNN[14] teniendo cuidado con las versiones tanto de *TensorFlow* como de estas dos anteriores. No es extraño que haya incompatibilidades entre las versiones más recientes. Es mejor informarse bien antes de instalar ninguno de estos elementos.

Por lo general, en la página oficial de *Tensorflow* te especifcan qué versiones son compatibles con qué otros sistemas[8].

5.5. Entorno Virtual

Como seguramente haya que estar calibrando las versiones de las librerías que se vayan a instalar, es altamente recomendable crear un entorno virtual

[14] Acceso a la página de nvidia: https://developer.nvidia.com/cudnn

para que no haya problemas, ni en el proyecto en el que se está trabajando, ni en los proyectos ajenos a este.

Por lo general, lo ideal es trabajar con versiones actualizadas, pero en este caso es mejor tener la documentación a mano y comprender qué necesidades tiene el clasificador que se quiere entrenar.

5.6. Repositorio de TensorFlow

En las pruebas que se han hecho se ha trabajado con el repositorio de *GitHub* de *TensorFlow*[15]

En él podemos encontrar todos los recursos necesarios para entrenar un modelo. Se puede clonar o descargar para poder hacer uso de sus funciones.

En este repositorio hay ejemplos de uso para las diferentes funcionalidades que ofrece, por lo que es recomendable saber de antemano qué recursos necesitas para la finalidad que se desea.

5.7. Modelos pre-entrenados

Se ha intentado entrenar al clasificador con dos tipos de modelos[16] distintos, pero realmente Tensorflow nos ofrece muchas alternativas, que pueden ser más o meno precisas y cuya velocidad de entrenamiento variará de unas a otras (ver imagen 5.10).

[15] Repositorio de TensorFlow: https://github.com/tensorflow/models

[16] Enlace para acceder a los modelos: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

Figura 5.10: Modelos ya entrenados que TensorFlow pone a disposición pública

5.8. Entrenamiento

Teniendo los conjuntos de imágenes ya etiquetadas y estando generados todos los recursos necesarios para empezar con el entrenamiento, el programa falla debido a una incompatibilidad entre los tipos de datos que se procesan.

Existen varias posibilidades desde mi punto de vista:

- Que el problema esté entre las versiones de las librerías de las que se hace uso.
- Que el etiquetador esté recogiendo un tipo de dato que no es compatible.
- Que los modelos con los que se ha probado el entrenamiento procesen un tipo distinto de datos.
- Que me haya equivocado en cualquier parte del proceso previo al entrenamiento.

En cualquier caso, el resultado ha sido que no ha podido entrenarse el modelo.

Para poner un ejemplo ilustrativo a cerca de como debería haberse clasificado una imagen, podemos observar el de la imagen 5.11. Este clasificador ha sido entrenado con el modelo *ssd mobilenet v1 coco*, uno de los más rápidos y que menos potencia necesita, pero también uno de los menos precisos.

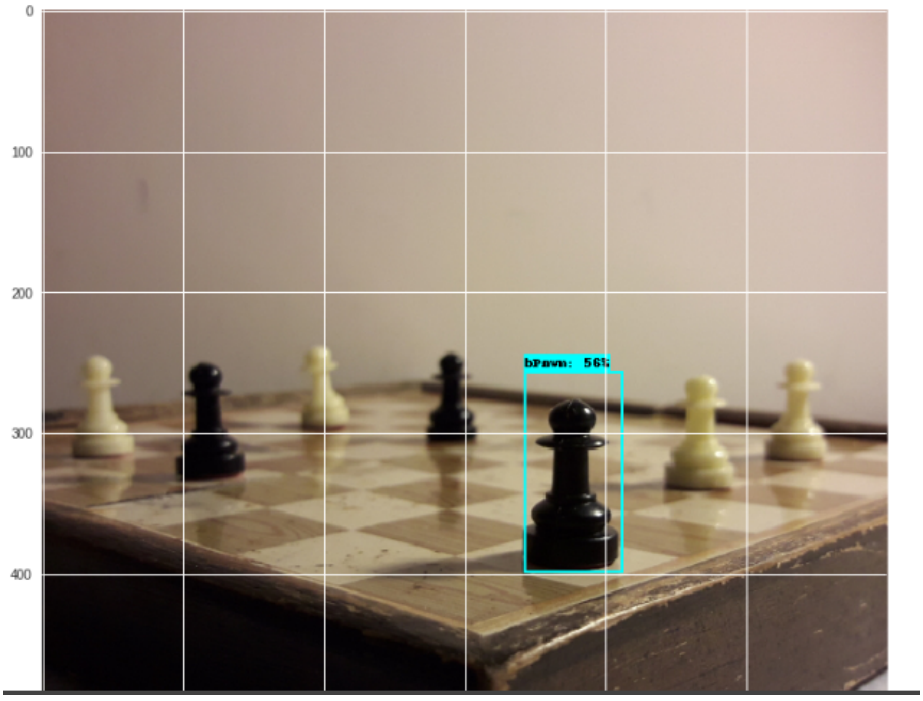

Figura 5.11: Ficha negra clasificada correctamente con el clasificador

Trabajos relacionados

En cuanto a reconocimiento automático de fitolitos o cualquier otro proyecto que intente solventar la misma problemática, el más cercano es el trabajo final de grado realizado por Jaime Sagüillo [9].

Su trabajo se llama *Sistema de reconocimieto automático en arqueobotánica* y se centra en la construción de un sistema de reconocimiento automático de fitolitos.

Su proyecto distingue entre fitolito y no fitolito y trata de solucionar la problemática que se presenta al analizar las muestras de forma manual. Hace uso de la ventana deslizante para la subdivisión de las imágenes en varios recortes que serán los que el clasificador identifique como fitolito o no fitolito.

El mayor problema al que se ha enfrentado en dicho proyecto es el de la falta de imágenes pero lo solventa haciendo uso de técnicas de data augmentation, para generar conjuntos más grandes a partir de otros más pequeños e insuficientes.

Cuando empecé yo el proyecto comprobamos previamente mediante un pequeño script si las etiquetas que devolvía su clasificador eran correctas, pero rara vez coincidían, seguramente debido al problema que hubo con la falta de imágenes de muestra.

Para generar el conjunto de datos de entrenamiento (aunque no haya podido hacer uso de él) usé parte de las imágenes que el generó gracias a las técnicas de data augmentation.

Gracias a su trabajo he logrado comprender muchos conceptos previos al inicio del mío.

Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

Se han presentado muchos problemas desde el inicio del proyecto en casi todos los aspectos técnicos que he tenido que afrontar. El primer problema al que me tuve que enfrentar fue al planteamiento de un etiquetador que recogiese de forma segura todo los datos en el formato correcto. Después de probar e investigar al final se decidió personalizar uno y dejarlo funcional.

Otro de los problemas que estuvo presente durante el despliegue de la aplicación con Nanobox, desde la instalación de los requisitos hasta los permisos de acceso a las carpetas una vez estuvo hosteado. Al final la solución consistió en cambiar de método de despliegue.

El mas pesado de los obstáculos ha sido sin ninguna duda el clasificador. Si bien es cierto que he aprendido a reducir un error, a reconocerlo y a diagnosticarlo, me hubiese gustado sacarlo adelante.

Por otro lado he aprendido mucho a cerca de la arquitectura cliente-servidor, he adquirido conocimientos en *HTML* y en *JavaScript*, nuevas librerías de *Python*, y aunque me hubiese gustado aprender más, también he aprendido algo de *Tensorflow*.

A nivel técnico he aprendido bastantes cosas, pero personalmente creo que la más importante ha sido a gestionar mi tiempo y conocer mi ritmo de trabajo.

Ha supuesto todo un reto para mí compaginarlo con el resto del curso, las labores de delegación y ya en verano, con las práctica extracurriculares.

Aunque sigo sin estar satisfecha con el resultado práctico del trabajo y siendo consciente de que hay muchas cosas mejorables, sí que estoy satisfecha con el hecho de haber puesto a prueba los conocimientos obtenidos a lo largo de la carrera y por supuesto, con todos los conocimientos nuevos que he adquirido.

7.2. Líneas de trabajo futuras

La principal línea de trabajo que veo es la conseguir entrenar al clasificador y ponerlo en producción en el servidor. Si se sigue usando Tensorflow como lo estaba haciendo yo, ya se dispondría del material necesario para el entrenamiento (en el caso de que el material sea bueno).

Aunque probablemente sea mejor explorar otros tipos de frentes. Existen más librerías a parte de TensorFlow, por ejemplo Keras.

El etiquetador es mejorable, estéticamente no me gusta como escala cuando la imagen que le pasas es demasiado grande. Sería deseable que se añadiese una opción para poder subir imágenes y etiquetas, a veces el conjunto es muy grande y el usuario puede decidir guardar los cambios y seguir en otro momento.

Quizá sería buena idea plantear un sistema de proyectos para que el usuario pueda guardar diferentes conjuntos de entrenamiento clasificar diferentes conjuntos de imágenes. En ese caso lo más seguro es que hubiese que cambiar de servidor a otro con mayor capacidad.

La galería es una parte de mi aplicación en la que considero que se pueden incluir muchos más aspectos de carácter funcional, como por ejemplo, eliminar fotos.

Bibliografía

[1] *Documentación de python.*

[2] Redes neuronales convolucionales.

[3] Anaconda. Anaconda documentation.

[4] David Baumgold. Flask dance.

[5] Raúl E. López Briega. Redes neuronales convolucionales con tensorflow.

[6] Latex. Documentación de latex.

[7] Carlos A. Monsalve M. Catálogo preliminar de fitolitos producidos por algunas plantas asociadas a las actividades humanas en el suroeste de antioquía, colombia.

[8] TensorFlow Official. Tensorflow version compatibility.

[9] Jaime Sagüillo Revilla. Sistema de reconocimieto automático en arqueobotánica.

[10] Dr Melissa Rosenzweig. Archaeobotany is a sub-specialization within environmental.

[11] Margaret Rouse. Inteligencia artificial, o ai.

[12] Wikipedia. Bootstrap.

[13] Wikipedia. Cliente-servidor.

[14] Wikipedia. Fitolitos.

[15] Wikipedia. Html.

[16] Wikipedia. Pythonanywhere, mayo 2012.