



Universitat Oberta
de Catalunya

Comparación de clasificadores basados en *deep learning* para la identificación de especies vegetales a partir de imágenes de herbario y de datos geográficos

Berta Gallego Páramo

Máster universitario de Bioinformática y Bioestadística
Área 4: Machine Learning

Romina Astrid Rebrij
David Merino Arranz

Junio 2020



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](#)

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Comparación de clasificadores basados en deep learning para la identificación de especies vegetales a partir de imágenes de herbario y de datos geográficos</i>
Nombre del autor:	<i>Berta Gallego Páramo</i>
Nombre del consultor/a:	<i>Romina Astrid Rebrij</i>
Nombre del PRA:	<i>David Merino Arranz</i>
Fecha de entrega (mm/aaaa):	<i>06/2020</i>
Titulación::	<i>Máster en Bioinformática y Bioestadística</i>
Área del Trabajo Final:	<i>Área 4: Machine Learning</i>
Idioma del trabajo:	<i>Castellano</i>
Palabras clave	<i>Deep learning, ConvNet, clasificación automática de plantas</i>
Resumen del Trabajo (máximo 250 palabras): <i>Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.</i>	
<p>La identificación de especies vegetales se ha ido automatizando desde hace décadas mediante diversos métodos. En la actualidad, la disponibilidad de datos de biodiversidad en plataformas públicas facilita el acceso a una cantidad creciente de datos.</p> <p>En el presente trabajo se ha desarrollado un <i>pipeline</i> en Python para la descarga y pre-procesado de imágenes de herbario y datos geográficos del repositorio público de biodiversidad GBIF, obteniéndose un conjunto de más de 11400 imágenes pertenecientes a 43 especies de árboles nativos de Reino Unido.</p> <p>A partir de dichas imágenes se ha construido un clasificador por transferencia de aprendizaje, basado en la arquitectura de CNN VGG16, que alcanza una exactitud de en torno al 70%. Asimismo, se ha obtenido un clasificador de coordenadas geográficas, basado en redes neuronales de aprendizaje profundo con una exactitud menor del 20%. Dada la limitación de ambos modelos por separado para obtener un clasificador fiable, se propone la combinación de ambos.</p> <p>El método de descarga, preprocesado y construcción de los clasificadores diseñado es extrapolable a otros grupos de especies vegetales y puede facilitar el desarrollo de clasificadores en este área.</p>	

Abstract (in English, 250 words or less):

Plant species identification has been automated for decades using a diverse range of methods. Nowadays, the availability of biodiversity data on public platforms facilitates access to an increasing amount of data.

In the present work, a Python *pipeline* has been developed for the download and pre-processing of herbarium images and geographic data from the public repository of biodiversity GBIF, obtaining a set of more than 11,400 images belonging to 43 species of native trees in the United Kingdom.

Based on these images, a classifier based on the CNN VGG16 architecture has been built using transfer learning, achieving an accuracy of about 70%. Likewise, a geographic coordinate classifier, based on deep learning, has been obtained with an accuracy of only around 20%. Given the limitation of both models separately, it is proposed that using stacking could improve the overall accuracy.

The download, pre-processing and models implementation can be extrapolated to other groups of plant species, so it can facilitate the development of classifiers in this area.

Índice general

1. Introducción	1
1.1. Contexto y justificación del Trabajo	1
1.2. Objetivos del Trabajo	1
1.3. Enfoque y método seguido	2
1.4. Planificación del Trabajo	3
1.5. Breve sumario de productos obtenidos	3
1.5.1. Memoria	3
1.5.2. Repositorio	3
1.6. Breve descripción de los otros capítulos de la memoria	3
1.6.1. Capítulo 2 : Obtención de datos y procesado.	3
1.6.2. Capítulo 3 : Modelos de clasificación.	4
2. Obtención y procesado de datos	6
2.1. Metodología	6
2.1.1. Planteamiento general	6
2.1.2. La API de GBIF	7
2.1.3. El conjunto de datos	8
2.1.4. La descarga de datos	8
2.1.5. El pre-procesado de datos	8
El pre-procesado de imágenes	8
El pre-procesado de datos geográficos	10
2.2. Resultados y discusión	10
3. Modelos de clasificación	16
3.1. Clasificador a partir de imágenes	16
3.1.1. Metodología	16
La elección del clasificador	16
Generación de los datos de entrada	17
Ajuste del modelo	18
3.1.2. Resultados y discusión	19
3.2. Clasificador a partir de datos geográficos	19
3.2.1. Metodología	19
Implementación del clasificador	19
3.2.2. Resultados y discusión	22
4. Conclusiones	24
Glosario	26
Bibliografía	31

A. Especies de la búsqueda	33
B. Filtros de la descarga	34
C. Arquitecturas VGG16 y VGG19	35
D. Código del módulo <i>data</i>	37
D.1. Obtención de datos	37
D.1.1. Módulos ejecutables desde el terminal	37
Imágenes	37
Datos geográficos	40
D.1.2. Módulos comunes	41
D.2. Pre-procesado de datos	46
D.2.1. Imágenes	46
D.2.2. Datos geográficos	60
E. Código de los clasificadores	62

Índice de figuras

1.1.	Planificación temporal del trabajo.	5
2.1.	Ejemplo de solicitud-respuesta a través de la API de GBIF	7
2.2.	Estructura del módulo <i>data</i>	9
2.3.	<i>Pipeline</i> de descarga.	10
2.4.	Ejemplo de ejecución de los scripts de descarga.	11
2.5.	Selección del área de interés de las imágenes.	12
2.6.	<i>Pipeline</i> del preprocesado de imágenes.	13
2.7.	Tamaño de las imágenes descargadas.	14
2.8.	<i>Pipeline</i> del preprocesado de datos geográficos.	14
2.9.	Imágenes y datos geográficos descargados.	15
2.10.	Distribución y frecuencia del dato geográfico.	15
3.1.	Efectos de la aumentación de imágenes.	17
3.2.	Ajuste de transferencia de aprendizaje para clasificadores VGG16 y VGG19.	20
3.3.	Ajuste del número de épocas en el entrenamiento del clasificador VGG16 con tres bloques pre-entrenados.	21
3.4.	Exactitud y pérdida en el entrenamiento del clasificador basado en datos geográficos.	22
3.5.	Matriz de confusión para las predicciones del clasificador basado en datos geográficos.	23
C.1.	Arquitectura del modelo VGG16.	35
C.2.	Arquitectura del modelo VGG19.	36

Capítulo 1

Introducción

1.1. Contexto y justificación del Trabajo

La identificación de especies vegetales se ha ido automatizando desde hace décadas, con el avance de la computación, por medio de distintos métodos (1) (2) (3) (4) (5) (6). En los últimos años ha tenido lugar un gran aumento de la cantidad de imágenes de especímenes disponibles, debido tanto a la digitalización masiva de las colecciones de herbario, como a la captura de imágenes por naturalistas amateur y su publicación en plataformas de ciencia ciudadana (7) (8) (9). Las imágenes son, por tanto, un tipo de dato comparativamente barato y fácil de obtener. Las redes neuronales convolucionales (CNN/ConvNet) es el tipo de algoritmo de aprendizaje profundo que se emplea comúnmente, por su probada precisión, para construir dichas herramientas (10) (11). Sin embargo, la aplicación de estos modelos para clasificar imágenes heterogéneas, como son las capturadas *in situ*, limita su precisión (12).

El aprendizaje automático se aplica para resolver problemas en áreas del conocimiento diversas, lo cual lo convierte en una herramienta muy atractiva. En concreto, su uso para la identificación automática de especies aporta un gran valor en proyectos relacionados con la conservación de la biodiversidad. En estos proyectos, catalogar especies es el punto de partida para evaluar qué medidas son necesarias para asegurar la preservación de los recursos de un ecosistema. Es por ello que este tema ha estado generando gran interés en la comunidad científica en los últimos años.

Recientemente, se han construido modelos que combinan imágenes de especímenes de herbario en bruto (13) y también procesadas junto con características morfológicas anotadas (10). Wäldchen y Mäder ponen en valor en su artículo de revisión en 2018 (14) la utilidad de las áreas de distribución de las especies, así como su fenología, como complemento a las imágenes. En el presente trabajo se desea automatizar el proceso de obtención y procesado de datos de biodiversidad, así como evaluar clasificadores basados en digitalizaciones de herbario y en coordenadas geográficas para la identificación de especímenes de herbario, tomando como ejemplo las especies de árboles nativos de Reino Unido.

1.2. Objetivos del Trabajo

1. Obtener un *pipeline* para la descarga y pre-procesado de imágenes de herbario y datos geográficos de biodiversidad, con vistas al desarrollo de clasificadores automáticos.

- a) Obtener un *pipeline* para la descarga y pre-procesado de digitalizaciones de herbario.
 - b) Obtener un *pipeline* para la descarga y pre-procesado de datos geográficos.
2. Obtener modelos de clasificación para la identificación de especies de árboles de Reino Unido con vistas a su integración.
 - a) Obtener un modelo de clasificación a partir de las imágenes de especímenes de herbario.
 - b) Obtener un modelo de clasificación basado en el área de distribución de las especies de estudio..

1.3. Enfoque y método seguido

El lenguaje de programación elegido para desarrollar los *pipelines* de descarga y procesado, así como los modelos de clasificación es Python. Una de las razones de elegir Python frente a otros lenguajes de programación, como R, es su versatilidad y librerías disponibles. En concreto para la construcción y validación de los modelos se ha elegido el *framework* TensorFlow (junto con la librería Keras) frente a PyTorch. Según un artículo de 2018 ([14](#)) esta combinación es la más popular y, ante una funcionalidad similar, se prefiere por la posibilidad de obtener recursos de aprendizaje más fácilmente.

Se ha llevado a cabo un control de versiones del código usando git, creando un repositorio público en la plataforma GitHub. Además de ser una medida de seguridad, permite que el código esté disponible para otros usuarios y sea reproducible. En este contexto, se ha creado un entorno virtual con las dependencias instaladas usando las librerías pip y virtualenv.

La metodología seguida en las diferentes partes del proyecto se resume a continuación (se amplia en la sección correspondiente).

- La obtención de datos se ha planteado de forma programática a través de la API de la Global Biodiversity Information Facility ([GBIF](#))[\(15\)](#), que es uno de los mayores repositorios de datos de biodiversidad. La justificación y descripción de proceso se detalla en en capítulo [2](#).
- Con el fin de mejorar la clasificación conviene seleccionar el área de interés de las imágenes de las láminas de herbario, lo cual se ha realizado automáticamente con unos parámetros fijados por iteración y empleando la librería Pillow de Python (ver sección [2.1.5](#)).
- Las coordenadas geográficas también precisan ser filtradas para descartar datos incorrectos (fuera del área de distribución acotada), o poco precisos (ver la sección [2.1.5](#), relativa al preprocesado de datos geográficos).
- En cuanto al clasificador de imágenes, las redes neuronales de aprendizaje profundo, y en concreto las redes neuronales convolucionales ([CNN/ConvNet](#)),

son el método que proporciona actualmente mejores resultados en la clasificación de imágenes y es el que se ha empleado recientemente en modelos de clasificación de especies vegetales (13),(14), (16). En concreto, se ha decidido emplear un tipo de clasificador de arquitectura conocida con distintas opciones de **transferencia de aprendizaje** (ver sección 3.1.1).

- En el caso de los datos geográficos, se ha decidido emplear como clasificador una red neuronal de aprendizaje profundo sencilla (ver detalles en 3.2.1).
- La evaluación de cada modelo se ha realizado a través de la exactitud en el **conjunto de validación** durante el **entrenamiento**. Además se han empleado matrices de confusión para comprobar la calidad de las predicciones sobre el **conjunto de test**.

1.4. Planificación del Trabajo

La planificación temporal del trabajo se resume en un diagrama de Gantt (1.1). Se ha precisado realizar ajustes del mismo, debido al desconocimiento inicial de algunos aspectos metodológicos, incumplimiento y cambios de plazos.

1.5. Breve sumario de productos obtenidos

1.5.1. Memoria

La memoria hace referencia a este texto, donde se resume y discute los resultados obtenidos y la metodología empleada, así como adjunta el código desarrollado.

1.5.2. Repositorio

El repositorio se refiere al conjunto de scripts de Python desarrollados, que se presentan de forma estructurada en una plataforma pública (GitHub). La reproducción de dichos scripts está facilitada por la posibilidad de crear un entorno virtual con las dependencias necesarias. Se puede acceder al mismo aquí: .

1.6. Breve descripción de los otros capítulos de la memoria

1.6.1. Capítulo 2 : Obtención de datos y procesado.

Este capítulo hace referencia al trabajo desarrollado para alcanzar el primer objetivo general. En él se ahonda en la metodología, así como se presentan los resultados relacionados.

1.6.2. Capítulo 3 : Modelos de clasificación.

En este capítulo se explica cómo se ha llevado a cabo la implementación de los modelos de clasificación mencionados en el segundo objetivo general.

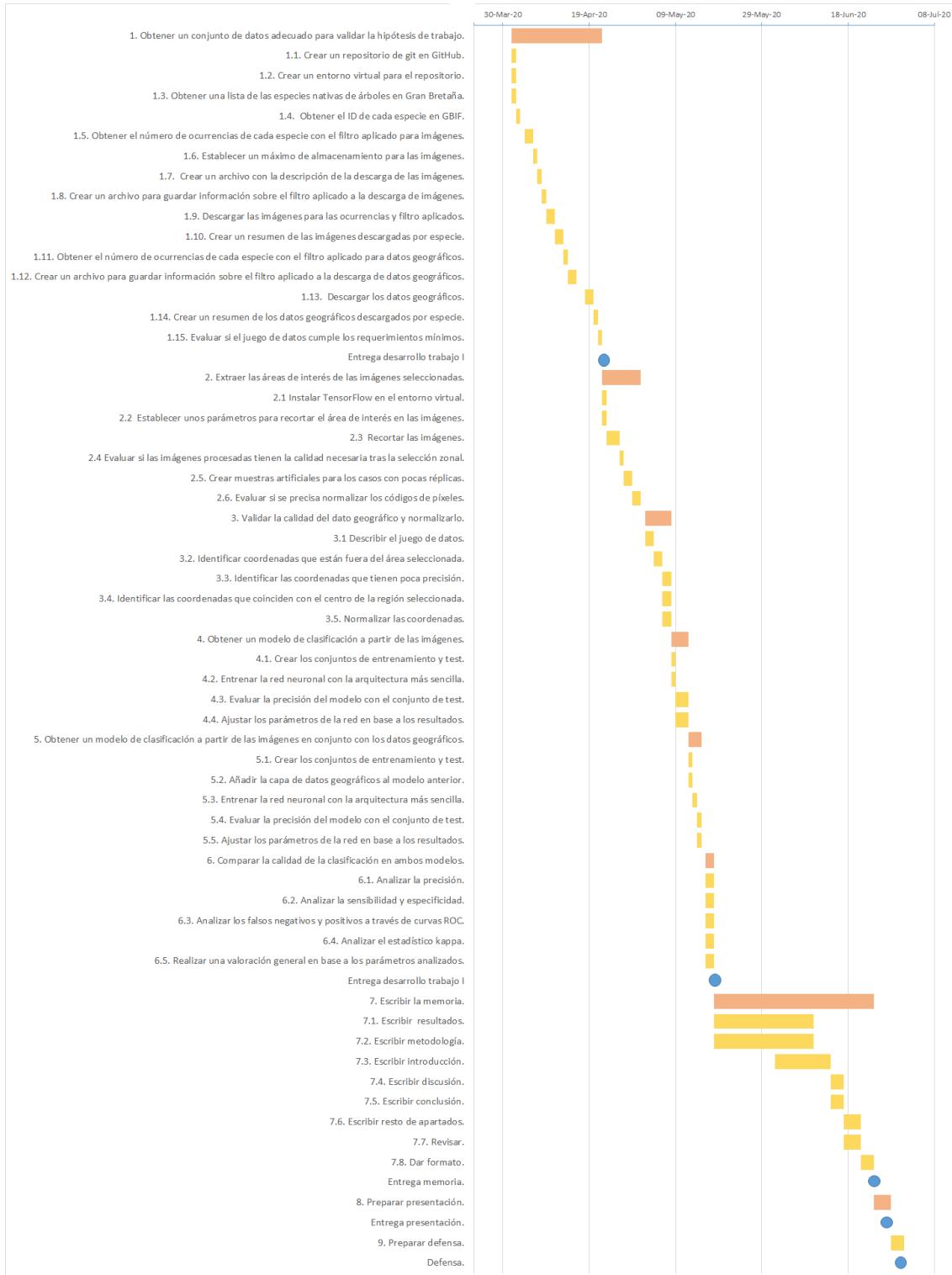


Figura 1.1: Planificación temporal del trabajo, donde se indica la duración de las tareas (barras amarillas) que contribuyen al desarrollo de cada objetivo (barras naranjas) y los hitos durante el proyecto (puntos azules).

Capítulo 2

Obtención y procesado de datos

2.1. Metodología

2.1.1. Planteamiento general

En el planteamiento inicial de este proyecto se tuvo que decidir:

- ¿Qué grupo de especies elegir para entrenar los clasificadores?
- ¿Cómo obtener los datos relativos a las especies seleccionadas?

En cuanto al grupo de especies, tendría que ser un conjunto conceptualmente acotado para que la identificación tuviera un sentido funcional (por ejemplo, las especies de un género, o los géneros dentro de una familia o las especies de una región determinada...). Además el grupo seleccionado habría de contar con un número de clases lo suficientemente grande como para poder plantear un experimento realista, pero lo suficientemente pequeño como para poder realizar una prueba de concepto con las limitaciones de tiempo y computacionales asociadas al proyecto.

En lo referente a cómo obtener los datos, cabría elegir un conjunto de imágenes ya disponible, listo para entrenar al clasificador. Esta opción, aunque facilita la consecución del objetivo final, oculta una parte del proceso de construcción de un clasificador de aprendizaje automático que es fundamental: la obtención de datos y el pre-procesado de los mismos. Además, obvia la utilidad del creciente número de imágenes de herbario disponibles en los repositorios de biodiversidad. También ignoraba el aumento de datos de biodiversidad generados a partir de plataformas de ciencia ciudadana, como iNaturalist o Pl@ntNet ([8](#)). Por ello, se decidió obtener los datos crudos de un repositorio público.

En base a las dos premisas anteriores el proceso de selección tiene implícita una incertidumbre y potencialmente una iteración en el proceso de descarga de datos. Ésta fue la motivación inicial para plantear una obtención y procesado de datos de forma programática, pero no la única. El desarrollo del pipeline de descarga y pre-procesado responde a tres motivaciones:

1. Ofrecer flexibilidad de cambio en el acotado del conjunto de datos.
2. Implementar un proceso transparente que pueda ser reproducido y mejorado.
3. Facilitar la obtención de nuevos conjuntos de datos para entrenar otros clasificadores (puesto que no existe un clasificador único de la biodiversidad y que se necesitará construir nuevos clasificadores para otros grupos de especies o regiones determinadas).

2.1.2. La API de GBIF

El repositorio de biodiversidad de referencia a nivel global es el Global Biodiversity Information Facility (**GBIF**) (15), que recibe datos de instituciones científicas y de otros repositorios de biodiversidad y es, por tanto, el portal que se ha elegido para la descarga de las imágenes y de los datos geográficos.

Con objeto de facilitar la transferencia de datos de forma programática GBIF pone a disposición de los usuarios una **API** de tipo **REST**. Un ejemplo de uso se ilustra en la figura 2.1. Aunque existen diferentes métodos posibles de realizar una solicitud todas las solicitudes a las que me voy a referir en el desarrollo del pipeline son de tipo *get* (obtener datos). La API de GBIF devuelve objetos de tipo **JSON** (JavaScript Object Notation) que tienen una estructura análoga a un objeto de tipo diccionario en Python y que también está representada en la figura 2.1.

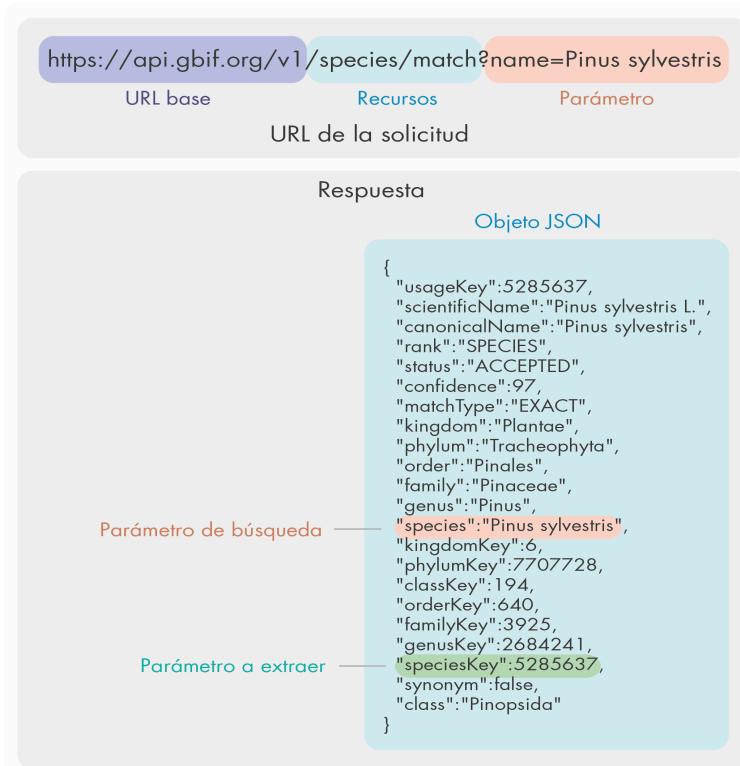


Figura 2.1: Ejemplo de solicitud-respuesta empleando la API de GBIF para la búsqueda del identificador interno (`speciesKey`) de la especie *Pinus sylvestris*. Se indican la URL de base de la API de GBIF, el punto final para buscar el recurso “`species/match`” la información relativa a la especie *Pinus sylvestris* en el parámetro “`name`”.

Cabe mencionar que GBIF ha desarrollado la librería de Python `pygbif` (17) para simplificar el uso de su API. Sin embargo, se ha decidido no emplearla por dos motivos. Un motivo tiene que ver coste-beneficio que supone instalar una librería nueva (si el trabajo ahorrado compensa el riesgo de que la librería quede desactualizada). Un segundo motivo es que las librerías ocultan, en diferentes grados, procesos subyacentes que se ha considerado oportuno exponer. La librería `requests` (18) es una alternativa más generalista que permite implementar la solicitud de datos de una forma relativamente sencilla y no opaca.

2.1.3. El conjunto de datos

Según lo expuesto al principio del capítulo existen múltiples opciones para conformar el conjunto de datos. Se ha acotado geográficamente la búsqueda a Reino Unido y a las especies de los árboles nativos porque ofrece un número de clases razonable (46, ver A) y presentan hojas conspícuas como característica identificable por el algoritmo de clasificación. Por tanto, los datos que se requieren descargar de GBIF son:

- Las imágenes de herbario de las 46 especies nativas de Reino Unido.
- Las coordenadas geográficas registradas en Reino Unido donde se han observado dichas especies.

2.1.4. La descarga de datos

La descarga de los datos se realiza a través de una serie de scripts escritos en Python ejecutables desde el terminal. Para ello he construido una estructura de módulos dentro del módulo “data”, como se ilustra en la figura 2.2. En este módulo existen una serie de archivos comunes a la descarga de las imágenes y de los datos geográficos y otros propios de cada tipo de dato. El código completo puede consultarse en el apéndice D.

El proceso de descarga, esquematizado en la figura 2.3, consiste en dos pasos a nivel de usuario:

1. **Configuración de los filtros para la solicitud de datos.** La lista de especies, así como un cierto número de parámetros, se configuran en un CSV para cada tipo de dato (los filtros). Los parámetros considerados se pueden consultar en el apéndice B.
2. **Ejecución de los scripts de la solicitud.** La ejecución de los scripts como módulos permite organizar el código en unidades conceptuales reutilizables e importables desde otros scripts. La sintaxis del comando completo se indica en la figura 2.3.

En la figura 2.4 se muestra el proceso de descarga a través de los mensajes de ejecución.

2.1.5. El pre-procesado de datos

El pre-procesado de imágenes

Las digitalizaciones de especímenes de herbario presentan tres elementos en común: el espécimen (normalmente situado en la parte central), una etiqueta con el nombre de la especie y otros datos asociados a la recolección (generalmente en la parte inferior) y una paleta de colores de referencia (se suele situar en alguno de los márgenes), como se observa en la primera imagen de la figura 2.5. La inclusión de elementos que no son propios del organismo a identificar puede dar lugar a sesgos

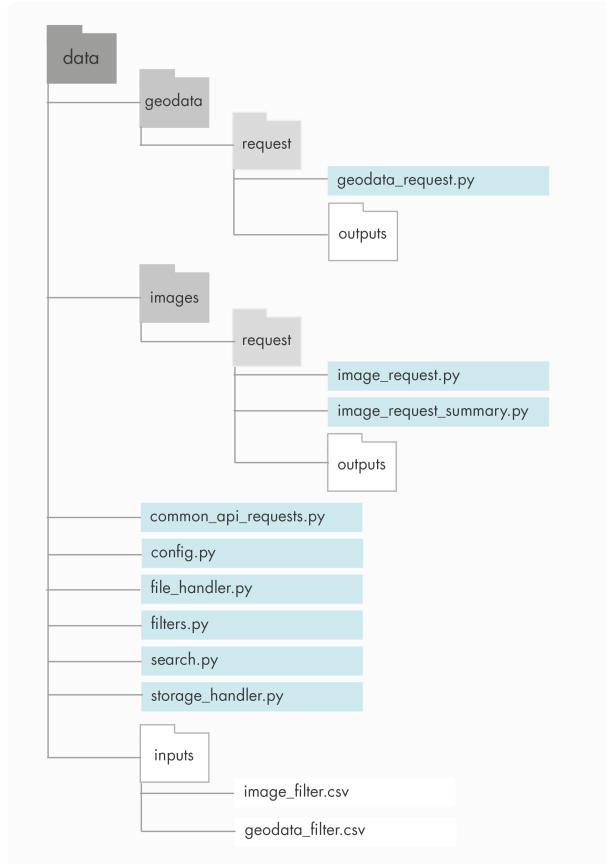


Figura 2.2: Estructura del módulo *data* relativa a la descarga de imágenes y datos geográficos. Las carpetas y archivos incluidos en el repositorio se representan en gris y azul, respectivamente. Las carpetas no pertenecientes al repositorio, pero que se integran en el proceso de descarga de datos (*inputs* y *outputs*), así como los archivos dentro de ellas, aparecen en blanco.

en la clasificación y se ha considerado importante eliminarlos en la medida de lo posible.

En la figura 2.5 se muestra, además, cómo se selecciona el área de interés de la lámina. Para ello, se han seguido los pasos detallados en la figura 2.6. A partir del histograma y del gráfico de dispersión (figura 2.7) se evalúa cuáles son la altura y anchura más apropiadas para el conjunto de imágenes (se ha elegido 1200x2000). El área de corte se fija por ensayo-error en base a las nuevas dimensiones y teniendo en cuenta que los elementos ajenos al espécimen se localizan en los márgenes.

Una vez se han aplicado los patrones de redimensionado y corte a todas las imágenes, se debe realizar una exploración visual para identificar imágenes no válidas (fotos de especímenes vivos, preparaciones de microscopio, especímenes cubiertos...). Este proceso no está optimizado, ya que requiere anotar los nombres de todas las imágenes incorrectas en el script *5_discard_images.py*. El número de imágenes a descartar ha sido en torno al 5 % del total de imágenes descargadas.

Por último, se separan pseudo-aleatoriamente las imágenes en subcarpetas según se vayan a emplear para entrenar el clasificador (*train* y validación -*val*-) o para probarlo (*test*).

La siguiente parte del procesado (normalización y aumentación) se realiza de

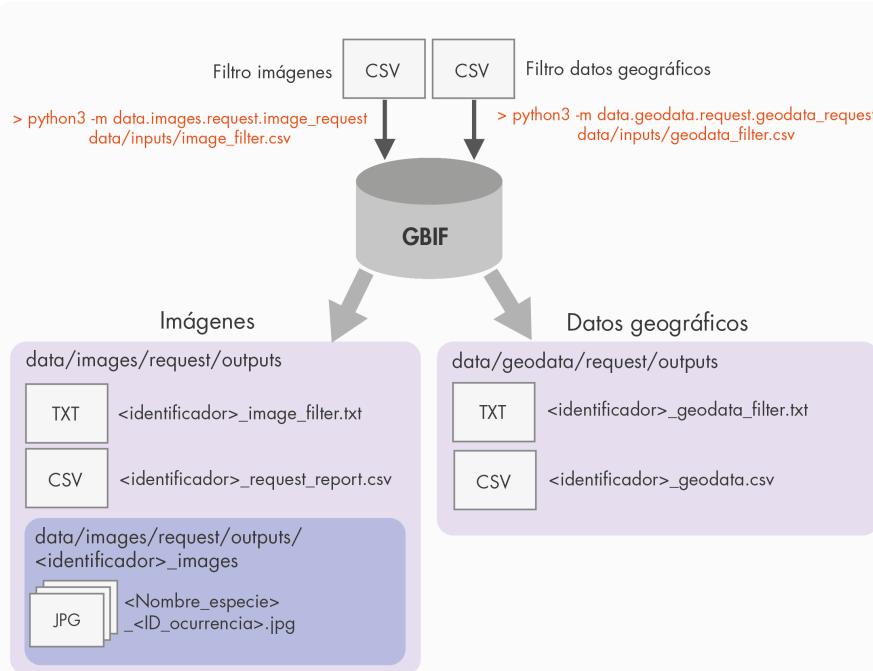


Figura 2.3: Proceso de descarga de imágenes y datos geográficos de la base de datos de GBIF, empleando su API. En la parte superior se aparecen los archivos de tipo CSV donde se han de configurar los filtros. La ruta de acceso a dichos archivos ha de incluirse como argumento al ejecutar los módulos `data.images.request.image_request` y `data.geodata.request.geodata_request` (indicado en rojo). Como resultado de ejecutar cada uno de los scripts se obtienen los archivos en su correspondiente carpeta “outputs” (abajo, en morado). Los resultados de los distintos filtros aplicados se diferencian gracias a un identificador único autogenerado.

forma integrada con la implementación del clasificador, utilizando las herramientas de las librerías TensorFlow y Keras.

El pre-procesado de datos geográficos

El filtrado de los datos geográficos se realiza en un solo paso (figura 2.8), que incluye: seleccionar las variables de interés (latitud, longitud e incertidumbre), des-
cartar los registros donde falta alguno de estos datos, así como los puntos con una
incertidumbre mayor a 10 km y generar gráficos para la explorar los datos disponibles
por especie, el histograma de la latitud y la longitud y la distribución de las ocurren-
cias en el mapa de Reino Unido. Esta exploración se vuelve a realizar previo a la
implementación del clasificador.

2.2. Resultados y discusión

Se descargaron las imágenes y los datos geográficos de un máximo de 300 ocu-
rrencias por especie. En el conjunto de datos geográficos el filtrado dio lugar a la
muestra representada en la figura 2.9 B. En ella se aprecia que algunas especies
cuentan con un número de ejemplos considerablemente inferior a la media, como

Oblención de los identificadores de especie

Nombre de la búsqueda Identificador del filtro

Starting request **Geographical data from native trees in GB** identified by: **ec90f56d45a6c0a36ada6d3b50df37ba**

Getting GBIF taxon key for species in the list.
Alnus glutinosa identified by taxon key: 2076213.
Frangula alnus identified by taxon key: 3039454.
Fraxinus excelsior identified by taxon key: 3172358.
Populus tremula identified by taxon key: 3040249.
Fagus sylvatica identified by taxon key: 3092336.
Betula pubescens identified by taxon key: **31188014**. **Identificador de especie**
Betula pendula identified by taxon key: 3033221.
Prunus spinosa identified by taxon key: 3023221.
Buxus sempervirens identified by taxon key: 2984671.
Rhamnus cathartica identified by taxon key: 3039491.
Prunus padus identified by taxon key: 3020837.
Prunus avium identified by taxon key: 3020791.

Oblención de identificadores de ocurrencias de cada especie

Número de ocurrencias

Salix cinerea subsp. oleifolia identified by taxon key: 5582546.
Salix viminalis identified by taxon key: 5372933.
Unable to find taxon key for Salix alba.
WARNING: Species without a speciesKey: ['Sorbus arranensis', 'Sorbus aria', 'Salix alba']

Getting GBIF occurrences keys for species in the list.
Number of occurrences for Alnus glutinosa: **50** Número de ocurrencias
Number of occurrences for Frangula alnus: 50
Number of occurrences for Fraxinus excelsior: 50
Number of occurrences for Populus tremula: 50
Number of occurrences for Fagus sylvatica: 50
Number of occurrences for Betula pubescens: 50
Number of occurrences for Betula pendula: 50

Descarga de datos de cada ocurrencia

Datos geográficos

Alnus glutinosa [sp.1/42]: records downloaded
Frangula alnus [sp.2/42]: records downloaded
Fraxinus excelsior [sp.3/42]: records downloaded
Populus tremula [sp.4/42]: records downloaded
Fagus sylvatica [sp.5/42]: records downloaded
Betula pubescens [sp.6/42]: records downloaded

Imágenes

Alnus glutinosa[sp.1/42]: 23/27 images downloaded.
Alnus glutinosa[sp.1/42]: 24/27 images downloaded.
Alnus glutinosa[sp.1/42]: 25/27 images downloaded.
Alnus glutinosa[sp.1/42]: 26/27 images downloaded.
Frangula alnus[sp.2/42]: 1/21 images downloaded.
Frangula alnus[sp.2/42]: 2/21 images downloaded.
Frangula alnus[sp.2/42]: 3/21 images downloaded.
Frangula alnus[sp.2/42]: 4/21 images downloaded.
Frangula alnus[sp.2/42]: 5/21 images downloaded.

Figura 2.4: Mensajes de ejecución de los scripts de descarga de imágenes y datos geográficos (`image_request.py` y `geodata_request.py`) durante los tres subprocesos de solicitud a través de la API de GBIF

es el caso de *Pyrus cordata* y *Tilia x europaea*. Existen estrategias para abordar el problema del **desbalance de clases** (19), sin embargo, por cuestión de recursos, se ha optado por incluir todas las especies en el entrenamiento y no tratar de paliarlo, aunque sería deseable mejorar este punto.

El mapa de la figura 2.10 A prueba que las coordenadas de todas la ocurrencias filtradas caen dentro del área acotada, así como C y D ilustran cómo los datos geográficos pueden ayudar a discernir entre especies que son similares morfológicamente.

Este tipo de pre-preprocesado de imágenes de herbario supone una mejora con respecto al método empleado por Carranza-Rojas *et al.* en 2017, donde se alimentaba el clasificador con imágenes sin seleccionar el área de interés.



Figura 2.5: Selección del área de interés de una digitalización de una lámina de herbario de *Pinus sylvestris*. Debajo de cada imagen se indica su dimensión en píxeles (px) tal y como se ha seleccionado para el conjunto de imágenes de estudio.

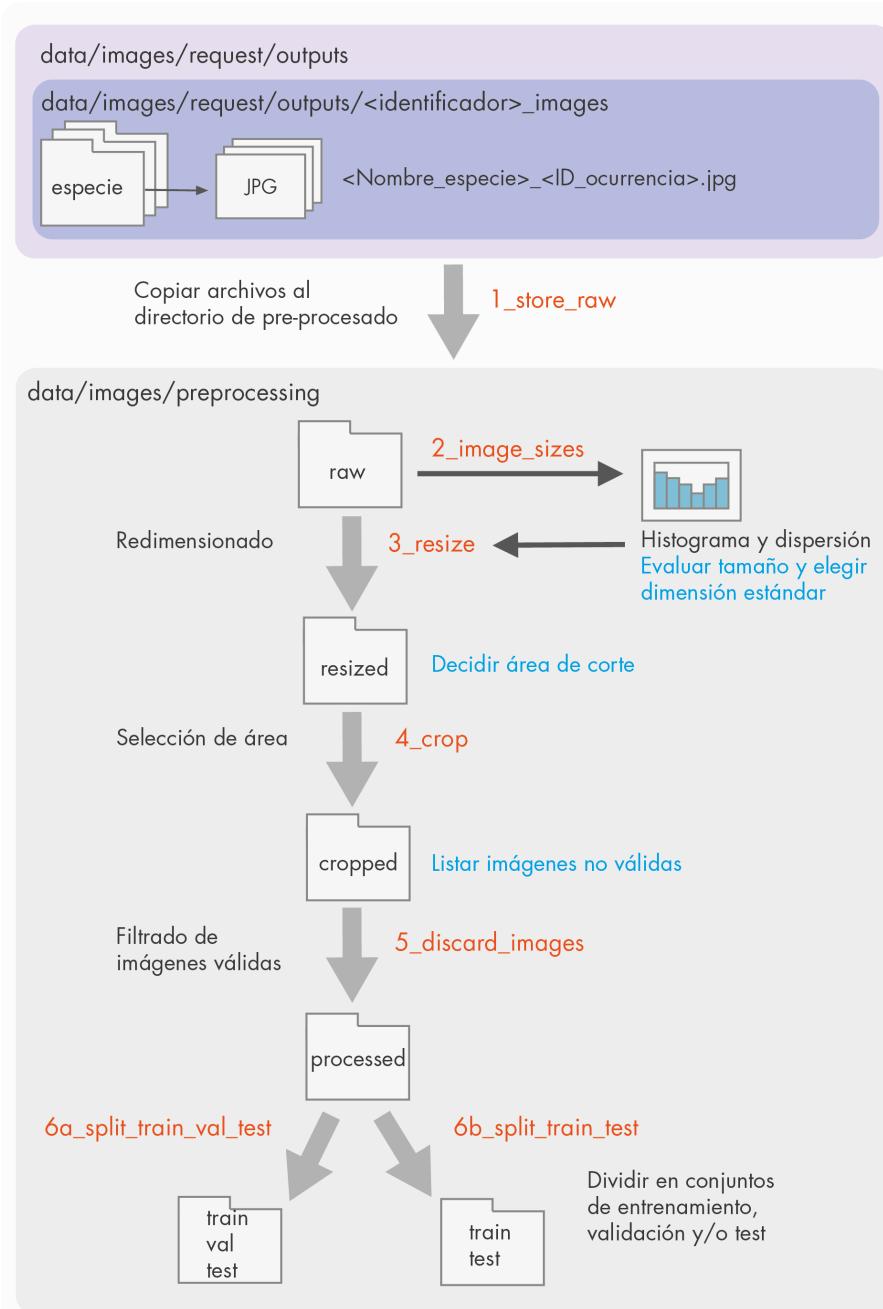


Figura 2.6: Secuencia del pre-procesado de imágenes. En rojo se indica el nombre de los scripts a ejecutar en cada paso y en azul la manipulación adicional requerida para cada nuevo juego de datos.

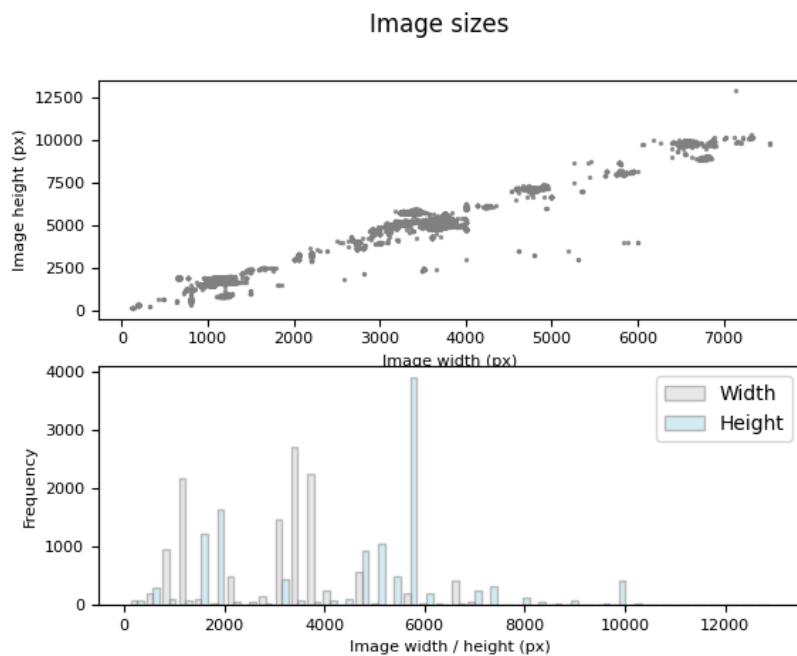


Figura 2.7: Dispersión (arriba) y frecuencia (abajo) de la anchura y altura de las imágenes del conjunto de datos de estudio. Las anchuras más frecuentes están en torno a 1000 y 3500 px, mientras que las alturas que más se repiten están alrededor de 2000 y 6000 px. Consideradas conjuntamente, las imágenes con dimensiones cercanas a 3500 x 6000 son las más numerosas, seguidas de las que miden en torno a 1200 x 2000 px.

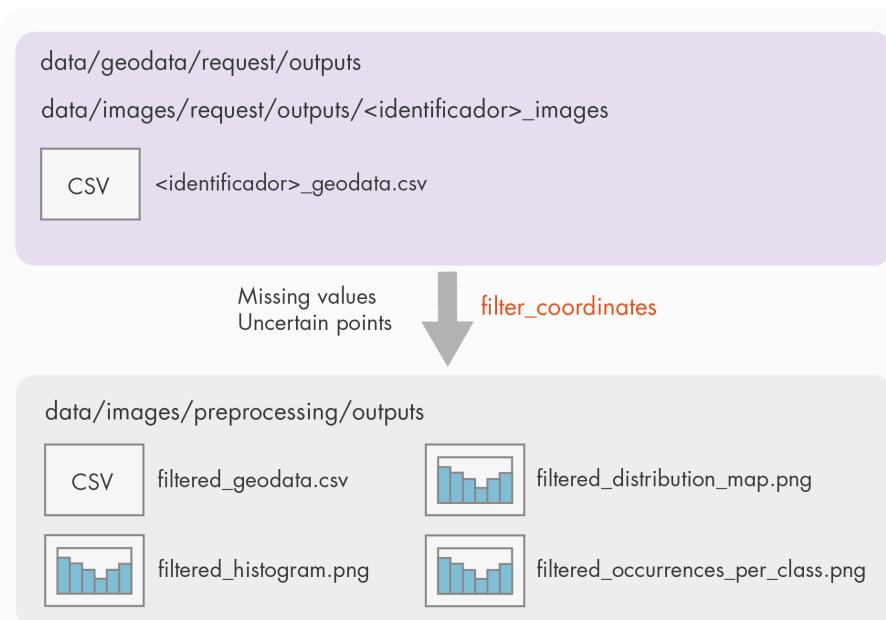


Figura 2.8: Secuencia del pre-procesado de datos geográficos. En rojo se indica el nombre del scripts a ejecutar.

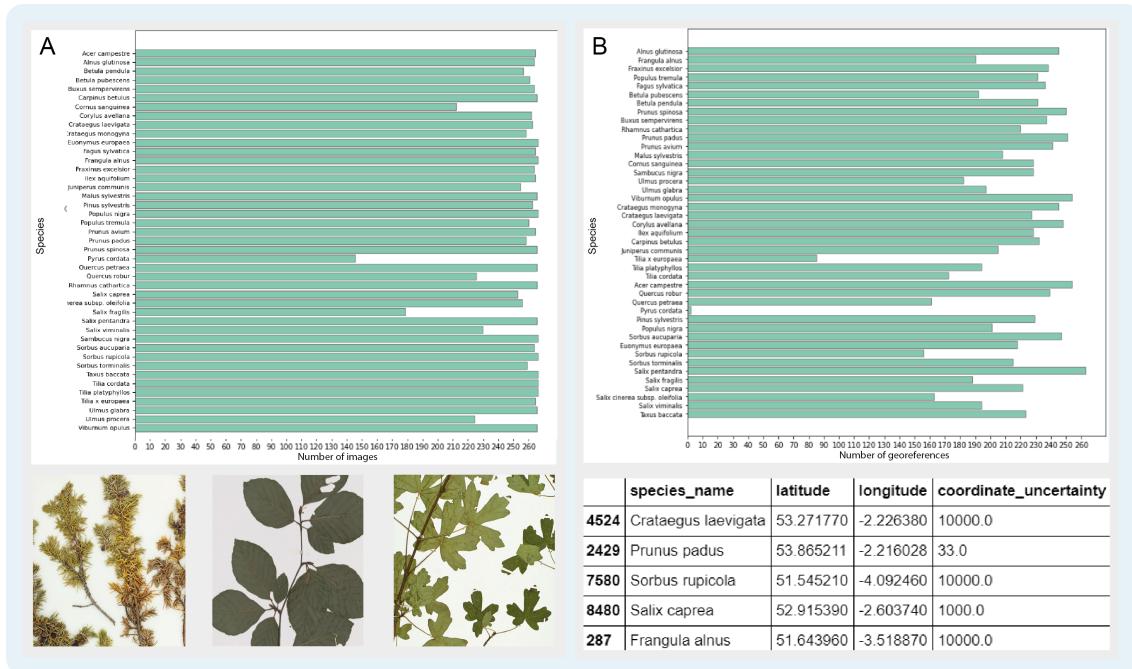


Figura 2.9: Imágenes (A) y datos geográficos (B) descargados. Número de ejemplos por especie (A y B, arriba), muestra de imágenes procesadas (A, abajo) y muestra de las variables del conjunto de datos geográficos (B, abajo). El conjunto de datos geográficos se presenta después del filtrado.

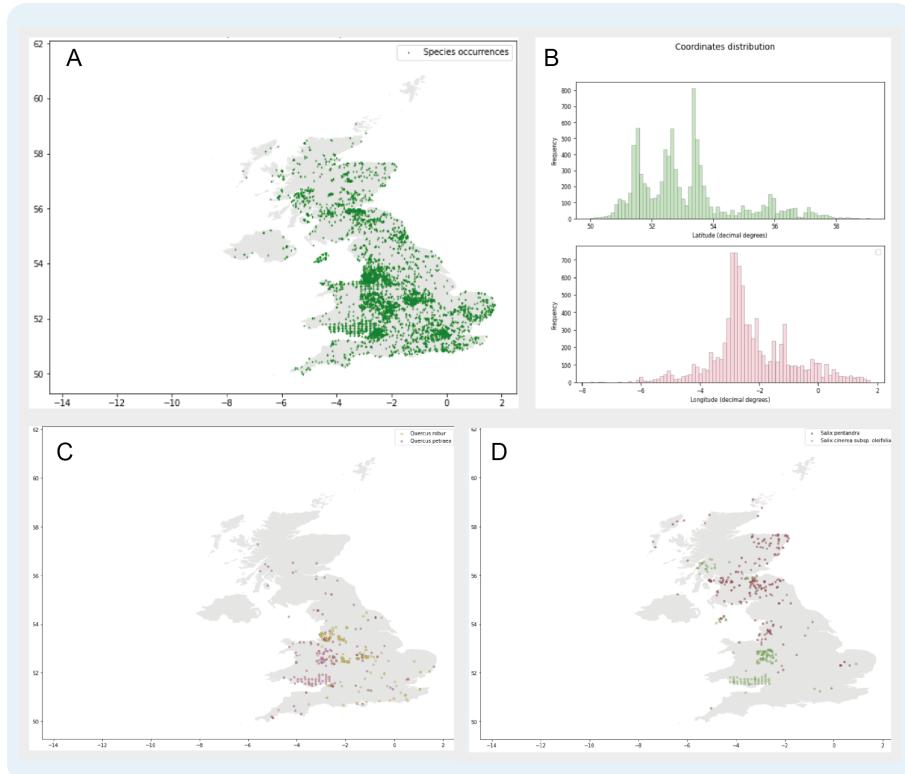


Figura 2.10: Distribución (A, C y D) y frecuencia (B) de las coordenadas geográficas en el área de Reino Unido. C y D ilustran cómo la localización geográfica puede ayudar a identificar especies de árboles morfológicamente similares.

Capítulo 3

Modelos de clasificación

3.1. Clasificador a partir de imágenes

3.1.1. Metodología

La elección del clasificador

En los certámenes de **ImageNet** (referente actual de la clasificación de imágenes a gran escala (20)) de 2010 y 2012, Krizhevsky *et al.* (21) entrenaron una de las mayores redes neuronales convolucionales hasta el momento, obteniendo unos resultados considerablemente mejores que el resto de participantes. Este tipo de red se basaba en los modelos con múltiples capas propuestos por Fukushima y Miyake en 1982 (22) y LeCun en 1989 (23). La CNN de Krizhevsky *et al.* contenía cinco capas convolucionales y tres completamente conectadas. También incluyeron capas de agrupamiento y *dropout* en dos de las capas completamente conectadas (para evitar *overfitting*). Este modelo, conocido como **AlexNet**, marcó un rumbo en la clasificación automática de imágenes a gran escala. Un artículo reciente de Khan *et al.* compila la evolución de las distintas arquitecturas de CNNs desde sus orígenes (24).

Dada la probada eficacia de las CNNs para la clasificación de imágenes y, en concreto, la de los modelos presentados a los sucesivos certámenes de ImageNet, tenía sentido emplear una de las arquitecturas conocidas para entrenar el clasificador con el conjunto de datos de trabajo. Sin embargo el tamaño de mi conjunto de entrenamiento (unas 6800 imágenes repartidas en 43 clases) limita el rendimiento del modelo si ha de entrenarse desde cero. Una práctica común cuando se parte de conjuntos de datos reducidos es la **transferencia de aprendizaje**. Esta práctica permite usar los pesos de una o más capas de una red neuronal que ya ha sido entrenada con otro conjunto de datos.

Carranza-Rojas *et al.* (13) abordaron un problema similar en la clasificación de imágenes de especímenes de herbario. Probaron diferentes opciones de transferencia de aprendizaje en una versión modificada del modelo de 2014 **GoogleNet** (25), concluyendo que los modelos cuyas primeras capas habían sido entrenadas con el conjunto de datos de ImageNet conseguían mayor exactitud en la clasificación que otros que habían sido pre-entrenados con conjuntos de datos de imágenes de plantas, pero de menor tamaño que el de ImageNet. En 2018 Brahimi *et al.* publicaron un comprensivo análisis de diferentes CNNs y tipos de entrenamiento aplicados a la detección de enfermedad en hoja, obteniendo exactitudes mayores en aquéllas en las que se había realizado transferencia de aprendizaje (16).

En la librería keras de TensorFlow existen algunos de las arquitecturas de redes populares, a las que se les pueden cargar los pesos obtenidos del entrenamiento con ImageNet. Algunas de las arquitecturas más recientes basadas en CNN son variantes de **VGG16** (26)(2014), **ResNet** (27)(2015) y **DenseNet** (28)(2016). En un escenario con tiempo y recursos ilimitado hubiera probado las diferentes arquitecturas disponibles. Sin embargo, como en cada modelo ya existe un cierto número de parámetros a ajustar he decidido centrarme en la arquitectura **VGG16** y su variante **VGG19** (el diagrama de su arquitectura puede consultarse en el el apéndice C).

Generación de los datos de entrada

Las entradas de datos en el modelo ha de realizarse en forma de matrices (tensores). Las imágenes son matrices de 3 dimensiones: ancho, alto y canales. El ancho y la altura se miden en píxeles y los canales hacen referencia a los valores de color (el valor del canal es 1 si está en blanco y negro y 3 si es color en RGB (rojo, verde, azul) o HSV (tono, saturación , valor) formatos: 2 y 4 son en blanco y negro o en color con un canal alfa (transparencia).

Con el fin de mejorar la exactitud y prevenir el **overfitting** del clasificador, se realizó una **aumentación** de las imágenes. Los parámetros empleados, así como el método y los resultados se muestran en la figura 3.1 y en el código E. Esta operación se realiza directamente al cargar los datos en un objeto de tipo tensor con `ImageDataGenerator()` y `flow_from_directory()`. Además, se incorpora en este paso la **normalización** (**re-escalado de píxeles**) de valores originales (entre 0 y 255) a normalizados (entre 0 y 1). Se eligió un tamaño de imagen de 224x224 px porque son las dimensiones establecidas en la capa de entrada de los modelos VGG16 y VGG19. El parámetro `class_mode` hace referencia al tipo de clasificación (en este caso categórica, por tener más de dos categorías).



Figura 3.1: Efectos de la aumentación de imágenes en el conjunto de entrenamiento. Arriba: muestra aleatoria de imágenes de distintos especímenes sin modificar. Abajo: muestra de algunas de las modificaciones realizadas sobre la imagen de un espécimen (sin ordenar: rotación, giro vertical, giro horizontal, brillo, acercamiento, desplazamiento).

De modo análogo a la generación del conjunto de entrenamiento, se accede a las imágenes del **conjunto de validación** (ver apéndice E). Este conjunto se emplea para evaluar las predicciones realizadas por el modelo durante el entrenamiento.

Ajuste del modelo

Aplicar **transferencia de aprendizaje** implica tener que decidir qué capas se entrena con el conjunto de datos de trabajo y cuáles se "congelan" (se fija el valor de los pesos obtenidos con ImageNet). Se parte de que las capas con los pesos fijados han de ser las primeras, ya que extraen características más generales de las imágenes, mientras que conviene entrenar las últimas con el conjunto de datos de trabajo, especialmente si éste es muy distinto al conjunto que se ha empleado para pre-entrenar el modelo. En los modelos VGG cuentan con cinco bloques de capas C y he decidido acotar el problema a entrenar o "congelar" bloques determinados en lugar de capas individuales. Por tanto, existen cuatro posibilidades de **ajuste del modelo** fijando los pesos en: bloque 1, bloques 1 y 2, bloques 1-3 y bloques 1-4, además de la **extracción de características** fijado todos los pesos (bloques 1-5). En el experimento planteado (cuyos resultados se muestran en la figura 3.2) se empezó fijado los pesos tres de las primeras capas (3.2 B). Como fijar los pesos de los bloques 1 y 2 y los de los cuatro primeros bloques no mejoraron los resultados de fijar los bloques 1-3, no se incluyeron en el experimento los casos extremos de fijar sólo el bloque 1, así como fijar todos (aunque sí se probaron con un menor número de épocas y no se incluyen en la memoria). El mejor resultado del VGG16 se compara con un tipo de entrenamiento análogo en el modelo VGG19 (3.2 B y D). Estos resultados se comentan en la siguiente sección (3.1.2).

Además de fijar o no los pesos de las capas en el modelo de base VGG16, se requiere modificar la última para que presente tantos nodos como número de clases. Para ello, se crea un nuevo modelo al que se le añaden las capas del VGG16 excepto la última, que se añade después como una capa completamente conectada con una **función de activación** de tipo **softmax**. La función *softmax* transforma el vector de la capa anterior en otro vector que representa las probabilidades de cada clase, por lo que es la función indicada para la clasificación en múltiples categorías. La implementación de la arquitectura del modelo se ejemplifica en el código en 3.1.

```

1 # load pre-trained model with the weights
2 vgg16_model = tf.keras.applications.VGG16()
3 # Add the layers of vgg16 model to a new sequential model
4 model = Sequential()
5 for layer in vgg16_model.layers[:-1]: # remove last layer
6     model.add(layer)
7 # Freeze the weights in the layers of blocks 1 and 2
8 for layer in model.layers[:6]:
9     layer.trainable = False
10 for layer in model.layers[6:]:
11     layer.trainable = True
12 # Add last layer for categories
13 model.add(Dense(len(class_names), activation = "softmax"))

```

LISTING 3.1: Implementación del modelo VGG16 con los dos primeros bloques pre-entrenados.

En el experimento descrito, el número máximo de épocas fue 500, aunque se estableció una parada temprana del entrenamiento cuando la pérdida en el conjunto de validación no bajara de un mínimo alcanzado durante 30 épocas (pacienza). Se eligió un **optimizador SGD** y la **tasa de aprendizaje** por defecto (0.01) para el entrenamiento (ver código en el apéndice E). Otra opción común para el optimizador es el Adam (Adaptive Moment Estimation). Se decidió SGD frente a Adam porque éste último bajo las mismas condiciones de entrenamiento y con las tasas de aprendizaje por defecto (en Adam es 0.001) la exactitud máxima alcanzada fue menor del 7 % antes de detenerse en la época 65 (gráfica no incluida). La **función de pérdida** elegida

fue **categorical_crossentropy**, dado que es la que se emplea cuando existen múltiples clases.

3.1.2. Resultados y discusión

La máxima exactitud (en torno al 50 %) se alcanza con el modelo VGG16 con los tres primeros bloques pre-entrenados (figura 3.2 B) después de las 300 épocas. Cuando se pre-entrena un bloque más (3.2 C), la exactitud empieza a alcanzar su máximo (sobre un 40 %) alrededor de las 200 épocas. Asimismo, cuando sólo se transfieren los pesos de los dos primeros bloques a las 300 épocas se alcanza una exactitud menor del 50 %. En este caso, además, se observa un cierto *overfitting* (la exactitud en el conjunto de validación está por debajo que en el de entrenamiento), lo cual coincide con que un número mayor de capas se están entrenando con el conjunto de datos de trabajo y que resulta en que el modelo no generalice bien. El entrenamiento en el modelo VGG19 con los tres primeros bloques pre-entrenados (3.2 D) no parece suponer una ventaja con respecto al modelo VGG16, dado que no alcanza una exactitud del 40 %. Sin embargo, hay que tener en cuenta que esta cifra se alcanza a las 200 épocas y que no se aprecia saturación, como en el VGG16 con el mismo tipo de entrenamiento 100 épocas más tarde. Se podría esperar que esta modalidad de entrenamiento del VGG19 alcanzara valores de exactitud similares a las 300 épocas.

Tras los resultados obtenidos con las distintas opciones de transferencia de aprendizaje, se eligió entrenar el clasificador VGG16 con los tres primeros bloques pre-entrenados con vistas a alcanzar el máximo valor de exactitud posible. Para ello se entrenó el modelo durante 3000 épocas (3.3 A), observándose que hacia las 1000 épocas dejaba de generalizar bien (*había overfitting*), ya que la exactitud medida en el conjunto de validación no aumentaba a la par que la exactitud sobre el conjunto de entrenamiento. Por este motivo se volvió a entrenar el modelo durante 1000 épocas (3.3 B), alcanzando valores de exactitud en torno al 70 %. Carranza-Rojas *et al.* (13) obtuvieron valores máximos similares en su modelo pre-entrenado para clasificar imágenes de herbario (con un juego diferente de datos), pero necesitando un número de épocas muy superior (más de 50 000).

Una mejora de los resultados podría obtenerse con

3.2. Clasificador a partir de datos geográficos

3.2.1. Metodología

Implementación del clasificador

Al contrario que en el caso de las imágenes, la elección del clasificador para las coordenadas geográficas está abierta a diferentes algoritmos de aprendizaje automático que no tienen por qué ser redes neuronales. Por ejemplo, Random Forests superó a otros tipos de algoritmos, incluidas las redes neuronales en un estudio de 2014 basado en coordenadas y otros datos de tipo geológico (29). Sin embargo, se ha implementado una red neuronal sencilla.

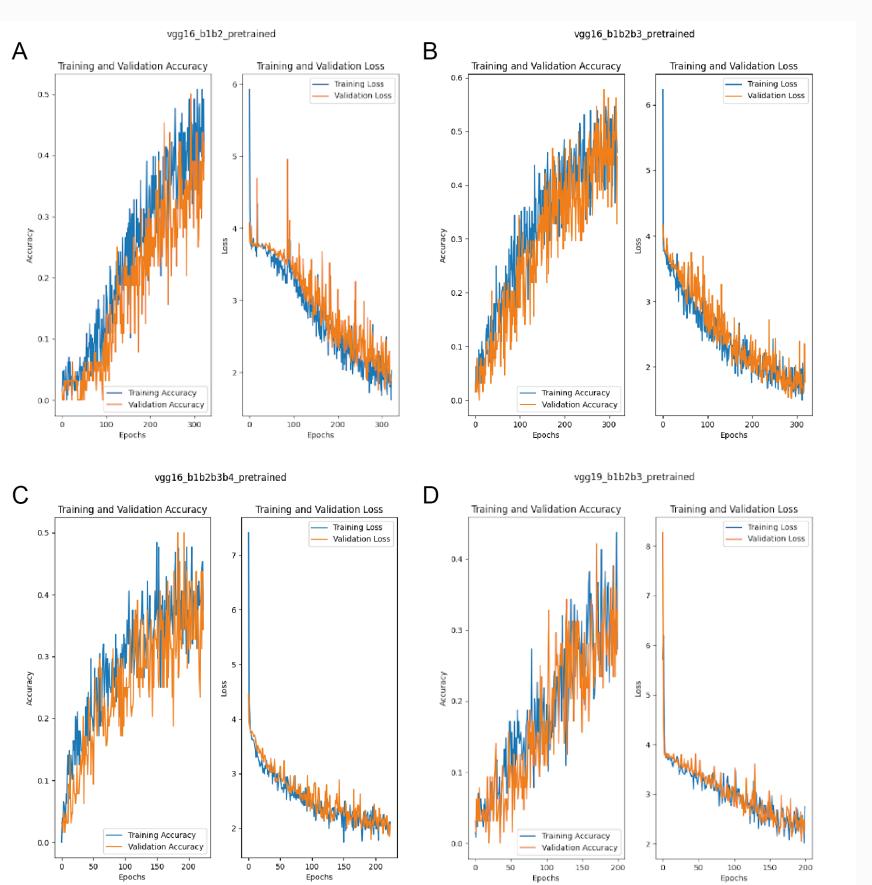


Figura 3.2: Ajuste de transferencia de aprendizaje para clasificadores VGG16 y VGG19. Se mide exactitud (*accuracy*) y pérdida (*loss*) en los conjuntos de entrenamiento (azul) y validación (naranja) durante el entrenamiento de cuatro clasificadores basados en los modelos VGG16 (A, B y C) y VGG19 (D) en los que se han fijado los pesos de diferentes bloques de capas pre-entrenados con el juego de datos de ImageNet (A: bloques 1 y 2; B: bloques 1, 2, y 3; C: bloques 1, 2, 3 y 4; D: bloques 1, 2 y 3). Se empleó un optimizador SGD y una tasa de aprendizaje del 0.01 para entrenar en un máximo de 500 épocas (4 pasos por época y una paciencia de 30 épocas para la pérdida en el conjunto de validación).

La arquitectura de red más simple consiste en 3 capas: La capa de entrada (con un número de nodos igual al número de entidades en el modelo), una capa oculta (con un número variable de nodos) y la capa de salida (con un número de nodos igual al número de clases). Se probaron arquitecturas con mayor número de capas ocultas y nodos, pero daba lugar a *overfitting* del modelo (la pérdida en el conjunto de validación aumentaba en el entrenamiento mientras en el de entrenamiento disminuía). También se redujo el número de ejemplos por clase a 150, dado que se observó en la matriz de confusión que la predicción estaba muy sesgada hacia las clases con mayor.

```

1 # Importing the dataset
2 full_dataset = pd.read_csv(os.path.join(local_path, source_dir, "filtered_coordinates.csv"))
3 dataset = full_dataset.drop(columns="coordinate_uncertainty")
4 # Encode class names
5 class_names.sort()
6 species_to_number = {species_name:class_names.index(species_name) for species_name in class_names}
7 dataset['target']=dataset.apply(lambda r:species_to_number[r.species_name],axis=1)
8 train, test = train_test_split(dataset_num, test_size=0.2)

```

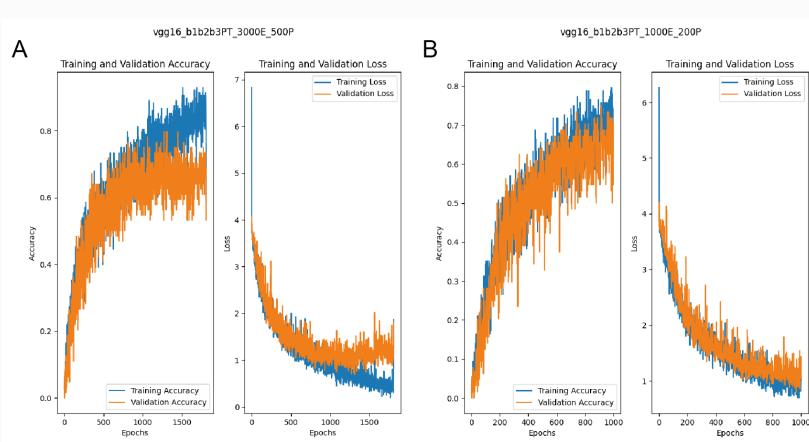


Figura 3.3: Ajuste de número de épocas en el entrenamiento del clasificador VGG16 con tres bloques pre-entrenados. Se mide exactitud (*accuracy*) y pérdida (*loss*) en los conjuntos de entrenamiento (azul) y validación (naranja). Se empleó un optimizador SGD y una tasa de aprendizaje del 0.01 para entrenar en un máximo de 3000 (A) y 1000 (B) épocas (4 pasos por época y una paciencia de 500 (A) y 200 (B). épocas para la pérdida en el conjunto de validación).

```

9 # Split the dataset in train, validation and test
10 train, val = train_test_split(train, test_size=0.2)
11 # Extract predictors (latitude and longitude) for each subset
12 X_train = train.iloc[:, 1:3].values
13 X_val = val.iloc[:, 1:3].values
14 X_test = test.iloc[:, 1:3].values
15 # Extract labels for each subset
16 y_train = train.iloc[:, 2].values
17 y_val = val.iloc[:, 2].values
18 test_labels = test.iloc[:, 2].values
19 y_train = to_categorical(y_train)
20 y_val = to_categorical(y_val)
21 y_test = to_categorical(test_labels)
22 # Normalize the train and validation datasets
23 X_train_norm = normalize(X_train)
24 X_val_norm = normalize(X_val)

```

LISTING 3.2: Preparación de los datos geográficos.

```

1 # CREATING the model
2 model = Sequential()
3 tf.keras.layers.Flatten(input_shape=(X_train_norm.shape[-1],))
4 model.add(layers.Dense(64, activation='relu'))
5 model.add(layers.Dense(len(class_names), activation='softmax'))
6 # COMPIILING the model
7 model.compile(optimizer = "sgd", loss = 'categorical_crossentropy', metrics = ['accuracy'])
8 # Early stopping (when loss does not fall anymore to avoid overfitting)
9 callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience = 50)
10 # Checkpoint to save model weights and history before it stops training
11 checkpoint_filepath = os.path.join(save_dir, "/tmp/checkpoint")
12 model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath = checkpoint_filepath,
13 save_weights_only = True,
14 monitor= "val_acc",
15 save_best_only = True)
16 # TRAINING the model
17 epochs = 500
18 history = model.fit(
19 X_train_norm,
20 y_train,
21 validation_data=(X_val_norm, y_val),
22 batch_size = 10,
23 epochs = epochs
24 )

```

LISTING 3.3: Implementación y entrenamiento del clasificador de datos geográficos.

3.2.2. Resultados y discusión

Según se observa en la figura 3.4 el modelo basado en datos geográficos no ofrece un buen resultado por si solo, ya que la exactitud no llega al 20 %. Además, como se aprecia en la matriz de confusión (3.5), sólo clasifica correctamente ejemplos pertenecientes a algunas de las clases, como *Buxus serpervirens* o *Ulmus glabra*. Por la exploración realizada en los mapas (figura 2.10 y mapas de distribución de otras especies no incluidos), no sorprende que en un área relativamente pequeña como es Reino Unido, aunque la localización de las especies pueda ayudar a identificar especies concretas, al considerar las 43 no se encuentren grandes diferencias en las áreas de distribución.

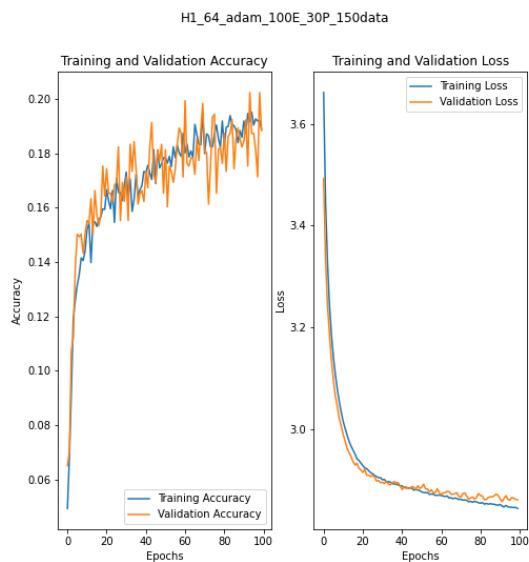


Figura 3.4: Exactitud (*accuracy*) y pérdida (*loss*) durante el entrenamiento del clasificador basado en datos geográficos en los conjuntos de entrenamiento (azul) y validación (naranja). Se empleó un optimizador Adam y una tasa de aprendizaje del 0.001 para entrenar en 200 épocas.

La agregación de datos en un modelo que combine las imágenes y los datos geográficos podría suponer una mejora al 70 % obtenido con el clasificador basado en imágenes, que actualmente tiene limitaciones para una aplicación real. Se ha explorado la posibilidad de combinar ambos modelos en un clasificador con dos entradas de datos y una salida en TensorFlow (ver apéndice E). El método más adecuado para este caso parece ser construir un *stack*, sin embargo, la implementación, ajuste y evaluación del mismo quedan fuera del alcance de este proyecto.

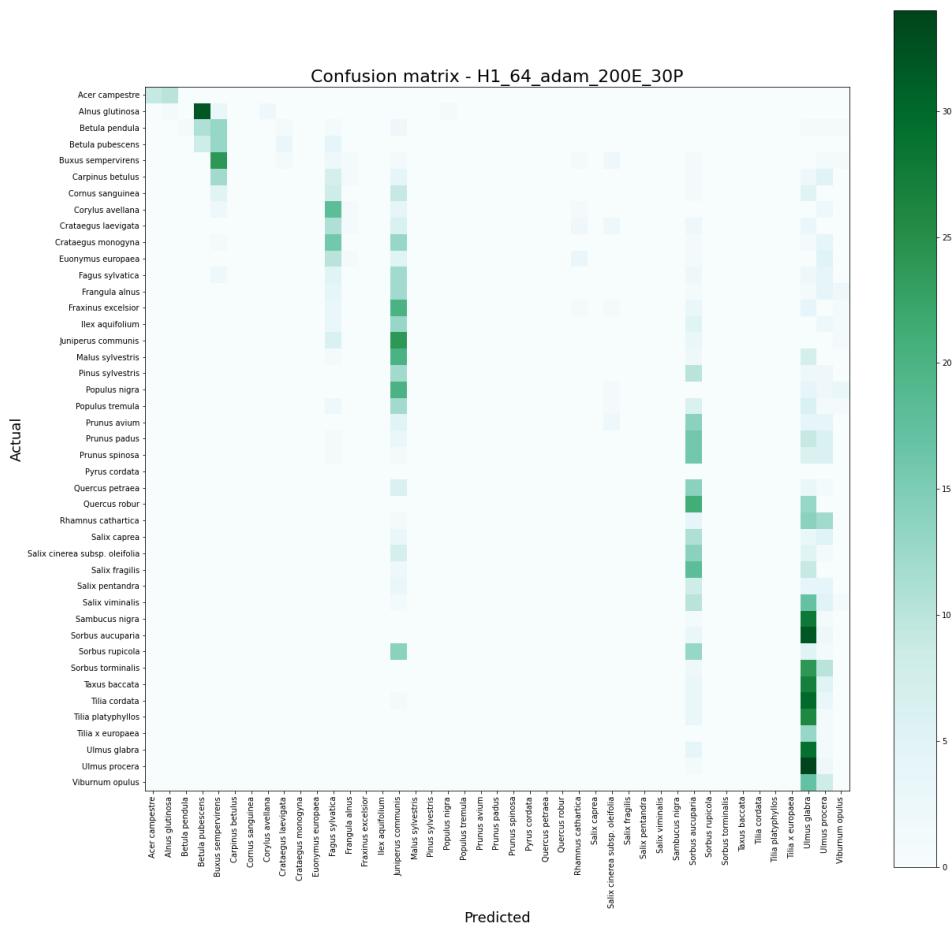


Figura 3.5: Matriz de confusión para las predicciones del clasificador basado en datos geográficos. Muestra el número de casos predichos en cada clase en comparación con la clase real a la que pertenece. Se empleó un optimizador Adam y una tasa de aprendizaje del 0.001 para entrenar en 100 épocas.

Capítulo 4

Conclusiones

1. Se ha desarrollado un *pipeline* en Python para la descarga y pre-procesado de imágenes de herbario del repositorio público de biodiversidad GBIF.
2. Asimismo, se ha desarrollado un *pipeline* análogo para la descarga y pre-procesado de datos geográficos.
3. A partir del *pipeline* de descarga y pre-procesado de digitalizaciones de herbario se ha obtenido un juego de datos de 11405 imágenes pertenecientes a 43 especies de árboles nativos de Reino Unido.
4. Se ha construido un clasificador de imágenes de herbario de árboles nativos de Reino Unido con una exactitud de en torno al 70 % en 1000 épocas. Dicho clasificador se basa en una arquitectura CNN VGG16 en la que se ha realizado transferencia de aprendizaje con el juego de datos de ImageNet para los tres primeros bloques de capas.
5. Se ha obtenido un clasificador de coordenadas geográficas de especies de árboles nativos de Reino Unido con una exactitud menor del 20 %. El clasificador es una red neuronal con una capa oculta.
6. Existe una limitación de cada modelo por separado para obtener un clasificador fiable, por lo que se propone una combinación de ambos clasificadores.
7. El método de descarga, preprocesado y construcción de los clasificadores diseñado es extrapolable a otros grupos de especies vegetales, por lo que puede suponer una facilitación para el desarrollo de clasificadores en este área.

Glosario

AlexNet es una arquitectura de CNN ganadora del certamen de 2012 de ImageNet. Consiste en cinco capas convolucionales y tres completamente conectadas, con capas de agrupamiento y *dropout* en dos de las capas completamente conectadas. Fue la CNN de mayor tamaño empleada hasta la fecha y fue aplicada a la clasificación de un imágenes a gran escala con un éxito considerablemente mayor que el resto de modelos existentes.

API (Application Programming Interface) es una herramienta que se basa en el ciclo de solicitud/respuesta del protocolo HTTP (Hyper-Text Transfer Protocol) y que sirve de intermediaria entre dos aplicaciones, definiendo una serie de reglas de intercambio de información. Las API emplean, por tanto, las URLs (Uniform Resource Locator, direcciones web) para ese intercambio de información. Existen dos arquitecturas principales de una API: SOAP y REST (Representational State Transfer), entre las cuales REST es la que ha tenido una tasa de adopción mayor en los últimos años.

aumentación de imágenes en el contexto del entrenamiento de un algoritmo de clasificación, es un incremento del número de ejemplos de un conjunto de imágenes de entrenamiento aplicando transformaciones aleatorias a las imágenes originales con el fin de mejorar la exactitud y prevenir el *overfitting* del modelo.

capa en una red neuronal, es el conjunto de neuronas que integran valores de entrada de otras neuronas (o de entrada) y devuelven valores de salida, pero que no están conectadas entre sí.

capa completamente conectada (fully connected layer) en una red neuronal de aprendizaje profundo, es una serie de neuronas donde cada una recibe las entradas de todas las neuronas de la capa anterior, multiplicado por sus pesos, sumados y transformados por la función de activación.

capa convolucional en una red neuronal, es una de serie de neuronas cuya característica principal es que aplica un filtro (el kernel) a cada uno de los elementos de una matriz. El kernel consiste en otra matriz (generalmente de pequeño tamaño, 2x3, 3x2, 3x3 ...) con un conjunto de números reales fijos. En caso de que la matriz de entrada sea una image, cada píxel de la imagen original se multiplica por la matriz del kernel y el resultado se suma para generar otro valor de píxel para la imagen transformada. Cada vez que el filtro se aplica a todos los píxeles de una imagen se llama convolución. El rendimiento de la extracción de características de una imagen depende, en parte, de los valores del kernel y de la concatenación de capas convolucionales.

capa de agrupamiento (Pooling layer) en una red neuronal de aprendizaje profundo, es una técnica mediante la cual se reduce la dimensión de una matriz. mediante la aplicación de un filtro (una matriz n x n) a los elementos de dicha matriz. El filtro se desliza a través de la matriz en una cantidad fija llamado paso (*stride*). En el caso de que las matrices representen los píxeles de una imagen, un filtro de 3x3 píxeles tomaría los 3x3 píxeles en la parte superior izquierda de la imagen de entrada, les aplicaría un

cálculo como conjunto, dando como resultado un único valor de píxel para la imagen de salida. Un tipo de capa de agrupamiento es la capa de agrupamiento máximo 2D.

capa de agrupamiento máximo 2D (Maxpooling layer) en una red neuronal de aprendizaje profundo, es una capa de agrupamiento donde el cálculo realizado por el filtro es seleccionar el valor máximo de la matriz en cada paso del mismo. En la clasificación de imágenes, se suelen añadir después de una capa convolucional, de modo que se reduce la dimensión de la imagen y pasan a la siguiente capa los píxeles más activados (de mayor valor). Además, ayuda a reducir el *overfitting* del modelo y la carga computacional (y se puedan añadir más nodos).

capa de aplanamiento (flatten layer), en una red neuronal de aprendizaje profundo, es una serie de neuronas que reduce la dimensionalidad de las matrices de entrada. En la clasificación de imágenes, se suele emplear para convertir la matriz 2D de la imagen en una matriz 1D antes de pasarla a la capa de salida, dado que ésta ha de generar vectores 1D con las probabilidades de cada clase.

categorical_crossentropy en la librería Keras, es un tipo de función de pérdida indicada para implementarse en clasificadores de múltiples categorías.

CNN/ConvNet Convolutional Neural Network (Red Neuronal Convolutiva) es una red neuronal que contiene al menos una capa convolucional.

conjunto de entrenamiento en aprendizaje automático, es la colección de ejemplos y características que se emplea durante la fase de entrenamiento. En el caso de la clasificación con aprendizaje supervisado, el juego de datos contiene las etiquetas de cada clase. Suele contener entre el 60 y el 80 % del total de ejemplos en el juego de datos.

conjunto de test en aprendizaje automático, es la colección de ejemplos y características que se emplea durante la fase de prueba. En el caso de la clasificación con aprendizaje supervisado, el juego de datos se presenta sin las etiquetas de cada clase. Suele contener alrededor del 20 % del total de ejemplos en el juego de datos.

conjunto de validación en aprendizaje automático, es la colección de ejemplos y características que se emplea opcionalmente durante la fase de entrenamiento y que sirve para evaluar a tiempo real la exactitud de las predicciones del modelo. En el caso de la clasificación con aprendizaje supervisado, el juego de datos se presenta etiquetado. Suele contener alrededor del 20 % del total de ejemplos en el juego de datos.

Conv2D nomenclatura empleada en la librería Keras para la capa convolucional aplicada a una matriz 2D.

Dense nomenclatura empleada en la librería Keras para la capa de completamente conectada.

desbalance de clases en un conjunto de datos de entrenamiento, es el fenómeno de que existe una cantidad muy diferente de ejemplos entre clases y que puede producir un sesgo en la clasificación en favor de las que cuentan con más ejemplos..

dropout (stochastic dropout training) en una red neuronal de aprendizaje profundo, es una técnica empleada para evitar *overfitting*. Consiste en eliminar algunas de las neuronas (hacer que su valor sea cero) de las capas profundas de forma aleatoria y con una cierta probabilidad.

entrenamiento en aprendizaje automático, significa ajustar un modelo para predecir las clases o valores de un conjunto de datos (conjunto de entrenamiento). En el caso de las redes neuronales, se refiere a ajustar los valores de los pesos de los nodos cada vez que la red trata de clasificar o predecir el conjunto de datos de entrenamiento en cada época. Los pesos se modifican en función de un optimizador, que trata de minimizar la función de pérdida (la cual se emplea para calcular el gradiente). Los valores de los nuevos pesos se obtienen al multiplicar el gradiente por la tasa de aprendizaje..

exactitud (accuracy) en un algoritmo de clasificación de machine learning, es el número de ejemplos clasificados correctamente entre el número de ejemplos totales. Es un valor entre 0 y 1 o entre 0 y 100 si se presenta como porcentaje.

Flatten nomenclatura empleada en la librería Keras para la capa de aplanamiento.

función de activación en una red neuronal, realiza una operación sobre el conjunto de valores de entrada en un nodo antes de pasar el valor a la siguiente capa. Algunos tipos son la sigmoidea, la ReLu y la softmax.

función de pérdida en el entrenamiento de una red neuronal, mide el error de las predicción o clasificación.

GBIF (Global Biodiversity Information Facility) es un repositorio global de referencia de datos de biodiversidad ([15](#)).

GPU (Graphic Processor Unit). Es un circuito electrónico análogo a la CPU (Central Processing Unit) pero especializada en la manipulación de imágenes.

gradiente en una red neuronal, es el cálculo del error de la predicción en relación con el peso (la derivada del error dividido en la derivada del peso). Se multiplica por la tasa de aprendizaje para obtener los nuevos pesos.

ILSVRC ImageNet Large Scale Visual Recognition Challenge (ver ImageNet).

ImageNet (ImageNet Large Scale Visual Recognition Challenge): Es un concurso celebrado anualmente desde 2010 que marca el estado del arte en la clasificación y detección de objetos a partir de imágenes. Da nombre al juego de datos ImageNet, que contiene millones de imágenes anotadas con cientos de categorías y que también es una referencia en el campo. Es el sucesor del certamen PASCAL VOC ([20](#)).

MaxPooling2D nomenclatura empleada en la librería Keras para la capa de agrupamiento máximo 2D.

neurona/nodo en una red neuronal, son las unidades conceptuales que conforman las capas y que integran valores de las neuronas de las capas anteriores (o de los datos de entrada) a través de la función de activación y la transmiten a las neuronas de la siguiente capa (o a la salida).

normalización referido a un conjunto de datos, consiste en realizar una operación para que todos los valores estén dentro de un rango. Un tipo de normalización consiste en restar a cada dato el valor mínimo del conjunto y dividirlo entre la resta del máximo y el mínimo. Se emplea en machine learning para evitar que las predicciones y clasificaciones estén sesgadas a tener más en cuenta características cuyos valores varían en órdenes de magnitud superiores a los de otras características.

optimizador en el entrenamiento de una red neuronal, es un algoritmo que trata de minimizar la función de pérdida..

overfitting en un modelo de clasificación o predicción, ocurre cuando el modelo clasifica o predice con una cierta exactitud el conjunto de datos de entrenamiento, pero obtiene peores resultados cuando se emplean ejemplos nuevos. En este caso se dice que el modelo no generaliza bien.

PASCAL VOC (Pascal Visual Object Classes): Fue un concurso de clasificación de imágenes que tuvo lugar anualmente entre 2005 y 2012. El juego de datos empleado contaba con menos de 20000 imágenes anotadas en 20 categorías en su última edición y el método de clasificación ganador fue NUS_SCM, basado en SVM ([30](#)).

peso en una red neuronal, un peso es un número que multiplica el valor del nodo de entrada antes de pasarlo al nodo de salida en la siguiente capa. Un nodo de salida recibe los valores de cada uno de los nodos de entrada multiplicados por sus pesos, después de que han sido sumados y transformados por la función de activación.

prueba en aprendizaje automático, es el proceso que se realiza después del entrenamiento de un modelos, mediante el cual se evalúan las predicciones realizadas sobre un conjunto de datos sin etiquetar.

pérdida en redes neuronales, es una medida error en la predicción o clasificación que se emplea para ajustar los pesos.

re-escalado de píxeles es un tipo de normalización que consiste en dividir cada valor de píxel entre el valor de píxel máximo posible (255), de modo que todos los píxeles adquieran valores entre 0 y 1.

ReLU (Rectified Linear Units) es un tipo de función de activación típicamente empleada en CNN. Devuelve el valor cero cuando los valores de entrada son iguales o menores a cero y el mismo valor de salida que de entrada cuando éstos son positivos.

REST (Representational state transfer) es un tipo de API que ofrece una serie de recursos definidos en el punto final de una URL. Los recursos son los tipos de información que se ha decidido que estén disponibles.

SGD (Stochastic Gradient Descent, descenso por gradiente) es un tipo de optimizador.

softmax es un tipo de función de activación que convierte un vector de entrada en un vector de salida con probabilidades (la suma de todos sus valores es 1).

tamaño de lote (batch size) en el entrenamiento de una red neuronal, es la cantidad de ejemplos que se procesan a la vez. Si el tamaño del lote es igual al número de ejemplos en el conjunto de datos de entrenamiento, entonces se procesa un lote por cada época. Si el tamaño del lote es menor, entonces se procesan varios lotes por época. A mayor tamaño de lote, mayor es la velocidad del entrenamiento, pero a un coste computacional también mayor. Por el contrario, si es muy pequeño, el entrenamiento puede ser demasiado lento.

tasa de aprendizaje en una red neuronal, es un valor pequeño (normalmente de 0.01 a 0.001) por el cual se multiplica el gradiente para obtener los nuevos pesos en cada iteración e la red.

transferencia de aprendizaje en redes neuronales de aprendizaje profundo, es una práctica consistente en el uso de los pesos obtenidos tras entrenar un modelo con un juego de datos de gran tamaño, para aplicarlos a otro juego de datos (normalmente más reducido). Según se usen todos los pesos o sólo los de algunas capas, se trata de **extracción de características** (feature extraction) o de **ajuste** (fine-tuning).

VGG16 es una arquitectura de CNN ganadora del certamen de 2014 de ImageNet. Consiste una secuencia de capas convolucionales y de agrupamiento máximo con un filtro de tamaño de pixel de 2x2 (reduce la dimensión de la imagen a la mitad) distribuidas en cinco bloques de la forma 2+1, 2+1, 3+1, 3+1, 3+1. Su característica principal es que introdujo filtros convolucionales de pequeño tamaño ([26](#)).

VGG19 variante de VGG16 con la forma 2+1, 2+1, 4+1, 4+1, 4+1.

época en una red neuronal, es el número de iteraciones a través de la red para ajustar los pesos. Cuanto mayor sea el número de épocas, mejor será el rendimiento del modelo, hasta un límite.

Bibliografía

- (1) Sokal, R. R. (1963). THE PRINCIPLES AND PRACTICE OF NUMERICAL TAXONOMY. *TAXON* 12, 190-199.
- (2) Sneath, P. H. en *Bergey's Manual® of Systematic Bacteriology*, Brenner, D. J., Krieg, N. R., Staley, J. T. y Garrity, G. M., eds.; Springer US: Boston, MA, 2005, págs. 39-42.
- (3) Petrini, O. y Sieber, T. N. en *Systematics and Evolution*, McLaughlin, D. J., McLaughlin, E. G. y Lemke, P. A., eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2001, págs. 203-216.
- (4) Hearn, D. J. (2009). Shape analysis for the automated identification of plants from images of leaves. *TAXON* 58, 934-954.
- (5) Wäldchen, J. y Mäder, P. (2018). Plant Species Identification Using Computer Vision Techniques: A Systematic Literature Review. *Archives of Computational Methods in Engineering* 25, 507-543.
- (6) Kho, S. J., Manickam, S., Malek, S., Mosleh, M. y Dhillon, S. K. (2017). Automated plant identification using artificial neural network and support vector machine. *Frontiers in Life Science* 10, 98-107.
- (7) MacLeod, N., Benfield, M. y Culverhouse, P. (2010). Time to automate identification. *Nature* 467, 154-155.
- (8) Van Horn, G., Mac Aodha, O., Song, Y., Cui, Y., Sun, C., Shepard, A., Adam, H., Perona, P. y Belongie, S. en *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE: Salt Lake City, UT, 2018, págs. 8769-8778.
- (9) James, S. A., Soltis, P. S., Belbin, L., Chapman, A. D., Nelson, G., Paul, D. L. y Collins, M. (2018). Herbarium data: Global biodiversity and societal botanical needs for novel research. *Applications in Plant Sciences* 6, e1024.
- (10) Younis, S., Weiland, C., Hoehndorf, R., Dressler, S., Hickler, T., Seeger, B. y Schmidt, M. (2018). Taxon and trait recognition from digitized herbarium specimens using deep convolutional neural networks. *Botany Letters* 165, 377-383.
- (11) Lee, S. H., Chan, C. S., Mayo, S. J. y Remagnino, P. (2017). How deep learning extracts and learns leaf features for plant classification. *Pattern Recognition* 71, 1-13.
- (12) Al-Qurran, R., Al-Ayyoub, M. y Shatnawi, A. en *2018 International Arab Conference on Information Technology (ACIT)*, 2018 International Arab Conference on Information Technology (ACIT), IEEE: Werdanye, Lebanon, 2018, págs. 1-5.
- (13) Carranza-Rojas, J., Goeau, H., Bonnet, P., Mata-Montero, E. y Joly, A. (2017). Going deeper in the automated identification of Herbarium specimens. *BMC Evolutionary Biology* 17, 181.
- (14) Wäldchen, J. y Mäder, P. (2018). Machine learning for image based species identification. *Methods in Ecology and Evolution* 9, ed. por Cooper, N., 2216-2225.
- (15) GBIF. <https://www.gbif.org/> (visitado 23 de jun. de 2020).

- (16) Brahimí, M., Arsenovic, M., Laraba, S., Sladojevic, S., Boukhalfa, K. y Moussaoui, A. en *Human and Machine Learning*, Zhou, J. y Chen, F., eds.; Springer International Publishing: Cham, 2018, págs. 93-117.
- (17) Chamberlain, S. pygbif: Python client for GBIF., ver. 0.4.0.
- (18) Reitz, K. requests: Python HTTP for Humans., ver. 2.24.0.
- (19) Johnson, J. M. y Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data* 6, 27.
- (20) Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. y Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 211-252.
- (21) Krizhevsky, A., Sutskever, I. e Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM* 60, 84-90.
- (22) Fukushima, K. y Miyake, S. en *Competition and Cooperation in Neural Nets*, Amari, S.-i. y Arbib, M. A., eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 1982; vol. 45, págs. 267-285.
- (23) LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. y Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* 1, 541-551.
- (24) Khan, A., Sohail, A., Zahoor, U. y Qureshi, A. S. (2020). A Survey of the Recent Architectures of Deep Convolutional Neural Networks. *Artificial Intelligence Review*, DOI: [10.1007/s10462-020-09825-6](https://doi.org/10.1007/s10462-020-09825-6).
- (25) Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. y Rabinovich, A. (2014). Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*.
- (26) Simonyan, K. y Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*.
- (27) He, K., Zhang, X., Ren, S. y Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*.
- (28) Huang, G., Liu, Z., van der Maaten, L. y Weinberger, K. Q. (2018). Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*.
- (29) Cracknell, M. J. y Reading, A. M. (2014). Geological mapping using remote sensing data: A comparison of five machine learning algorithms, their response to variations in the spatial distribution of training data and the use of explicit spatial information. *Computers & Geosciences* 63, 22-33.
- (30) Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J. y Zisserman, A. (2015). The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision* 111, 98-136.
- (31) Trust, W. UK Native Trees., Woodland Trust <https://www.woodlandtrust.org.uk/trees-woods-and-wildlife/british-trees/native-trees/> (visitado 30 de mar. de 2020).

Apéndice A

Especies de la búsqueda

Lista de árboles nativos de Reino Unido ((*31*))

- *Alnus glutinosa*
- *Frangula alnus*
- *Fraxinus excelsior*
- *Populus tremula*
- *Fagus sylvatica*
- *Betula pubescens*
- *Betula pendula*
- *Prunus spinosa*
- *Buxus sempervirens*
- *Rhamnus cathartica*
- *Prunus padus*
- *Prunus avium*
- *Malus sylvestris*
- *Cornus sanguinea*
- *Sambucus nigra*
- *Ulmus procera*
- *Ulmus glabra*
- *Viburnum opulus*
- *Crataegus monogyna*
- *Crataegus laevigata*
- *Corylus avellana*
- *Ilex aquifolium*
- *Carpinus betulus*
- *Juniperus communis*
- *Tilia x europaea*
- *Tilia platyphyllos*
- *Tilia cordata*
- *Acer campestre*
- *Quercus robur*
- *Quercus petraea*
- *Pyrus cordata*
- *Pinus sylvestris*
- *Populus nigra*
- *Sorbus aucuparia*
- *Euonymus europaea*
- *Sorbus aria*
- *Sorbus arranensis*
- *Sorbus rupicola*
- *Sorbus torminalis*
- *Salix pentandra*
- *Salix fragilis*
- *Salix caprea*
- *Salix cinerea* subsp. *oleifolia*
- *Salix viminalis*
- *Salix alba*
- *Taxus baccata*

Apéndice B

Filtros de la descarga

Parámetro	Significado	Imagen	Geodato
species_name	Lista de especies	Especies en A	Especies en A
search_name	Nombre de la búsqueda		
media_type	Formato del medio	StillImage	
country	País de origen		GB
has_coordinate	Si tiene coordenada		True
kingdom	Reino de la naturaleza		
basis_of_record	Estado del especímen	PRESERVED_SPECIMEN	
institution_code	El código de la institución		
limit	Registros/especie máximo	500	600

CUADRO B.1: Filtros aplicados en la descarga de imágenes y datos geográficos (geodato) de GBIF.

Apéndice C

Arquitecturas VGG16 y VGG19

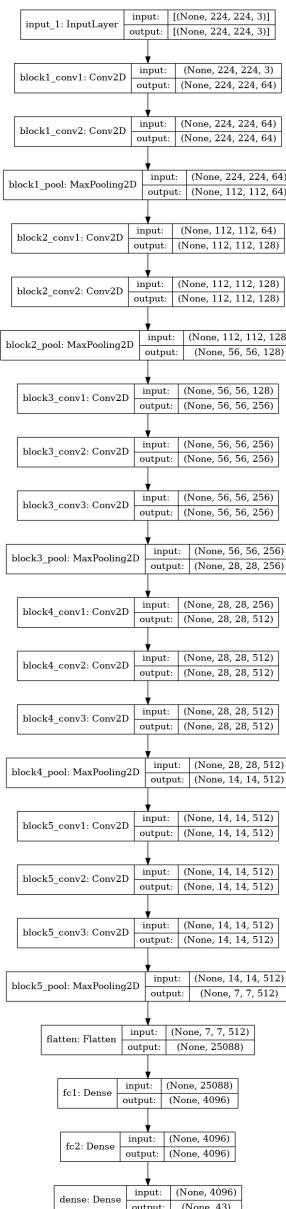


Figura C.1: Arquitectura del modelo VGG16, basada en secuencias de capas convolucionales (Conv2D) y de agrupación (Maxpooling2D) agrupadas en cinco bloques (2+1,2+1,3+1,3+1,3+1).

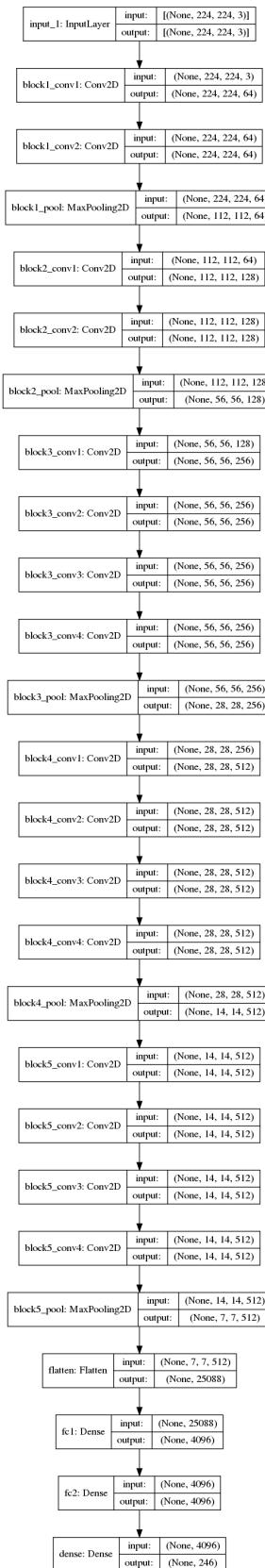


Figura C.2: Arquitectura del modelo VGG19, basada en secuencias de capas convolucionales (Conv2D) y de agrupación (Maxpooling2D) agrupadas en cinco bloques (2+1,2+1,4+1,4+1,4+1).

Apéndice D

Código del módulo *data*

D.1. Obtención de datos

D.1.1. Módulos ejecutables desde el terminal

Imágenes

data/images/request/image_request.py

```

1 import requests
2 import urllib.request
3 import os
4 import pandas as pd
5 import hashlib
6 import pdb
7 from data.storage_handler import stop_if_size
8 from data.common_api_request import get_taxon_key, get_occurrence_key, species_without_speciesKey
9 from data.config import images_filter, species_list
10 from data.search import filter_hash
11
12
13 def get_occurrence_image(species_occurrences_keys, folder):
14     """
15     Download image for the occurrences of the given species.
16     Image naming format:
17         taxon key + occurrence key + occurrence number
18         example: 2878688_1056970865_1.jpg
19     """
20     base_url = "https://api.gbif.org/v1/"
21     has_image_dict = {}
22     count_species = 1
23     for species_name, occurrences in species_occurrences_keys.items():
24         taxon_key = species_taxon_key[species_name]
25         species_words = species_name.split(" ")
26         species_class = '_'.join(species_words)
27         current_folder = folder+"/"+species_class
28         count_occurrence = 1
29         for occurrence in range(0,(len(occurrences))):
30             occurrence_key = occurrences[occurrence]
31             response = requests.get(f"{base_url}occurrence/{occurrence_key}")
32             if response.status_code == 200:
33                 occurrence_result = response.json()
34                 try:
35                     occurrence_url = occurrence_result["media"][0]["identifier"]
36                     try:
37                         # Create folder where to store images
38                         if not os.path.exists(current_folder):
39                             os.makedirs(current_folder)
40                         # Name and download file
41                         image_path = f"{current_folder}/{species_class}_{occurrence_key}.jpg"
42                         urllib.request.urlretrieve(occurrence_url, image_path)
43                         has_image_dict[occurrence_key] = "1"
44                         stop_if_size(10)
45                         print(f'{species_name}[{sp.{count_species}}/{len(species_occurrences_keys)}]: {'
46                         count_occurrence}/{len(occurrences)} images downloaded.')
47                         count_occurrence +=1
48                     except:
49                         # urlretrieve fails
50                         has_image_dict[occurrence_key] = "0"
51                     except:
52                         # parsing request result fails
53                         has_image_dict[occurrence_key] = "0"
54                 elif response.status_code == 404:
55                     print('Error 404: Page not found.')
56                 else:
57                     print("Error. Undetermined status code.")
58             count_species +=1
59     return has_image_dict
60
61 def get_results_table(species_occurrences_keys, occurrence_has_image):

```

```

62 """
63     Return a data frame with:
64     - Species names.
65     - Species keys (speciesKey/taxonKey)
66     - Occurrences keys (key/gbifID)
67     - Information about if the image has been downloaded (0 or 1)
68 """
69 column_names = ["species_name", "taxon_key", "occurrence_key", "has_image"]
70 df = pd.DataFrame(columns = column_names)
71 for species_name, occurrences in species_occurrences_keys.items():
72     for occurrence in range(0,(len(occurrences))):
73         taxon_key = species_taxon_key[species_name]
74         data_in_row = []
75         data_in_row.extend([species_name,taxon_key, occurrences[occurrence], ""])
76         new_row = pd.DataFrame([data_in_row], columns=column_names)
77         df = df.append(new_row, ignore_index = True)
78     for occurrence_key, has_image in occurrence_has_image.items():
79         df.loc[df["occurrence_key"] == occurrence_key, ["has_image"]] = has_image
80 df.sort_values(by=["taxon_key", "occurrence_key"])
81 return df
82
83
84 def image_result_to_csv(df, filter_hash):
85 """
86     Create a CSV file with the results
87     of the data request to GBIF
88     and name it with a hash of the filter applied.
89     Results from different filters
90     are stored in a new file, while results
91     from the same filter are overwritten.
92 """
93 folder = "data/images/request/outputs"
94 if not os.path.exists(folder):
95     os.makedirs(folder)
96 csv_file_name = f'{folder}/{filter_hash}_request_results.csv'
97 df.to_csv(csv_file_name, sep = ",", header = True, index = None, encoding="utf-8")
98 print(f"+{csv_file_name} file created.")
99
100
101 def open_filter_report(filter_hashed):
102 """
103     Open a text file with the following naming format:
104     md5hash_request_filter.txt
105     Example: 2020-03-31T22:00:00_request_summary.txt
106 """
107 folder = "data/images/request/outputs"
108 if not os.path.exists(folder):
109     os.makedirs(folder)
110 text_file_name = f'{folder}/{filter_hashed}_image_filter.txt'
111 print(f'{text_file_name} file created.')
112 return open(text_file_name, "w")
113
114
115 if __name__ == "__main__":
116
117     # 1- Import search parameters
118     # Species input
119     species_list = species_list.species_list
120     # Search name
121     search_name = images_filter.search_name
122     # Filter parameters
123     media_type = images_filter.media_type
124     country = images_filter.country
125     has_coordinate = images_filter.has_coordinate
126     kingdom = images_filter.kingdom
127     basis_of_record = images_filter.basis_of_record
128     institution_code = images_filter.institution_code
129     limit = images_filter.limit
130     ##########
131
132     # 2- Handle filter parameters
133     filter = {"mediaType": media_type, "country": country, "hasCoordinate": has_coordinate, "kingdom": kingdom,
134     "basisOfRecord": basis_of_record, "institutionCode": institution_code, "limit": limit}
135     filter_information = images_filter.filter_information()
136     ## Hash the filter information + species list string to use it for naming the results file
137     filter_hash = filter_hash(images_filter, species_list)
138     print(f"\nStarting request '{search_name}' identified by: {filter_hash}.")
139
140     ## Save filter and species information to text file
141     save_filter = open_filter_report(filter_hash)
142     save_filter.write(f"{search_name}\n")
143     save_filter.write(f"{str(filter_information)}\n")
144     save_filter.write(f"Image download limit: {limit}\n\n")
145     save_filter.write("Input species list:\n")
146     for species in species_list:
147         save_filter.write(f"{species}\n")
148     save_filter.close()
149
150     # 3- Get species keys (same as taxon key)
151     species_taxon_key = get_taxon_key(species_list)
152     ## Check if all species entered have a taxon key
153     species_without_speciesKey(species_list, species_taxon_key)
154
155     # 4- Get occurrences keys
156     species_occurrences_keys = get_occurrence_key(species_taxon_key, filter)
157
158     # 5- Get images from occurrences

```

```

158     folder = "data/images/request/outputs/" + filter_hash + "_images"
159     occurrence_has_image = get_occurrence_image(species_occurrences_keys, folder)
160
161     # 6- Save results information for the applied filter
162     ## Save results summary to csv file
163     result_df = get_results_table(species_occurrences_keys, occurrence_has_image)
164     image_result_to_csv(result_df, filter_hash)

```

data/images/request/image_request_summary.py

```

1 import pandas as pd
2 from data.file_handler import create_dataframe_from_csv
3
4 def get_summary_of_results(path_to_file):
5     """
6         Given the path to the CSV file with the results of the request:
7         Returns a data frame with a summary of
8             - Number of occurrences species
9             - Number of images per species
10    """
11    request_results = create_dataframe_from_csv(path_to_file)
12    occurrences_per_taxon = request_results.groupby("species_name")["taxon_key"].value_counts().reset_index(
13        name="occurrence_count")
14    images_per_taxon = request_results[request_results.has_image == 1].groupby("species_name")["taxon_key"].value_counts().reset_index(name="image_count")
15    images_per_taxon = images_per_taxon.iloc[:, -1]
16    return occurrences_per_taxon.join(images_per_taxon)
17
18 def save_summary_as_csv(path_to_file):
19     """
20         Given the path to the CSV file with the results of the request
21         creates a summary as a data frame object using get_summary_of_results()
22         and saves it to the request_reports folders as a CSV.
23     """
24     summary = get_summary_of_results(path_to_file)
25     file_name = path_to_file.split("/")[-1]
26     filter_hash = file_name.split("_")[0]
27     summary_file_name = "data/images/request/outputs/" + filter_hash + "_summary.csv"
28     summary.to_csv(summary_file_name)
29     print(f"File {summary_file_name} created.")
30
31 if __name__ == "__main__":
32     path_to_file = input("Enter file name (request_results.csv, full path): ")
33     save_summary_as_csv(path_to_file)

```

data/images/request/plot_image_per_species.py

```

1 from matplotlib import pyplot as plt
2 import pandas as pd
3
4
5 def get_summary_csv():
6     path_to_file = input("Enter file name (summary.csv, full path): ")
7     return pd.read_csv(path_to_file, sep=",", encoding="utf-8")
8
9
10 def get_species_list(df):
11     return df["species_name"].tolist()
12
13
14 def get_image_count(df):
15     return df["image_count"].tolist()
16
17
18 def plot_image_count(image_count, species_name):
19     """
20         Plot number of images per class.
21     """
22     image_count = list(reversed(image_count))
23     species_name = list(reversed(species_name))
24     plt.figure(figsize=(10,10))
25     plt.barh(species_name, image_count, color="#7cd9b7", height=0.8, edgecolor="grey")
26     plt.title("Number of images per class")
27     # x axis, species
28     xint = range(0,(max(image_count)+1))
29     plt.xticks(xint)
30     plt.xlabel("Number of images")
31     # y axis, image count
32     plt.ylabel("Species", rotation="vertical")
33     plt.yticks(fontsize=8)
34     # save file
35     plt.savefig("data/images/image_request/request_reports/images_per_species.png")
36
37 if __name__ == "__main__":
38     summary = get_summary_csv()
39     species_name = get_species_list(summary)
40     image_count = get_image_count(summary)
41     plot_image_count(image_count, species_name)

```

Datos geográficos

data/geodata/request/geodata_request.py

```

1 import requests
2 import urllib.request
3 import os
4 import pandas as pd
5 import hashlib
6 import pdb
7 from data.common_api_request import get_taxon_key, get_occurrence_key, species_without_speciesKey
8 from data.config import geodata_filter, species_list
9 from data.search import filter_hash
10
11 def get_occurrence_data(species_occurrences_keys):
12     """
13         Given the species occurrence key
14         return a data frame with a series of geographical parameters
15         associated to the occurrence.
16     """
17     base_url = "https://api.gbif.org/v1/"
18     column_names = ["species_name", "taxon_key", "occurrence_key", "basis_of_record", "institution_code",
19                     "coordinate_system", "decimal_longitude", "decimal_latitude", "coordinate_uncertainty", "elevation", "date",
20                     "issues"]
21     df = pd.DataFrame(columns = column_names)
22     count_species = 1
23     for species_name, occurrences in species_occurrences_keys.items():
24         taxon_key = species_taxon_key[species_name]
25         for occurrence in range(0,(len(occurrences))):
26             occurrence_key = occurrences[occurrence]
27             response = requests.get(f"{base_url}occurrence/{occurrence_key}")
28             if response.status_code == 200:
29                 occurrence_result = response.json()
30                 try:
31                     basis_of_record = occurrence_result["basisOfRecord"]
32                 except:
33                     basis_of_record = ""
34                 pass
35                 try:
36                     institution_code = occurrence_result["institutionCode"]
37                 except:
38                     institution_code = ""
39                 pass
40                 try:
41                     coordinate_system = occurrence_result["geodeticDatum"]
42                 except:
43                     coordinate_system = ""
44                 pass
45                 try:
46                     decimal_longitude = occurrence_result["decimalLongitude"]
47                 except:
48                     decimal_longitude = ""
49                 pass
50                 try:
51                     decimal_latitude = occurrence_result["decimalLatitude"]
52                 except:
53                     decimal_latitude = ""
54                 pass
55                 try:
56                     coordinate_uncertainty = occurrence_result["coordinateUncertaintyInMeters"]
57                 except:
58                     coordinate_uncertainty = ""
59                 pass
60                 try:
61                     elevation = occurrence_result["elevation"]
62                 except:
63                     elevation = ""
64                 pass
65                 try:
66                     date = occurrence_result["eventDate"]
67                 except:
68                     date = ""
69                 pass
70                 try:
71                     issues = occurrence_result["issues"]
72                 except:
73                     issues = ""
74                 pass
75                 data_in_row = []
76                 data_in_row.extend([species_name, taxon_key, occurrences[occurrence], basis_of_record,
77                                     institution_code, coordinate_system, decimal_longitude, decimal_latitude, coordinate_uncertainty,
78                                     elevation, date, issues])
79                 new_row = pd.DataFrame([data_in_row], columns=column_names)
80                 df = df.append(new_row, ignore_index = True)
81             elif response.status_code == 404:
82                 print('Error 404: Page not found.')
83             else:
84                 print("Error. Undetermined status code.")
85             print(f'{species_name} [{sp.count_species}/{len(species_occurrences_keys)}]: records downloaded.')
86             count_species +=1
87     return df
88
89
90 if __name__ == "__main__":
91     # 1- Import search parameters

```

```

89     # Species input
90     species_list = species_list.species_list
91     # Search name
92     search_name = geodata_filter.search_name
93     # Filter parameters
94     media_type = geodata_filter.media_type
95     country = geodata_filter.country
96     has_coordinate = geodata_filter.has_coordinate
97     kingdom = geodata_filter.kingdom
98     basis_of_record = geodata_filter.basis_of_record
99     institution_code = geodata_filter.institution_code
100    limit = geodata_filter.limit
101    ##########
102
103    # 2- Handle filter parameters
104    filter = {"mediaType": media_type, "country": country, "hasCoordinate": has_coordinate, "kingdom": kingdom,
105              "basisOfRecord": basis_of_record, "institutionCode": institution_code, "limit": limit}
106    filter_information = geodata_filter.filter_information()
107    ## Hash the filter information + species list string to use it for naming the results file
108    filter_hash = filter_hash(geodata_filter, species_list)
109    print(f"Starting request '{search_name}' identified by: {filter_hash}.")
110
111    # 3- Save filter and species information to text file
112    folder = "data/geodata/request/outputs"
113    if not os.path.exists(folder):
114        os.makedirs(folder)
115    text_file_name = f"{folder}/{filter_hash}_geodata_filter.txt"
116    save_filter = open(text_file_name, "w")
117    save_filter.write(f"{search_name}\n")
118    save_filter.write(f"{str(filter_information)}\n")
119    save_filter.write("Records download limit: {limit}\n\n")
120    save_filter.write("Input species list:\n")
121    for species in species_list:
122        save_filter.write(f"{species}\n")
123    save_filter.close()
124
125    # 4- Get species keys (same as taxon key)
126    species_taxon_key = get_taxon_key(species_list)
127    ## Check if all species entered have a taxon key
128    species_without_speciesKey(species_list, species_taxon_key)
129
130    # 5- Get occurrences keys
131    species_occurrences_keys = get_occurrence_key(species_taxon_key, filter)
132
133    # 6- Get occurrence data into a CSV file
134    save_dir = "data/geodata/request/outputs"
135    if not os.path.exists(save_dir):
136        os.makedirs(save_dir, exist_ok=True)
137    occurrence_data_table = get_occurrence_data(species_occurrences_keys)
138    csv_file_name = "data/geodata/request/outputs/" + filter_hash + ".geodata.csv"
139    occurrence_data_table.to_csv(csv_file_name, sep = ",", header = True, index = None, encoding="utf-8")

```

D.1.2. Módulos comunes

data/config.py

```

1  from data.search import SearchFilter, Species
2  from data.image import ImageDimension
3  from data.filters import get_filter_str_or_bool, get_filter_int, get_species_list
4
5
6  # SEARCH FILTERS #####
7  images_filter = SearchFilter(
8      search_name = get_filter_str_or_bool("search_name"),
9      media_type = get_filter_str_or_bool("media_type"),
10     country = get_filter_str_or_bool("country"),
11     has_coordinate = get_filter_str_or_bool("has_coordinate"),
12     kingdom = get_filter_str_or_bool("kingdom"),
13     basis_of_record = get_filter_str_or_bool("basis_of_record"),
14     institution_code = get_filter_str_or_bool("institution_code"),
15     limit = get_filter_int("limit")
16 )
17 geodata_filter = SearchFilter(
18     search_name = get_filter_str_or_bool("search_name"),
19     media_type = get_filter_str_or_bool("media_type"),
20     country = get_filter_str_or_bool("country"),
21     has_coordinate = get_filter_str_or_bool("has_coordinate"),
22     kingdom = get_filter_str_or_bool("kingdom"),
23     basis_of_record = get_filter_str_or_bool("basis_of_record"),
24     institution_code = get_filter_str_or_bool("institution_code"),
25     limit = get_filter_int("limit")
26 )
27
28 species_list = Species(species_list = get_species_list())
29 #####
30
31 # IMAGE PREPROCESSING #####
32
33 cropping = ImageDimension(
34     left = 200,
35     top = 400,
36     right = 950,

```

```

37     bottom = 1250,
38     width = "",
39     height = ""
40 )
41
42 resizing = ImageDimension(
43     left = "",
44     top = "",
45     right = "",
46     bottom = "",
47     width = 1200,
48     height = 2000
49 )

```

data/filters.py

```

1 import pandas as pd
2 import argparse
3 import sys
4
5
6 def get_df_from_csv():
7     """
8     Return a data frame object
9     given a path to a CSV
10    """
11    parser = argparse.ArgumentParser(prog = "Get data request filter", usage = "Given a CSV with filter
12        parameters, extract their values individually. Example: python3 -m data.filter data/sample_inputs/
13        image/filter.csv")
14    parser.add_argument('src_path', type=str, help = "path to the CSV file with the species names. Example:
15        data/inputs/species_list.csv")
16    arg = parser.parse_args()
17    src_path = arg.src_path
18    df = pd.read_csv(src_path + "", sep=",", encoding="utf-8")
19    return df.where(df.notnull(), None)
20
21
22 def get_species_list():
23     """Return a list of species"""
24     df = get_df_from_csv()
25     first_column = df.columns[0]
26     result = df[first_column].tolist()
27     if result is not None:
28         return result
29     else:
30         print("You must enter at least one species name in the first column.")
31
32 def get_filter_str_or_bool(filter_parameter):
33     """Return a string or a boolean with the result of the filter parameter"""
34     df = get_df_from_csv()
35     result = df[filter_parameter].tolist()[0]
36     if result is None:
37         return ""
38     else:
39         return result
40
41 def get_filter_int(filter_parameter):
42     """Return an integer with the result of the filter parameter"""
43     df = get_df_from_csv()
44     result = int(df[filter_parameter].tolist()[0])
45     if result is None:
46         return ""
47     else:
48         return result
49
50 if __name__ == "__main__":
51     print(f'species list: {get_species_list()}')
52     print(f'search_name: {get_filter_str_or_bool("search_name")}')
53     print(f'media_type: {get_filter_str_or_bool("media_type")}')
54     print(f'country: {get_filter_str_or_bool("country")}')
55     print(f'has_coordinate: {get_filter_str_or_bool("has_coordinate")}')
56     print(f'basis_of_record: {get_filter_str_or_bool("basis_of_record")}')
57     print(f'institution_code: {get_filter_str_or_bool("institution_code")}')
58     print(f'limit: {get_filter_int("limit")}')

```

data/search.py

```

1 import hashlib
2
3
4 class SearchFilter(object):
5     """
6     Search parameters in GBIF API.
7     """
8     def __init__(self, search_name, media_type, country, has_coordinate, kingdom, basis_of_record,
9                  institution_code, limit):
10         self.search_name = search_name
11         self.media_type = media_type
12         self.country = country
13         self.has_coordinate = has_coordinate # True/False
14         self.kingdom = kingdom # Plantae
15         self.basis_of_record = basis_of_record
16         self.institution_code = institution_code # K (RBG Kew)
17         self.limit = limit

```

```

17     def filter_information(self):
18         """
19             Return the filter information to add to the filter text file
20             and to form the filter identifier.
21         """
22         return f"""
23             Filters:
24             mediaType: {self.media_type}
25             country: {self.country}
26             hasCoordinate: {self.has_coordinate}
27             kingdom:{self.kingdom}
28             basisOfRecord: {self.basis_of_record}
29             institutionCode: {self.institution_code}
30             """
31
32
33
34     class Species(object):
35         """
36             Species list used in the search.
37         """
38         def __init__(self, species_list):
39             self.species_list = species_list
40
41
42     def filter_hash(search_filter, species_list):
43         """
44             Return an alphanumeric search identifier by combining:
45             - Search filter.
46             - Species list.
47         """
48         filter_information = search_filter.filter_information()
49         filter_and_species_information = filter_information + str(species_list)
50         return hashlib.md5(str.encode(filter_and_species_information)).hexdigest()

```

data/image.py

```

1     class ImageDimension(object):
2         """
3             Parameters related to image preprocessing.
4         """
5         def __init__(self, left, top, right, bottom, width, height):
6             self.left = left
7             self.top = top
8             self.right = right
9             self.bottom = bottom
10            self.width = width
11            self.height = height
12
13        def coordinates(self):
14            """
15                Return a list with coordinates for area selection.
16            """
17            return (self.left, self.top, self.right, self.bottom)
18
19        def image_size(self):
20            """
21                Return a list with width and height for area selection.
22            """
23            return (self.width, self.height)
24
25
26
27
28
29     class Species(object):
30         """
31             Species list used in the search.
32         """
33         def __init__(self, species_list):
34             self.species_list = species_list
35
36
37     def filter_hash(search_filter, species_list):
38         """
39             Return an alphanumeric search identifier by combining:
40             - Search filter.
41             - Species list.
42         """
43         filter_information = search_filter.filter_information()
44         filter_and_species_information = filter_information + str(species_list)
45         return hashlib.md5(str.encode(filter_and_species_information)).hexdigest()

```

data/common_api_request.py

```

1     import requests
2     import urllib.request
3     import os
4     import pandas as pd
5
6
7     def get_taxon_key(species_list):
8         """
8             Get the GBIF taxon key for a given species name
9             using its API.
10            Return a dictionary for the species in the list passed as argument.
11

```

```

12 """
13     print("\nGetting GBIF taxon key for species in the list.")
14     base_url = "https://api.gbif.org/v1/"
15     resource = "species/match"
16     parameter = "name"
17     species_dict = {}
18     for species_name in species_list:
19         response = requests.get(f"{base_url}{resource}?{parameter}={species_name}")
20         if response.status_code == 200:
21             try:
22                 species = response.json()
23                 species_key = species["speciesKey"]
24                 species_dict[species_name] = species_key
25                 print(f"{species_name} identified by taxon key: {species_key}.")
26             except:
27                 print(f"Unable to find taxon key for {species_name}.")
28             pass
29         elif response.status_code == 404:
30             print('Error 404: Page not found.')
31         else:
32             print("Error. Undetermined status code.")
33     return species_dict
34
35
36 def get_occurrence_key(species_taxon_key, filter):
37 """
38     Get data from GBIF using its API
39 """
40     print("\nGetting GBIF occurrences keys for species in the list.")
41     base_url = "https://api.gbif.org/v1/"
42     occurrences_dict = {}
43     for species_name, taxon_key in species_taxon_key.items():
44         filter["taxonKey"] = taxon_key
45         response = requests.get(f"{base_url}occurrence/search", params=filter)
46         if response.status_code == 200:
47             occurrences = response.json()
48             occurrences_results = occurrences.get("results", {})
49             species_occurrence_dict = {}
50             for occurrence_no in range(0, len(occurrences_results)):
51                 occurrences_key = occurrences_results[occurrence_no].get("key")
52                 species_occurrence_dict[occurrence_no] = occurrences_key
53         elif response.status_code == 404:
54             print('Error 404: Page not found.')
55         else:
56             print("Error. Undetermined status code.")
57     occurrences_dict[species_name] = species_occurrence_dict
58     print(f"Number of occurrences for {species_name}: {len(occurrences_results)}")
59     return occurrences_dict
60
61
62 def species_without_speciesKey(species_list, species_taxon_key):
63 """
64     Return the species in the list with their corresponding taxon key
65     that are not in the original species list
66     because the taxon key was not found.
67 """
68     species_with_taxon_keys = list(species_taxon_key.keys())
69     diff = list(set(species_list) - set(species_with_taxon_keys))
70     if len(diff) == 0:
71         print("All species have a speciesKey.")
72     else:
73         print(f"WARNING: Species without a speciesKey:{diff}")
74     return diff

```

data/file_handler.py

```

1  from datetime import datetime
2  import os
3  import pandas as pd
4  import shutil
5
6
7
8  def get_timestamp():
9      """Get date and time for naming the files"""
10     timestamp = datetime.now()
11     timestamp_iso = timestamp.isoformat()
12     timestamp_iso_seconds = timestamp_iso[:-7]
13     return timestamp_iso_seconds
14
15
16  def open_filter_report(filter_hashed):
17 """
18     Open a text file with the following naming format:
19     md5hash_request_filter.txt
20     Example: 2020-03-31T22:00:00_request_summary.txt
21 """
22     folder = "data/images/image_request/request_reports"
23     if not os.path.exists(folder):
24         os.makedirs(folder)
25     text_file_name = f'{folder}/{filter_hashed}_request_filter.txt'
26     print(f'{text_file_name} file created.')
27     return open(text_file_name, "w")
28
29
30  def image_result_to_csv(df, filter_hash):
31 """

```

```

32     Create a CSV file with the results
33     of the data request to GBIF
34     and name it with a hash of the filter applied.
35     Results from different filters
36     are stored in a new file, while results
37     from the same filter are overwritten.
38     """
39     folder = "data/images/request/outputs"
40     if not os.path.exists(folder):
41         os.makedirs(folder)
42     csv_file_name = f"{folder}/{filter_hash}_request_results.csv"
43     df.to_csv(csv_file_name, sep = ",", header = True, index = None, encoding="utf-8")
44     print(f"+{csv_file_name} file created.")
45
46
47 def read_csv_name():
48     """
49     Prompt user for file path and name.
50     """
51     return input("Enter file name (full path): ")
52
53
54 def create_dataframe_from_csv(path_to_csv):
55     """
56     Given a full path, open a CSV file
57     and store it into a data frame object.
58     """
59     try:
60         df = pd.read_csv(""+path_to_csv+"", sep=",", encoding="utf-8") # Change to utf-8 if CSV has this
61         encoding
62     except:
63         raise
64     return df
65
66 def column_to_list(df, column):
67     try:
68         return df[column].tolist()
69     except:
70         print(f"Unable to extract {column} from {df}.")
71
72
73 def copy_dir(src_path, dest_path):
74     """
75     Copy all the files in the source directory
76     to a newly created destination directory.
77     """
78     try:
79         shutil.rmtree(dest_path)
80     except:
81         pass
82     finally:
83         shutil.copytree(src_path, dest_path)
84         print(f"{src_path} copied to {dest_path}")

```

data/storage_handler.py

```

1 import os
2 import subprocess
3
4 def get_images_size():
5     """
6     Return size of images folder in GB.
7     """
8     path = "images/"
9     total_size = 0
10    for root, directory, files in os.walk(path):
11        for file in files:
12            file_size = subprocess.check_output(f"wc -c < {path}{file}", shell=True)
13            file_size = int(file_size)
14            total_size = total_size + file_size
15    total_size_gb = round(((total_size/1024)/1024)/1024),6)
16    return total_size_gb
17
18
19 def stop_if_size(max_size):
20     """
21     Raise AssertionError if image folder reached size limit.
22     (I am using this as a security measure to not collapse
23     the VM where I am working due to lack of space
24     when downloading all the images).
25     """
26     assert get_images_size() < max_size, f"Image folder reached {max_size} GB."
27
28
29
30 if __name__ == "__main__":
31     stop_if_size(10)

```

D.2. Pre-procesado de datos

D.2.1. Imágenes

data/preprocessing/1_store_raw.py

```

1 import os
2 import shutil
3 import errno
4 from data.search import filter_hash
5 from data.config import images_filter, species_list
6
7 def create_raw_images_dir(base_path):
8     """
9         Create the folder to store the raw images
10        so that the image preprocessing can be decoupled
11        from the image request.
12    """
13    raw_images_dir = "raw_images"
14    path_raw = os.path.join(base_path, raw_images_dir)
15    print(f"Creating directory: {path_raw}")
16    try:
17        shutil.rmtree(path_raw)
18    except OSError:
19        raise
20    else:
21        os.mkdir(path_raw)
22        print(f"Created directory {path_raw}.")
23    return path_raw
24
25
26 def copy_directories(source_path, destination_path, symlinks=False, ignore=None):
27     """
28         Copy all directories and files from source to origin directory.
29     """
30     for item in os.listdir(source_path):
31         source = os.path.join(source_path, item)
32         destination = os.path.join(destination_path, item)
33         if os.path.isdir(source):
34             shutil.copytree(source, destination, symlinks, ignore)
35             print(f"Copied {item} directory into {destination_path}")
36         else:
37             shutil.copy2(source, destination)
38     print("End.")
39
40 if __name__ == "__main__":
41     # Base path
42     base_path = "data/images/image_preprocessing/"
43     # Source directory
44     filter_hash = filter_hash(images_filter, species_list.species_list)
45     source_path = f"data/images/image_request/{filter_hash}_images"
46     # Create destination directory and copy contents from source
47     destination_path = create_raw_images_dir(base_path)
48     copy_directories(source_path, destination_path)

```

data/preprocessing/2_image_sizes.py

```

1 from PIL import Image
2 import os
3 import statistics
4 import matplotlib.pyplot as plt
5
6
7 def get_image_sizes(source_dir):
8     """
9         Return a list of lists with
10        (image_width, image_height)
11        for every image in the directory
12        and classes subdirectories.
13    """
14     img_sizes = []
15     if os.path.exists(source_dir):
16         for species_folder in os.listdir(source_dir):
17             for image_file in os.listdir(os.path.join(source_dir, species_folder)):
18                 try:
19                     image_obj = Image.open(os.path.join(source_dir, species_folder, image_file))
20                     img_width, img_height = image_obj.size
21                     img_sizes.append([img_width, img_height])
22                 except:
23                     print(f"Unable to open {image_file}.")
24                     pass
25     else:
26         print(f"Folder {source_dir} not found.")
27     print("End.")
28     return img_sizes
29
30
31 def get_widths(img_sizes):
32     widths = []
33     for size in img_sizes:
34         widths.append(size[0])
35     return widths

```

```

37
38 def get_heights(img_sizes):
39     heights = []
40     for size in img_sizes:
41         heights.append(size[1])
42     return heights
43
44
45 def show_image_statistics(img_sizes):
46     print(f"Bigger: {max(img_sizes)}")
47     print(f"Smaller: {min(img_sizes)}")
48     widths = get_widths(img_sizes)
49     heights = get_heights(img_sizes)
50     print(f"Average width: {round(sum(widths)/len(widths))}, stdev: {round(statistics.stdev(widths))}")
51     print(f"Average height, stdev: {round(sum(heights)/len(heights))}, stdev: {round(statistics.stdev(heights))}")
52     print(f"Median width: {statistics.median(widths)}")
53     print(f"Median height: {statistics.median(heights)}")
54     print(f"Count width below 1000 px: {sum(i < 1000 for i in widths)} of {len(widths)}")
55     print(f"Count height below 2000 px: {sum(i < 2000 for i in heights)} of {len(heights)}")
56
57
58 def plot_image_dimensions(img_sizes):
59     widths = get_widths(img_sizes)
60     heights = get_heights(img_sizes)
61     plt.scatter(widths, heights, alpha=0.8, color="grey", marker="o", s=2)
62     plt.xlabel("Image width (px)", fontsize=8)
63     plt.ylabel("Image height (px)", fontsize=8)
64     plt.xticks(fontsize=8)
65     plt.yticks(fontsize=8)
66
67
68 def plot_dimension_distribution(img_sizes):
69     widths = get_widths(img_sizes)
70     heights = get_heights(img_sizes)
71     #fig, ax = plt.subplots()
72     plt.hist([widths, heights], bins = 40, alpha = 0.5,
73             color=[["lightgrey", "lightblue"]],
74             edgecolor="grey", label=["Width", "Height"])
75     plt.ylabel("Frequency", fontsize=8)
76     plt.xlabel("Image width / height (px)", fontsize=8)
77     plt.xticks(fontsize=8)
78     plt.yticks(fontsize=8)
79     plt.legend()
80
81
82 def plot_image_sizes(img_sizes, destination_path):
83     fig, axs = plt.subplots(2)
84     fig.suptitle('Image sizes')
85     plt.subplot(2,1,1)
86     plot_image_dimensions(img_sizes)
87     plt.subplot(2,1,2)
88     plot_dimension_distribution(img_sizes)
89     plt.savefig(destination_path)
90
91
92 if __name__ == "__main__":
93
94     # SOURCE images
95     source_dir = "data/images/raw_images"
96
97     img_sizes = get_image_sizes(source_dir)
98     show_image_statistics(img_sizes)
99
100    # DESTINATION plot
101    destination_path = "data/images/image_preprocessing/image_sizes.png"
102    plot_image_sizes(img_sizes, destination_path)

```

data/preprocessing/3_resize.py

```

1 import os
2 import errno
3 import shutil
4 from PIL import Image
5 from data.config import resizing
6 from data.image import ImageDimension
7
8
9 def create_empty_dir(base_path, directory):
10     """
11     Create a new directory.
12     If it already exists, it deletes it and all its contents.
13     It returns the new directory path.
14     """
15     full_path = base_path + directory
16     print(f"Creating directory: {full_path}")
17     try:
18         shutil.rmtree(full_path)
19     except:
20         pass
21     finally:
22         os.mkdir(full_path)
23         print(f"Created directory {full_path}.")
24         return full_path
25
26
27 def resize_image(source_dir, destination_dir, image_file, new_size):

```

```

28 """
29 """
30 try:
31     image = Image.open(os.path.join(source_dir, image_file))
32     try:
33         image = image.resize(new_size)
34         try:
35             if not os.path.exists(destination_dir):
36                 os.mkdir(destination_dir)
37             image.save(os.path.join(destination_dir, image_file))
38         except:
39             return print(f"Unable to save image {image_file}.")
40     except:
41         return print(f"Unable to resize image {image_file}.")
42 except:
43     print(f"Unable to open image {image_file}.")
44
45
46
47 def resize_all_images(source_dir, destination_dir, new_size):
48     """
49     Apply crop_image() to every image in the source directory.
50     The source directory must have subfolders for each of the classes.
51     """
52     if os.path.exists(source_dir):
53         for species_folder in os.listdir(source_dir):
54             for image_file in os.listdir(os.path.join(source_dir, species_folder)):
55                 resize_image(os.path.join(source_dir, species_folder),
56                             os.path.join(destination_dir, species_folder),
57                             image_file,
58                             new_size)
59     else:
60         print(f"Folder {source_dir} not found.")
61     print("End.")
62
63
64
65
66
67 if __name__ == "__main__":
68
69     # SOURCE DIR
70     source_dir = "data/images/raw_images/"
71
72     # DESTINATION DIR
73     base_path = "data/images/image_preprocessing/"
74     directory = "resized_images"
75     destination_dir = create_empty_dir(base_path, directory)
76
77     # Resize
78     new_size = resizing.image_size()
79     resize_all_images(source_dir, destination_dir, new_size)

```

data/preprocessing/4_crop.py

```

1 #import tensorflow as tf
2 import os
3 import errno
4 import shutil
5 from PIL import Image
6 from data.config import cropping
7 from data.image import ImageDimension
8
9
10 def create_empty_dir(base_path, directory):
11     """
12     Create a new directory.
13     If it already exists, it deletes it and all its contents.
14     It returns the new directory path.
15     """
16     full_path = base_path + directory
17     print(f"Creating directory: {full_path}")
18     try:
19         shutil.rmtree(full_path)
20     except:
21         pass
22     finally:
23         os.mkdir(full_path)
24         print(f"Created directory {full_path}.")
25     return full_path
26
27
28 def jpg_to_tensor(source_dir, image_file):
29     """
30     Given a path to an image,
31     return a 3-D tensor.
32     """
33     try:
34         return tf.image.decode_jpeg(tf.io.read_file(f"{source_dir}/{image_file}"), channels=3)
35     except:
36         return print(f"Unable to open image {image_file}.")
37
38
39 def crop_image(tensor, image_file):
40     """
41     Crop image at a fixed position.
42     Documentation: https://www.tensorflow.org/api_docs/python/tf/image/crop_to_bounding_box

```

```

43 """
44     ## Set the variable values here
45     # Offset variables values (top-left corner)
46     offset_height= 400 # Vertical coordinate
47     offset_width = 200 # Horizontal coordinate
48     # Target variables values
49     target_height = 200 # Height of the result
50     target_width = 200 # Width of the result
51
52     try:
53         return tf.image.crop_to_bounding_box(tensor, offset_height, offset_width, target_height, target_width)
54     except:
55         return print(f"Unable to crop image {image_file}.")

56
57 def crop_image_pil(source_dir, destination_dir, image_file, crop_coords):
58 """
59     Crop an image by reference points in x,y axis:
60     - left (x), top (y)
61     - right (x), bottom (y)
62     Opens image file from source directory
63     and saves it in the destination directory.
64 """
65
66     try:
67         source_image = Image.open(os.path.join(source_dir, image_file))
68     try:
69         cropped = source_image.crop(crop_coords)
70     try:
71         if not os.path.exists(destination_dir):
72             os.mkdir(destination_dir)
73         cropped.save(os.path.join(destination_dir,image_file))
74     except:
75         return print(f"Unable to save image {image_file}.")
76     except:
77         return print(f"Unable to crop image {image_file}.")
78     except:
79         return print(f"Unable to open image {image_file}.")

80
81 def resize_image(tensor, image_file):
82 """
83     Given an image stored as tensor,
84     scale to a fixed pixel number.
85     (From documentation: "Resizes an image to a target width and height
86     by keeping the aspect ratio the same without distortion.
87     If the target dimensions don't match the image dimensions,
88     the image is resized and then padded with zeroes
89     to match requested dimensions.")
90 """
91     size = [100,100]
92
93     try:
94         return tf.image.resize_images(tensor, size, method=tf.image.ResizeMethod.BILINEAR)
95     except:
96         return print(f"Unable to resize image {image_file}.")

97
98 def tensor_to_jpg(tensor, destination_dir, image_file):
99 """
100     Given a 3-D tensor
101     return a .jpg image and save it to destination.
102 """
103
104     try:
105         output_image = tf.image.encode_jpeg(tensor)
106         if not os.path.exists(destination_dir):
107             os.mkdir(destination_dir)
108         # Create a constant as filename
109         image_path = os.path.join(destination_dir, image_file)
110         file_name = tf.constant(image_path)
111         tf.io.write_file(file_name, output_image)
112     except:
113         print(f"Unable to save image {image_file}.")
114     else:
115         print(f"{image_file} saved to {image_path}.")

116
117 def crop_all(source_dir, destination_dir, cropping_coordinates):
118 """
119     Apply crop_image() to every image in the source directory.
120     The source directory must have subfolders for each of the classes.
121 """
122
123     if os.path.exists(source_dir):
124         for species_folder in os.listdir(source_dir):
125             for image_file in os.listdir(os.path.join(source_dir, species_folder)):
126                 #image = jpg_to_tensor(os.path.join(source_dir, species_folder), image_file)
127                 #image = resize_image(image, image_file)
128                 #image = crop_image(image, image_file)
129                 #tensor_to_jpg(image, os.path.join(destination_dir, species_folder), image_file)
130                 crop_image_pil(os.path.join(source_dir, species_folder),
131                               os.path.join(destination_dir, species_folder),
132                               image_file,
133                               cropping_coordinates
134 )
135     else:
136         print(f"Folder {source_dir} not found.")
137
138
139 if __name__ == "__main__":

```

```

140     # Create destination directory for the cropped images
141     base_path = "data/images/image_preprocessing/"
142     directory = "cropped_images"
143     destination_dir = create_empty_dir(base_path, directory)
144
145     # Crop all images in the directory
146     source_dir = "data/images/image_preprocessing/resized_images/"
147     crop_coords = cropping.coordinates()
148     crop_all(source_dir, destination_dir, crop_coords)
149

```

data/preprocessing/5_discard_images.py

```

1 import os
2
3 def remove_files(directory, file_list):
4     """
5         Given a path to a directory and a list of files,
6         remove them from directory.
7     """
8     count = 0
9     not_exist_count = 0
10    for file in file_list:
11        if os.path.exists(os.path.join(directory, get_species_folder(file), file)):
12            try:
13                os.remove(os.path.join(directory, get_species_folder(file), file))
14                count += 1
15            except:
16                print(f"Unable to remove image {file}")
17            else:
18                not_exist_count += 1
19
20    print(f"{not_exist_count} files not found. {count} files removed.")
21
22 def get_species_folder(file_name):
23     """
24         Return a string with the name of the species folder.
25         Arg.: a file name with the format:
26             Genus_species_occurrencenumber.jpg
27     """
28     file_split = file_name.split("_")
29     # Remove occurrence number and file extension
30     folder_name splitted = file_split[:-1]
31     return "_".join(folder_name splitted)
32
33
34 def faulty_images_list():
35     """
36         Return a list of file names of images considered incorrect
37         for the following reasons:
38             - Not a plant (an envelope, a label, a microscopic slide...)
39             - Too blurry
40             - Branch without leaves or fruits
41             - A living specimen
42             - Clearly incorrect species
43     (Manual curation)
44     """
45     return [
46         "Acer_campstre_1098946284.jpg",
47         "Acer_campstre_1098946313.jpg",
48         "Acer_campstre_1098946336.jpg",
49         "Acer_campstre_1098946338.jpg",
50         "Acer_campstre_1099839092.jpg",
51         "Acer_campstre_1122717794.jpg",
52         "Acer_campstre_1148621251.jpg",
53         "Acer_campstre_1697737720.jpg",
54         "Acer_campstre_1837899639.jpg",
55         "Acer_campstre_1935644575.jpg",
56         "Acer_campstre_1936420529.jpg",
57         "Acer_campstre_1936435993.jpg",
58         "Acer_campstre_1936563431.jpg",
59         "Acer_campstre_1936564441.jpg",
60         "Acer_campstre_1936564551.jpg",
61         "Acer_campstre_1936566860.jpg",
62         "Acer_campstre_1936659957.jpg",
63         "Acer_campstre_1949280499.jpg",
64         "Acer_campstre_1949280598.jpg",
65         "Acer_campstre_1949280602.jpg",
66         "Acer_campstre_1990252836.jpg",
67         "Acer_campstre_731858389.jpg",
68         "Alnus_glutinosa_1099839541.jpg",
69         "Alnus_glutinosa_1806488221.jpg",
70         "Alnus_glutinosa_1839391857.jpg",
71         "Alnus_glutinosa_1839953093.jpg",
72         "Alnus_glutinosa_1839953122.jpg",
73         "Alnus_glutinosa_1927931754.jpg",
74         "Alnus_glutinosa_1927937908.jpg",
75         "Alnus_glutinosa_1949280757.jpg",
76         "Alnus_glutinosa_2407331157.jpg",
77         "Betula_pendula_1099842753.jpg",
78         "Betula_pendula_1099842755.jpg",
79         "Betula_pendula_2405220187.jpg",
80         "Betula_pendula_2597535043.jpg",
81         "Betula_pendula_731859233.jpg",
82         "Betula_pendula_731859234.jpg",
83         "Betula_pubescens_1949917744.jpg",
84         "Betula_pubescens_237689001.jpg",
85         "Betula_pubescens_574752261.jpg",

```

```
85 "Buxus_sempervirens_1148620951.jpg",
86 "Buxus_sempervirens_1148621157.jpg",
87 "Buxus_sempervirens_1849127119.jpg",
88 "Buxus_sempervirens_1897272763.jpg",
89 "Buxus_sempervirens_2514282713.jpg",
90 "Buxus_sempervirens_2515325170.jpg",
91 "Buxus_sempervirens_2517320990.jpg",
92 "Buxus_sempervirens_731859238.jpg",
93 "Buxus_sempervirens_731859239.jpg",
94 "Carpinus_betulus_1099844167.jpg",
95 "Carpinus_betulus_1099844173.jpg",
96 "Carpinus_betulus_1099844178.jpg",
97 "Carpinus_betulus_1099844181.jpg",
98 "Carpinus_betulus_1122717966.jpg",
99 "Carpinus_betulus_1099845536.jpg",
100 "Carpinus_betulus_1099845331.jpg",
101 "Carpinus_betulus_1122717970.jpg",
102 "Carpinus_betulus_1122717972.jpg",
103 "Carpinus_betulus_1122717973.jpg",
104 "Carpinus_betulus_1122717980.jpg",
105 "Carpinus_betulus_1122717987.jpg",
106 "Carpinus_betulus_1122717989.jpg",
107 "Carpinus_betulus_1122717993.jpg",
108 "Carpinus_betulus_1122717995.jpg",
109 "Carpinus_betulus_1122718000.jpg",
110 "Carpinus_betulus_1122718001.jpg",
111 "Carpinus_betulus_1122718004.jpg",
112 "Carpinus_betulus_1122718009.jpg",
113 "Carpinus_betulus_1122718017.jpg",
114 "Carpinus_betulus_1148621501.jpg",
115 "Carpinus_betulus_1148635521.jpg",
116 "Carpinus_betulus_1148621501.jpg",
117 "Carpinus_betulus_1697777246.jpg",
118 "Carpinus_betulus_1701609281.jpg",
119 "Carpinus_betulus_1701817240.jpg",
120 "Carpinus_betulus_1949281049.jpg",
121 "Carpinus_betulus_1949281051.jpg",
122 "Carpinus_betulus_1949281054.jpg",
123 "Carpinus_betulus_1949281072.jpg",
124 "Carpinus_betulus_1949281144.jpg",
125 "Carpinus_betulus_1949281163.jpg",
126 "Carpinus_betulus_1990212683.jpg",
127 "Carpinus_betulus_2513853751.jpg",
128 "Carpinus_betulus_2516224887.jpg",
129 "Cornus_sanguinea_1099846507.jpg",
130 "Cornus_sanguinea_2515144511.jpg",
131 "Cornus_sanguinea_2515144512.jpg",
132 "Cornus_sanguinea_2515144513.jpg",
133 "Cornus_sanguinea_2515144514.jpg",
134 "Cornus_sanguinea_2515146850.jpg",
135 "Cornus_sanguinea_2515146851.jpg",
136 "Cornus_sanguinea_2515146852.jpg",
137 "Cornus_sanguinea_2515146853.jpg",
138 "Cornus_sanguinea_2515146854.jpg",
139 "Cornus_sanguinea_2515149452.jpg",
140 "Cornus_sanguinea_2515149453.jpg",
141 "Cornus_sanguinea_2515149454.jpg",
142 "Cornus_sanguinea_2515149455.jpg",
143 "Cornus_sanguinea_2515150495.jpg",
144 "Cornus_sanguinea_2515150496.jpg",
145 "Cornus_sanguinea_2515152448.jpg",
146 "Cornus_sanguinea_2515152449.jpg",
147 "Cornus_sanguinea_2515152450.jpg",
148 "Cornus_sanguinea_2515152451.jpg",
149 "Cornus_sanguinea_2515221837.jpg",
150 "Cornus_sanguinea_2515221838.jpg",
151 "Cornus_sanguinea_2515223131.jpg",
152 "Cornus_sanguinea_2515223132.jpg",
153 "Cornus_sanguinea_2515223133.jpg",
154 "Cornus_sanguinea_2515223134.jpg",
155 "Cornus_sanguinea_2515228003.jpg",
156 "Cornus_sanguinea_2515228004.jpg",
157 "Cornus_sanguinea_2515228005.jpg",
158 "Cornus_sanguinea_2515228006.jpg",
159 "Cornus_sanguinea_2515231484.jpg",
160 "Cornus_sanguinea_2515231485.jpg",
161 "Cornus_sanguinea_2515231486.jpg",
162 "Cornus_sanguinea_2515231487.jpg",
163 "Cornus_sanguinea_2515233522.jpg",
164 "Cornus_sanguinea_2515233523.jpg",
165 "Cornus_sanguinea_2515233524.jpg",
166 "Cornus_sanguinea_2515246661.jpg",
167 "Cornus_sanguinea_2515246662.jpg",
168 "Cornus_sanguinea_2515246663.jpg",
169 "Cornus_sanguinea_2515246664.jpg",
170 "Cornus_sanguinea_2515246665.jpg",
171 "Cornus_sanguinea_2515254315.jpg",
172 "Cornus_sanguinea_2515254316.jpg",
173 "Cornus_sanguinea_2515258496.jpg",
174 "Cornus_sanguinea_2515258497.jpg",
175 "Cornus_sanguinea_2515258498.jpg",
176 "Cornus_sanguinea_2515258499.jpg",
177 "Cornus_sanguinea_2515262137.jpg",
178 "Cornus_sanguinea_2515262138.jpg",
179 "Cornus_sanguinea_2515262139.jpg",
180 "Cornus_sanguinea_2515262140.jpg",
181 "Cornus_sanguinea_2515266729.jpg",
```

```
182 "Cornus_sanguinea_2515266730.jpg",
183 "Cornus_sanguinea_2515266731.jpg",
184 "Cornus_sanguinea_2515309091.jpg",
185 "Cornus_sanguinea_2515309092.jpg",
186 "Cornus_sanguinea_2515309093.jpg",
187 "Cornus_sanguinea_2515309094.jpg",
188 "Cornus_sanguinea_2515325378.jpg",
189 "Cornus_sanguinea_2515325379.jpg",
190 "Cornus_sanguinea_2515325380.jpg",
191 "Cornus_sanguinea_2515326493.jpg",
192 "Cornus_sanguinea_2515326494.jpg",
193 "Cornus_sanguinea_2515342311.jpg",
194 "Cornus_sanguinea_2515342312.jpg",
195 "Cornus_sanguinea_2515342313.jpg",
196 "Cornus_sanguinea_2515342314.jpg",
197 "Cornus_sanguinea_2515347757.jpg",
198 "Cornus_sanguinea_2515347758.jpg",
199 "Cornus_sanguinea_2515347759.jpg",
200 "Cornus_sanguinea_2515347760.jpg",
201 "Cornus_sanguinea_2515361541.jpg",
202 "Cornus_sanguinea_2515361542.jpg",
203 "Cornus_sanguinea_2515361543.jpg",
204 "Cornus_sanguinea_2515361544.jpg",
205 "Cornus_sanguinea_2515372528.jpg",
206 "Cornus_sanguinea_2515372529.jpg",
207 "Cornus_sanguinea_2515372530.jpg",
208 "Cornus_sanguinea_2515373770.jpg",
209 "Cornus_sanguinea_2515373771.jpg",
210 "Cornus_sanguinea_2515373772.jpg",
211 "Cornus_sanguinea_2515380888.jpg",
212 "Cornus_sanguinea_2515380889.jpg",
213 "Cornus_sanguinea_2515380890.jpg",
214 "Cornus_sanguinea_2515380891.jpg",
215 "Cornus_sanguinea_2515383127.jpg",
216 "Cornus_sanguinea_2515383128.jpg",
217 "Cornus_sanguinea_2515383129.jpg",
218 "Cornus_sanguinea_2515383130.jpg",
219 "Cornus_sanguinea_2515390692.jpg",
220 "Cornus_sanguinea_2515390693.jpg",
221 "Cornus_sanguinea_2515390694.jpg",
222 "Cornus_sanguinea_2515390695.jpg",
223 "Cornus_sanguinea_2515391866.jpg",
224 "Cornus_sanguinea_2515391867.jpg",
225 "Cornus_sanguinea_2515391868.jpg",
226 "Cornus_sanguinea_2515393127.jpg",
227 "Cornus_sanguinea_2515393128.jpg",
228 "Cornus_sanguinea_2516841397.jpg",
229 "Cornus_sanguinea_2516841398.jpg",
230 "Cornus_sanguinea_2516841399.jpg",
231 "Cornus_sanguinea_2516848384.jpg",
232 "Cornus_sanguinea_2516848385.jpg",
233 "Cornus_sanguinea_2516848386.jpg",
234 "Cornus_sanguinea_2516848387.jpg",
235 "Cornus_sanguinea_2516848388.jpg",
236 "Cornus_sanguinea_2516848389.jpg",
237 "Cornus_sanguinea_2516858442.jpg",
238 "Cornus_sanguinea_2516858443.jpg",
239 "Cornus_sanguinea_2516858444.jpg",
240 "Cornus_sanguinea_2516872404.jpg",
241 "Cornus_sanguinea_2516872405.jpg",
242 "Cornus_sanguinea_2516872406.jpg",
243 "Cornus_sanguinea_2516872407.jpg",
244 "Cornus_sanguinea_2516874353.jpg",
245 "Cornus_sanguinea_2516874354.jpg",
246 "Cornus_sanguinea_2516874355.jpg",
247 "Cornus_sanguinea_2516874356.jpg",
248 "Cornus_sanguinea_2516999041.jpg",
249 "Cornus_sanguinea_2516999042.jpg",
250 "Cornus_sanguinea_2516999043.jpg",
251 "Cornus_sanguinea_2516999044.jpg",
252 "Cornus_sanguinea_2517003103.jpg",
253 "Cornus_sanguinea_2517003104.jpg",
254 "Cornus_sanguinea_2517003105.jpg",
255 "Cornus_sanguinea_2517004097.jpg",
256 "Cornus_sanguinea_2517004098.jpg",
257 "Cornus_sanguinea_2517010090.jpg",
258 "Cornus_sanguinea_2517010091.jpg",
259 "Cornus_sanguinea_2517010092.jpg",
260 "Cornus_sanguinea_2517010093.jpg",
261 "Cornus_sanguinea_2517010094.jpg",
262 "Cornus_sanguinea_2517013065.jpg",
263 "Cornus_sanguinea_2517013066.jpg",
264 "Cornus_sanguinea_2517013067.jpg",
265 "Cornus_sanguinea_2517013068.jpg",
266 "Cornus_sanguinea_2517027107.jpg",
267 "Cornus_sanguinea_2517027108.jpg",
268 "Cornus_sanguinea_2517027109.jpg",
269 "Cornus_sanguinea_2517027110.jpg",
270 "Cornus_sanguinea_2517036153.jpg",
271 "Cornus_sanguinea_2517036154.jpg",
272 "Cornus_sanguinea_2517036155.jpg",
273 "Cornus_sanguinea_2517037080.jpg",
274 "Cornus_sanguinea_2517037081.jpg",
275 "Cornus_sanguinea_2517037082.jpg",
276 "Cornus_sanguinea_2517037083.jpg",
277 "Cornus_sanguinea_2517037084.jpg",
278 "Cornus_sanguinea_2517048084.jpg",
```

```
279 "Cornus_sanguinea_2517048085.jpg",
280 "Cornus_sanguinea_2517050028.jpg",
281 "Cornus_sanguinea_2517050029.jpg",
282 "Cornus_sanguinea_2517050030.jpg",
283 "Cornus_sanguinea_2517224082.jpg",
284 "Cornus_sanguinea_2517224083.jpg",
285 "Cornus_sanguinea_2517240035.jpg",
286 "Cornus_sanguinea_2517240036.jpg",
287 "Cornus_sanguinea_2517240037.jpg",
288 "Cornus_sanguinea_2517245055.jpg",
289 "Cornus_sanguinea_2517245056.jpg",
290 "Cornus_avellana_1099846490.jpg",
291 "Corylus_avellana_1099846500.jpg",
292 "Corylus_avellana_1099846508.jpg",
293 "Corylus_avellana_1122718086.jpg",
294 "Corylus_avellana_1122718088.jpg",
295 "Corylus_avellana_1322451155.jpg",
296 "Corylus_avellana_1697726069.jpg",
297 "Corylus_avellana_1697777547.jpg",
298 "Corylus_avellana_1701597164.jpg",
299 "Corylus_avellana_1701861121.jpg",
300 "Corylus_avellana_2462778608.jpg",
301 "Corylus_avellana_1990252990.jpg",
302 "Corylus_avellana_2462877875.jpg",
303 "Corylus_avellana_2462878498.jpg",
304 "Crataegus_laevigata_1936130792.jpg",
305 "Crataegus_monogyna_2514172238.jpg",
306 "Euonymus_europaea_1099000665.jpg",
307 "Euonymus_europaea_1099000689.jpg",
308 "Euonymus_europaea_1099000717.jpg",
309 "Euonymus_europaea_1099001058.jpg",
310 "Euonymus_europaea_1099001059.jpg",
311 "Euonymus_europaea_1099001063.jpg",
312 "Euonymus_europaea_1099001064.jpg",
313 "Euonymus_europaea_1122718544.jpg",
314 "Euonymus_europaea_1148621490.jpg",
315 "Euonymus_europaea_1148520675.jpg",
316 "Euonymus_europaea_2516607526.jpg",
317 "Fagus_sylvatica_1099929461.jpg",
318 "Fagus_sylvatica_1099929463.jpg",
319 "Fagus_sylvatica_1099929464.jpg",
320 "Fagus_sylvatica_1099929466.jpg",
321 "Fagus_sylvatica_1099934143.jpg",
322 "Fagus_sylvatica_1122718572.jpg",
323 "Fagus_sylvatica_1122718574.jpg",
324 "Fagus_sylvatica_1122718578.jpg",
325 "Fagus_sylvatica_1122718579.jpg",
326 "Fagus_sylvatica_1122718580.jpg",
327 "Fagus_sylvatica_1122718581.jpg",
328 "Fagus_sylvatica_1122718585.jpg",
329 "Fagus_sylvatica_1122718586.jpg",
330 "Fagus_sylvatica_1148621507.jpg",
331 "Fagus_sylvatica_1840493524.jpg",
332 "Fagus_sylvatica_1949281397.jpg",
333 "Fagus_sylvatica_1949281398.jpg",
334 "Fagus_sylvatica_1949281399.jpg",
335 "Fagus_sylvatica_1949281400.jpg",
336 "Fagus_sylvatica_1949281408.jpg",
337 "Fagus_sylvatica_1949281409.jpg",
338 "Fagus_sylvatica_1949281411.jpg",
339 "Fagus_sylvatica_1949281412.jpg",
340 "Fagus_sylvatica_1949281415.jpg",
341 "Fagus_sylvatica_1949281416.jpg",
342 "Fagus_sylvatica_1949281417.jpg",
343 "Fagus_sylvatica_1949281418.jpg",
344 "Fagus_sylvatica_1949281419.jpg",
345 "Fagus_sylvatica_1949281422.jpg",
346 "Fagus_sylvatica_1949281424.jpg",
347 "Fagus_sylvatica_1949281425.jpg",
348 "Fagus_sylvatica_1949281429.jpg",
349 "Fagus_sylvatica_1949281437.jpg",
350 "Fagus_sylvatica_1949281441.jpg",
351 "Fagus_sylvatica_1949281444.jpg",
352 "Fagus_sylvatica_1949281454.jpg",
353 "Fagus_sylvatica_1949281457.jpg",
354 "Fagus_sylvatica_1949281480.jpg",
355 "Fagus_sylvatica_1949281497.jpg",
356 "Fagus_sylvatica_1949281502.jpg",
357 "Fagus_sylvatica_1949281504.jpg",
358 "Fagus_sylvatica_1949281505.jpg",
359 "Fagus_sylvatica_1949281508.jpg",
360 "Fagus_sylvatica_1949281509.jpg",
361 "Fagus_sylvatica_1949281510.jpg",
362 "Fagus_sylvatica_1949281514.jpg",
363 "Fagus_sylvatica_1949281517.jpg",
364 "Fagus_sylvatica_1949281523.jpg",
365 "Fagus_sylvatica_1949281527.jpg",
366 "Fagus_sylvatica_1949281533.jpg",
367 "Fagus_sylvatica_1949281534.jpg",
368 "Fagus_sylvatica_1949281535.jpg",
369 "Fagus_sylvatica_1949281536.jpg",
370 "Fagus_sylvatica_1949281537.jpg",
371 "Fagus_sylvatica_1949281539.jpg",
372 "Fagus_sylvatica_1949281540.jpg",
373 "Fagus_sylvatica_1949281541.jpg",
374 "Fagus_sylvatica_1949281551.jpg",
375 "Fagus_sylvatica_1949281552.jpg",
```

```
376 "Fagus_sylvatica_1949281558.jpg",
377 "Fagus_sylvatica_1949281559.jpg",
378 "Fagus_sylvatica_1949281567.jpg",
379 "Fagus_sylvatica_1949281575.jpg",
380 "Fagus_sylvatica_1949281595.jpg",
381 "Fagus_sylvatica_1949281597.jpg",
382 "Fagus_sylvatica_1949281611.jpg",
383 "Fagus_sylvatica_1949281613.jpg",
384 "Fagus_sylvatica_1949281614.jpg",
385 "Fagus_sylvatica_1949281620.jpg",
386 "Fagus_sylvatica_1949281623.jpg",
387 "Fagus_sylvatica_1949281649.jpg",
388 "Fagus_sylvatica_1949281651.jpg",
389 "Fagus_sylvatica_1949281652.jpg",
390 "Fagus_sylvatica_1949281678.jpg",
391 "Fagus_sylvatica_1949281686.jpg",
392 "Fagus_sylvatica_1949281691.jpg",
393 "Fagus_sylvatica_1949281693.jpg",
394 "Fagus_sylvatica_1949281701.jpg",
395 "Fagus_sylvatica_1949281702.jpg",
396 "Fagus_sylvatica_1949281712.jpg",
397 "Fagus_sylvatica_1949281720.jpg",
398 "Fagus_sylvatica_1949281723.jpg",
399 "Fagus_sylvatica_1949281738.jpg",
400 "Fagus_sylvatica_1949281751.jpg",
401 "Fagus_sylvatica_1949281768.jpg",
402 "Fagus_sylvatica_1949281773.jpg",
403 "Fagus_sylvatica_1949281775.jpg",
404 "Fagus_sylvatica_1949281780.jpg",
405 "Fagus_sylvatica_1949281781.jpg",
406 "Fagus_sylvatica_1949281789.jpg",
407 "Fagus_sylvatica_1949281792.jpg",
408 "Fagus_sylvatica_1949281793.jpg",
409 "Fagus_sylvatica_1949281794.jpg",
410 "Fagus_sylvatica_1949281805.jpg",
411 "Fagus_sylvatica_1949281808.jpg",
412 "Fagus_sylvatica_1949281812.jpg",
413 "Fagus_sylvatica_1949281814.jpg",
414 "Fagus_sylvatica_1949281817.jpg",
415 "Fagus_sylvatica_1949281821.jpg",
416 "Fagus_sylvatica_1949281824.jpg",
417 "Fagus_sylvatica_1949281835.jpg",
418 "Fagus_sylvatica_1949281839.jpg",
419 "Fagus_sylvatica_1949281842.jpg",
420 "Fagus_sylvatica_1949281843.jpg",
421 "Fagus_sylvatica_1949281861.jpg",
422 "Fagus_sylvatica_1949281865.jpg",
423 "Fagus_sylvatica_1949281873.jpg",
424 "Fagus_sylvatica_1949281882.jpg",
425 "Fagus_sylvatica_1949281895.jpg",
426 "Fagus_sylvatica_1949281901.jpg",
427 "Fagus_sylvatica_1949281903.jpg",
428 "Fagus_sylvatica_1949281906.jpg",
429 "Fagus_sylvatica_1949281907.jpg",
430 "Fagus_sylvatica_1949281908.jpg",
431 "Fagus_sylvatica_1949281912.jpg",
432 "Fagus_sylvatica_1949281917.jpg",
433 "Fagus_sylvatica_1949281918.jpg",
434 "Fagus_sylvatica_1949281919.jpg",
435 "Fagus_sylvatica_1949281937.jpg",
436 "Fagus_sylvatica_1949281941.jpg",
437 "Fagus_sylvatica_1949281958.jpg",
438 "Fagus_sylvatica_1949281960.jpg",
439 "Fagus_sylvatica_1949281965.jpg",
440 "Fagus_sylvatica_1949281968.jpg",
441 "Fagus_sylvatica_1949281993.jpg",
442 "Fagus_sylvatica_1949282011.jpg",
443 "Fagus_sylvatica_1949282028.jpg",
444 "Fagus_sylvatica_1949282029.jpg",
445 "Fagus_sylvatica_1949282034.jpg",
446 "Fagus_sylvatica_1949282035.jpg",
447 "Fagus_sylvatica_1949282041.jpg",
448 "Fagus_sylvatica_1949282042.jpg",
449 "Fagus_sylvatica_1949282045.jpg",
450 "Fagus_sylvatica_1949282047.jpg",
451 "Fagus_sylvatica_1949282048.jpg",
452 "Fagus_sylvatica_1949282049.jpg",
453 "Fagus_sylvatica_1949282050.jpg",
454 "Fagus_sylvatica_1949282051.jpg",
455 "Fagus_sylvatica_1949282055.jpg",
456 "Fagus_sylvatica_1949282057.jpg",
457 "Fagus_sylvatica_1949282058.jpg",
458 "Fagus_sylvatica_1949282102.jpg",
459 "Fagus_sylvatica_1949282104.jpg",
460 "Fagus_sylvatica_1949282109.jpg",
461 "Fagus_sylvatica_1949282117.jpg",
462 "Fagus_sylvatica_1949282119.jpg",
463 "Fagus_sylvatica_1949282121.jpg",
464 "Fagus_sylvatica_1949282125.jpg",
465 "Fagus_sylvatica_1949282127.jpg",
466 "Fagus_sylvatica_1949282131.jpg",
467 "Fagus_sylvatica_1949282133.jpg",
468 "Fagus_sylvatica_1949282136.jpg",
469 "Fagus_sylvatica_1949282137.jpg",
470 "Fagus_sylvatica_1949282143.jpg",
471 "Fagus_sylvatica_1949282149.jpg",
472 "Fagus_sylvatica_1949282150.jpg",
```

```
473 "Frangula_alnus_1321042704.jpg",
474 "Frangula_alnus_1701887055.jpg",
475 "Frangula_alnus_1949282262.jpg",
476 "Frangula_alnus_2517278080.jpg",
477 "Frangula_alnus_1929351981.jpg",
478 "Frangula_alnus_2517142215.jpg",
479 "Frangula_alnus_2404021110.jpg",
480 "Frangula_alnus_1978879142.jpg",
481 "Frangula_alnus_1978879287.jpg",
482 "Frangula_alnus_1978879643.jpg",
483 "Frangula_alnus_1978880263.jpg",
484 "Frangula_alnus_1978898537.jpg",
485 "Frangula_alnus_2573422889.jpg",
486 "Fraxinus_exelsior_731859512.jpg",
487 "Fraxinus_exelsior_731859513.jpg",
488 "Fraxinus_exelsior_731859514.jpg",
489 "Fraxinus_exelsior_1148621420.jpg",
490 "Fraxinus_exelsior_1148621533.jpg",
491 "Fraxinus_exelsior_1148625546.jpg",
492 "Fraxinus_exelsior_2402451889.jpg",
493 "Fraxinus_exelsior_2410575524.jpg",
494 "Fraxinus_exelsior_1091151091.jpg",
495 "Fraxinus_exelsior_1099001785.jpg",
496 "Fraxinus_exelsior_1099001871.jpg",
497 "Fraxinus_exelsior_1099004901.jpg",
498 "Fraxinus_exelsior_1099004948.jpg",
499 "Fraxinus_exelsior_1099004954.jpg",
500 "Fraxinus_exelsior_1099004955.jpg",
501 "Fraxinus_exelsior_1099004959.jpg",
502 "Fraxinus_exelsior_1099004969.jpg",
503 "Fraxinus_exelsior_1099934320.jpg",
504 "Fraxinus_exelsior_1099934322.jpg",
505 "Fraxinus_exelsior_1099934324.jpg",
506 "Fraxinus_exelsior_1099934325.jpg",
507 "Fraxinus_exelsior_1099934327.jpg",
508 "Fraxinus_exelsior_1099934328.jpg",
509 "Fraxinus_exelsior_1099934329.jpg",
510 "Fraxinus_exelsior_1099934330.jpg",
511 "Fraxinus_exelsior_1122718593.jpg",
512 "Fraxinus_exelsior_1122718595.jpg",
513 "Fraxinus_exelsior_1122718596.jpg",
514 "Fraxinus_exelsior_1122718598.jpg",
515 "Fraxinus_exelsior_1122718599.jpg",
516 "Fraxinus_exelsior_1122718601.jpg",
517 "Fraxinus_exelsior_1122718602.jpg",
518 "Fraxinus_exelsior_1122718603.jpg",
519 "Fraxinus_exelsior_1697821489.jpg",
520 "Fraxinus_exelsior_1701920465.jpg",
521 "Fraxinus_exelsior_1840486996.jpg",
522 "Fraxinus_exelsior_1840487028.jpg",
523 "Fraxinus_exelsior_1840487036.jpg",
524 "Fraxinus_exelsior_1840487049.jpg",
525 "Fraxinus_exelsior_1840487057.jpg",
526 "Fraxinus_exelsior_1840487065.jpg",
527 "Fraxinus_exelsior_1840487150.jpg",
528 "Fraxinus_exelsior_1840533514.jpg",
529 "Fraxinus_exelsior_1990213052.jpg",
530 "Fraxinus_exelsior_2514247268.jpg",
531 "Fraxinus_exelsior_2517413826.jpg",
532 "Populus_tremula_1099070035.jpg",
533 "Populus_tremula_1099992226.jpg",
534 "Populus_tremula_1122936623.jpg",
535 "Populus_tremula_1122936691.jpg",
536 "Populus_tremula_1122936690.jpg",
537 "Populus_tremula_1122937179.jpg",
538 "Populus_tremula_1122937275.jpg",
539 "Populus_tremula_1122937288.jpg",
540 "Populus_tremula_1122937301.jpg",
541 "Populus_tremula_1148473449.jpg",
542 "Populus_tremula_1949283898.jpg",
543 "Populus_tremula_1949284010.jpg",
544 "Populus_tremula_1949284034.jpg",
545 "Populus_tremula_1949284048.jpg",
546 "Populus_tremula_1949284052.jpg",
547 "Populus_tremula_1949284055.jpg",
548 "Populus_tremula_1949284055.jpg",
549 "Populus_tremula_1949284069.jpg",
550 "Populus_tremula_1949284070.jpg",
551 "Populus_tremula_1949284074.jpg",
552 "Populus_tremula_1949284076.jpg",
553 "Populus_tremula_1949284081.jpg",
554 "Populus_tremula_1949284085.jpg",
555 "Populus_tremula_1949284088.jpg",
556 "Populus_tremula_1949284089.jpg",
557 "Populus_tremula_1949284092.jpg",
558 "Populus_tremula_1949284096.jpg",
559 "Populus_tremula_1949284099.jpg",
560 "Populus_tremula_1949284100.jpg",
561 "Populus_tremula_1949284101.jpg",
562 "Populus_tremula_1949284108.jpg",
563 "Populus_tremula_1949284110.jpg",
564 "Populus_tremula_1949284116.jpg",
565 "Populus_tremula_1949284117.jpg",
566 "Populus_tremula_1949284136.jpg",
567 "Populus_tremula_1949284142.jpg",
568 "Populus_tremula_1949284146.jpg",
569 "Populus_tremula_1949284150.jpg",
```

```
570 "Populus_tremula_1949284162.jpg",
571 "Populus_tremula_1949284169.jpg",
572 "Populus_tremula_1949284174.jpg",
573 "Populus_tremula_1949284180.jpg",
574 "Populus_tremula_1949284182.jpg",
575 "Populus_tremula_1949284209.jpg",
576 "Populus_tremula_1949284210.jpg",
577 "Populus_tremula_1148473449.jpg",
578 "Populus_tremula_1095009438.jpg",
579 "Populus_tremula_1701804758.jpg",
580 "Prunus_avium_1042044715.jpg",
581 "Prunus_avium_1122937069.jpg",
582 "Prunus_avium_1122937083.jpg",
583 "Prunus_avium_1949284226.jpg",
584 "Prunus_avium_1949284303.jpg",
585 "Prunus_avium_1148621615.jpg",
586 "Prunus_avium_1697350916.jpg",
587 "Prunus_avium_1701291014.jpg",
588 "Prunus_avium_2462806962.jpg",
589 "Prunus_padus_1122936847.jpg",
590 "Prunus_padus_1122936967.jpg",
591 "Prunus_padus_1949282625.jpg",
592 "Prunus_padus_1949284225.jpg",
593 "Prunus_padus_1148621069.jpg",
594 "Prunus_padus_1148621552.jpg",
595 "Prunus_padus_2427605359.jpg",
596 "Prunus_spinosa_1099077881.jpg",
597 "Prunus_spinosa_1100001827.jpg",
598 "Prunus_spinosa_1100001830.jpg",
599 "Prunus_spinosa_1100001832.jpg",
600 "Prunus_spinosa_1100001836.jpg",
601 "Prunus_spinosa_1122936573.jpg",
602 "Prunus_spinosa_1122937099.jpg",
603 "Prunus_spinosa_1148620928.jpg",
604 "Prunus_spinosa_1949284199.jpg",
605 "Prunus_spinosa_1949284212.jpg",
606 "Prunus_spinosa_1949284221.jpg",
607 "Prunus_spinosa_1949284222.jpg",
608 "Prunus_spinosa_1949284223.jpg",
609 "Prunus_spinosa_1949284224.jpg",
610 "Prunus_spinosa_1949284312.jpg",
611 "Prunus_spinosa_2515347701.jpg",
612 "Prunus_spinosa_2516971651.jpg",
613 "Pyrus_cordata_437270420.jpg",
614 "Quercus_petraea_1099080097.jpg",
615 "Quercus_petraea_1099084228.jpg",
616 "Quercus_petraea_1099084600.jpg",
617 "Quercus_petraea_1122936973.jpg",
618 "Quercus_petraea_1702017565.jpg",
619 "Quercus_petraea_1949284412.jpg",
620 "Quercus_petraea_1949284473.jpg",
621 "Quercus_petraea_1949284514.jpg",
622 "Quercus_petraea_1949284526.jpg",
623 "Quercus_petraea_1949284560.jpg",
624 "Quercus_petraea_1949284722.jpg",
625 "Quercus_petraea_1805284827.jpg",
626 "Quercus_petraea_1935716425.jpg",
627 "Quercus_petraea_1837903479.jpg",
628 "Quercus_petraea_19365663368.jpg",
629 "Quercus_petraea_1936567313.jpg",
630 "Quercus_petraea_1936578338.jpg",
631 "Quercus_robur_1099084598.jpg",
632 "Quercus_robur_1099084601.jpg",
633 "Quercus_robur_1099084602.jpg",
634 "Quercus_robur_1099084608.jpg",
635 "Quercus_robur_1100005476.jpg",
636 "Quercus_robur_1100005478.jpg",
637 "Quercus_robur_1122936483.jpg",
638 "Quercus_robur_1122936586.jpg",
639 "Quercus_robur_1122936704.jpg",
640 "Quercus_robur_1122936710.jpg",
641 "Quercus_robur_1122936773.jpg",
642 "Quercus_robur_1122936837.jpg",
643 "Quercus_robur_1122936889.jpg",
644 "Quercus_robur_1148595171.jpg",
645 "Quercus_robur_1697464203.jpg",
646 "Quercus_robur_1990213719.jpg",
647 "Quercus_robur_1148627598.jpg",
648 "Quercus_robur_1423911332.jpg",
649 "Quercus_robur_2515703380.jpg",
650 "Rhamnus_cathartica_1062857643.jpg",
651 "Rhamnus_cathartica_1100007183.jpg",
652 "Rhamnus_cathartica_2597554360.jpg",
653 "Rhamnus_cathartica_2597554365.jpg",
654 "Rhamnus_cathartica_2597554369.jpg",
655 "Rhamnus_cathartica_1148621060.jpg",
656 "Rhamnus_cathartica_1701859492.jpg",
657 "Rhamnus_cathartica_1978879129.jpg",
658 "Rhamnus_cathartica_1978880056.jpg",
659 "Ilex_aquifolium_1122937168.jpg",
660 "Ilex_aquifolium_1424200857.jpg",
661 "Ilex_aquifolium_1424200994.jpg",
662 "Ilex_aquifolium_1424201217.jpg",
663 "Ilex_aquifolium_1424216798.jpg",
664 "Ilex_aquifolium_1424509879.jpg",
665 "Ilex_aquifolium_1424509912.jpg",
666 "Ilex_aquifolium_1837901922.jpg",
```

```
667 "Ilex_aquifolium_2515327511.jpg",
668 "Ilex_aquifolium_2515914065.jpg",
669 "Ilex_aquifolium_2516098696.jpg",
670 "Ilex_aquifolium_2516105293.jpg",
671 "Ilex_aquifolium_2517819570.jpg",
672 "Ilex_aquifolium_731858278.jpg",
673 "Ilex_aquifolium_731858279.jpg",
674 "Juniperus_communis_1148621258.jpg",
675 "Juniperus_communis_1701770661.jpg",
676 "Juniperus_communis_1806488293.jpg",
677 "Juniperus_communis_1806488286.jpg",
678 "Juniperus_communis_1936696114.jpg",
679 "Juniperus_communis_1949917310.jpg",
680 "Juniperus_communis_575057232.jpg",
681 "Malus_sylvestris_1212583786.jpg",
682 "Malus_sylvestris_1291928715.jpg",
683 "Malus_sylvestris_1424499806.jpg",
684 "Malus_sylvestris_2417006450.jpg",
685 "Malus_sylvestris_2417006596.jpg",
686 "Malus_sylvestris_2462764590.jpg",
687 "Populus_nigra_1099070037.jpg",
688 "Populus_nigra_1099076558.jpg",
689 "Populus_nigra_1099076560.jpg",
690 "Populus_nigra_1099077072.jpg",
691 "Populus_nigra_1122936854.jpg",
692 "Populus_nigra_1122937284.jpg",
693 "Populus_nigra_1949284019.jpg",
694 "Populus_nigra_1949284023.jpg",
695 "Populus_nigra_1949284029.jpg",
696 "Populus_nigra_1949284036.jpg",
697 "Populus_nigra_1949284041.jpg",
698 "Populus_nigra_1949284046.jpg",
699 "Populus_nigra_1949284047.jpg",
700 "Populus_nigra_1949284051.jpg",
701 "Populus_nigra_1949284053.jpg",
702 "Populus_nigra_1949284063.jpg",
703 "Populus_nigra_1949284068.jpg",
704 "Populus_nigra_1949284093.jpg",
705 "Populus_nigra_1949284128.jpg",
706 "Populus_nigra_1949284133.jpg",
707 "Populus_nigra_1949284143.jpg",
708 "Populus_nigra_1949284154.jpg",
709 "Populus_nigra_2515275541.jpg",
710 "Populus_nigra_1424107396.jpg",
711 "Populus_nigra_1978891606.jpg",
712 "Populus_nigra_1978891730.jpg",
713 "Populus_nigra_1978891779.jpg",
714 "Populus_nigra_1978891658.jpg",
715 "Populus_nigra_1978892255.jpg",
716 "Populus_nigra_1978892285.jpg",
717 "Populus_nigra_1990213532.jpg",
718 "Populus_nigra_1990213565.jpg",
719 "Populus_nigra_1990223115.jpg",
720 "Pinus_sylvestris_1148621484.jpg",
721 "Pinus_sylvestris_1148621486.jpg",
722 "Pinus_sylvestris_1148621534.jpg",
723 "Pinus_sylvestris_1148621598.jpg",
724 "Pinus_sylvestris_1148625640.jpg",
725 "Pinus_sylvestris_1148625682.jpg",
726 "Pinus_sylvestris_1837903466.jpg",
727 "Pinus_sylvestris_1936648406.jpg",
728 "Salix_caprea_1099092089.jpg",
729 "Salix_caprea_1099092090.jpg",
730 "Salix_caprea_1099092091.jpg",
731 "Salix_caprea_1099092092.jpg",
732 "Salix_caprea_1099092102.jpg",
733 "Salix_caprea_1099092103.jpg",
734 "Salix_caprea_1099092104.jpg",
735 "Salix_caprea_1099092219.jpg",
736 "Salix_caprea_1701773018.jpg",
737 "Salix_caprea_2417022323.jpg",
738 "Salix_cinerea_subsp._oleifolia_1928696182.jpg",
739 "Salix_cinerea_subsp._oleifolia_1999028748.jpg",
740 "Salix_cinerea_subsp._oleifolia_2595694694.jpg",
741 "Salix_fragilis_1148621287.jpg",
742 "Salix_fragilis_1699402633.jpg",
743 "Salix_fragilis_1848984629.jpg",
744 "Salix_pentandra_2411719250.jpg",
745 "Salix_pentandra_2411719250.jpg",
746 "Salix_viminalis_2417023677.jpg",
747 "Salix_viminalis_2517263988.jpg",
748 "Sorbus_torminalis_2514166021.jpg",
749 "Sorbus_torminalis_2516014023.jpg",
750 "Sorbus_torminalis_2517601760.jpg",
751 "Sorbus_torminalis_2517893419.jpg",
752 "Sorbus_torminalis_575067571.jpg",
753 "Tilia_cordata_1122936454.jpg",
754 "Tilia_cordata_2517731939.jpg",
755 "Tilia_cordata_1949285287.jpg",
756 "Tilia_cordata_1122936977.jpg",
757 "Taxus_baccata_1701800192.jpg",
758 "Taxus_baccata_574678811.jpg",
759 "Taxus_baccata_575033930.jpg",
760 "Taxus_baccata_2517004045.jpg",
761 "Tilia_x_europaea_453662249.jpg",
762 "Tilia_x_europaea_1839797597.jpg",
763 "Tilia_x_europaea_2236814175.jpg",
```

```

764     "Tilia_x_europaea_2457118221.jpg",
765     "Tilia_x_europaea_2513382435.jpg",
766     "Tilia_x_europaea_2513382821.jpg",
767     "Tilia_x_europaea_2513384476.jpg",
768     "Tilia_x_europaea_2513385934.jpg",
769     "Tilia_x_europaea_2513434603.jpg",
770     "Tilia_x_europaea_2513435830.jpg",
771     "Tilia_x_europaea_2513435892.jpg",
772     "Tilia_x_europaea_2513436372.jpg",
773     "Ulmus_glabra_1122937220.jpg",
774     "Ulmus_glabra_1122937299.jpg",
775     "Ulmus_glabra_1701801877.jpg",
776     "Ulmus_glabra_1801063529.jpg",
777     "Ulmus_glabra_2516827297.jpg",
778     "Ulmus_glabra_2517264893.jpg",
779     "Ulmus_platyphyllos_1148624480.jpg",
780     "Ulmus_platyphyllos_1148624501.jpg",
781     "Ulmus_platyphyllos_1148789482.jpg",
782     "Ulmus_platyphyllos_1949285290.jpg",
783     "Ulmus_platyphyllos_1949285366.jpg",
784     "Viburnum_opulus_1697696217.jpg",
785     "Viburnum_opulus_1802796761.jpg",
786     "Viburnum_opulus_1805284773.jpg",
787     "Viburnum_opulus_1978879457.jpg",
788     "Viburnum_opulus_1988701454.jpg",
789     "Viburnum_opulus_2515327519.jpg",
790     "Ulmus_procera_731329357.jpg",
791     "Ulmus_procera_1099106169.jpg",
792     "Ulmus_procera_1099106179.jpg",
793     "Ulmus_procera_1321126987.jpg",
794     "Ulmus_procera_1424173144.jpg",
795     "Ulmus_procera_1928578800.jpg",
796     "Ulmus_procera_1930439448.jpg",
797     "Ulmus_procera_1930453152.jpg",
798     "Ulmus_procera_1977816549.jpg",
799     "Ulmus_procera_1986967729.jpg",
800     "Ulmus_procera_1998579678.jpg",
801     "Ulmus_procera_1998851816.jpg",
802     "Ulmus_procera_1999228440.jpg",
803     "Ulmus_procera_2236814366.jpg",
804     "Ulmus_procera_2397713222.jpg",
805     "Ulmus_procera_2513395783.jpg",
806     "Ulmus_procera_2513396922.jpg",
807     "Ulmus_procera_2513397955.jpg",
808     "Ulmus_procera_2513398892.jpg",
809     "Ulmus_procera_2513878521.jpg",
810     "Ulmus_procera_2514753522.jpg",
811     "Ulmus_procera_2515117606.jpg",
812     "Ulmus_procera_2515152896.jpg",
813     "Ulmus_procera_2515341442.jpg",
814     "Ulmus_procera_2515391156.jpg",
815     "Ulmus_procera_2517302540.jpg",
816     "Ulmus_procera_2573080092.jpg",
817     "Ulmus_procera_2575028568.jpg",
818     "Ulmus_procera_2575046562.jpg",
819     "Ulmus_procera_2575072832.jpg"]
820
821 if __name__ == "__main__":
822
823     # SOURCE DIR
824     base_path = "data/images/image_preprocessing/"
825     directory = "cropped_images"
826     source_dir = os.path.join(base_path, directory)
827
828     # DESTINATION DIR (keep in the last saved images)
829     base_path = "data/images/image_preprocessing/"
830     directory = "cropped_images"
831     destination_dir = os.path.join(base_path, directory)
832
833     # List of faulty images
834     faulty_images = faulty_images_list()
835     print(f"{len(faulty_images)} faulty images")
836
837     # Remove faulty images
838     remove_files(source_dir, faulty_images)

```

data/preprocessing/6a_split_train_val_test.py

```

1 import os
2 import random
3 from data.file_handler import copy_dir
4
5
6 def split_train_val_test(directory, train_size, val_size, test_size):
7     train_dir = os.path.join(directory, "train")
8     val_dir = os.path.join(directory, "val")
9     test_dir = os.path.join(directory, "test")
10    # Create train, validation and test folders
11    if not os.path.exists(train_dir):
12        os.system(f"mkdir {train_dir}")
13    if not os.path.exists(val_dir):
14        os.system(f"mkdir {val_dir}")
15    if not os.path.exists(test_dir):
16        os.system(f"mkdir {test_dir}")
17    # Separate images in train and test folders
18    for folder in os.listdir(directory):
19        # Rename species folder with underscores

```

```

20     species_words = folder.split(" ")
21     species_folder = '_'.join(species_words)
22     os.rename(os.path.join(directory, folder), os.path.join(directory, species_folder))
23     if os.path.exists(os.path.join(directory, species_folder)):
24         if (species_folder != "train") and (species_folder != "val") and (species_folder != "test"):
25             # List all files in each species folder
26             species_files = os.listdir((os.path.join(directory, species_folder)))
27             # Get train and test sizes per species folder
28             n_train = round(len(species_files)*(train_size/100))
29             n_val = round(len(species_files)*(val_size/100))
30             n_test = round(len(species_files)*(test_size/100))
31             # Select train and test files
32             random.seed(seed)
33             train_files = random.sample(species_files, n_train)
34             val_files = random.sample((list(set(species_files)) - set(train_files)),n_val)
35             test_files = list(set(species_files) - set(train_files) - set(val_files))
36             # Move files to folders
37             for file in train_files:
38                 if not os.path.exists(f"{train_dir}/{species_folder}"):
39                     os.system(f"mkdir {train_dir}/{species_folder}")
40                     os.rename(os.path.join(directory, species_folder, file), os.path.join(train_dir,
41                                     species_folder, file))
42                     for file in val_files:
43                         if not os.path.exists(f"{val_dir}/{species_folder}"):
44                             os.system(f"mkdir {val_dir}/{species_folder}")
45                             os.rename(os.path.join(directory, species_folder, file), os.path.join(val_dir,
46                                     species_folder, file))
47                             for file in test_files:
48                                 if not os.path.exists(f"{test_dir}"):
49                                     os.system(f"mkdir {test_dir}")
50                                     os.rename(os.path.join(directory, species_folder, file), os.path.join(test_dir,
51                                         species_folder) files splitted.")
52             else:
53                 print(f"Path not found: {os.path.join(directory, folder)}")
54
55
56 if __name__ == "__main__":
57     # CONFIGURATION
58     # Image directories
59     base_path = "data/images/image_preprocessing/"
60     ##########
61     src_directory = "cropped_images"
62     dest_directory = "processed_images_train_val_test"
63     src_path = os.path.join(base_path, src_directory)
64     dest_path = os.path.join(base_path, dest_directory)
65     #####
66     # Train and test sizes
67     train_size = 60
68     val_size = 20
69     test_size = 20
70     # Seed for the random image sampling
71     seed = 1234
72     #####
73
74     # Copy source dir to empty destination dir
75     copy_dir(src_path, dest_path)
76
77     # Split images dataset
78     split_train_val_test(dest_path, train_size, val_size, test_size)

```

data/preprocessing/6b_split_train_test.py

```

1 import os
2 import random
3 from data.file_handler import copy_dir
4
5
6 def split_train_test(directory, train_size, test_size):
7     train_dir = os.path.join(directory, "train")
8     test_dir = os.path.join(directory, "test")
9     # Create train, validation and test folders
10    if not os.path.exists(train_dir):
11        os.system(f"mkdir {train_dir}")
12
13    if not os.path.exists(test_dir):
14        os.system(f"mkdir {test_dir}")
15    # Separate images in train and test folders
16    for folder in os.listdir(directory):
17        # Rename species folder with underscores
18        species_words = folder.split(" ")
19        species_folder = '_'.join(species_words)
20        os.rename(os.path.join(directory, folder), os.path.join(directory, species_folder))
21        if os.path.exists(os.path.join(directory, species_folder)):
22            if (species_folder != "train") and (species_folder != "test"):
23                # List all files in each species folder
24                species_files = os.listdir((os.path.join(directory, species_folder)))
25                # Get train and test sizes per species folder
26                n_train = round(len(species_files)*(train_size/100))
27                n_test = round(len(species_files)*(test_size/100))
28                # Select train and test files
29                random.seed(seed)
30                train_files = random.sample(species_files, n_train)
31                test_files = list(set(species_files) - set(train_files))
32                # Move files to folders
33                for file in train_files:

```

```

34         if not os.path.exists(f"{train_dir}/{species_folder}"):
35             os.system(f"mkdir {train_dir}/{species_folder}")
36             os.rename(os.path.join(directory, species_folder, file), os.path.join(train_dir,
37             species_folder, file))
38             for file in test_files:
39                 if not os.path.exists(f"{test_dir}"):
40                     os.system(f"mkdir {test_dir}")
41                     os.rename(os.path.join(directory, species_folder, file), os.path.join(test_dir, file))
42             # Remove empty folders
43             os.rmdir(os.path.join(directory, species_folder))
44             print(f"{species_folder} files splitted.")
45     else:
46         print(f"Path not found: {os.path.join(directory, folder)}")
47
48 if __name__ == "__main__":
49     # CONFIGURATION
50     # Image directories
51     base_path = "data/images/image_preprocessing/"
52 ##########
53     src_directory = "cropped_images"
54     dest_directory = "processed_images_train_test"
55     src_path = os.path.join(base_path, src_directory)
56     dest_path = os.path.join(base_path, dest_directory)
57 ##########
58     # Train and test sizes
59     train_size = 80
60     test_size = 20
61     # Seed for the random image sampling
62     seed = 1234
63 ##########
64     # Copy source dir to empty destination dir
65     copy_dir(src_path, dest_path)
66
67     # Split images dataset
68     split_train_test(dest_path, train_size, test_size)
69

```

D.2.2. Datos geográficos

data/geodata/preprocessing/filter_coordinates.py

```

1 import pandas as pd
2 import os
3 import matplotlib.pyplot as plt
4 from data.search import filter_hash
5 from data.config import geodata_filter, species_list
6 from data.file_handler import create_dataframe_from_csv, column_to_list
7 from data.geodata.plot_coordinates import plot_occurrences_map, plot_coordinates_frequency
8 from data.geodata.plot_coordinates_per_species import plot_coordenates_count, unique, get_species_list,
9     get_occurrence_count
10
11 def which_none(column_as_list):
12     """
13     Return the position of elements in a list that are None
14     or inform if all elements are None
15     or there are not None values.
16     """
17     series = pd.Series(column_as_list)
18     series_none = series.isna()
19     if all(series_none):
20         print("All items are None in list.") # then column should be dropped
21         return
22     else:
23         index_list = []
24         for i in range(0, len(series_none)):
25             if series_none[i]:
26                 index_list.append(i)
27             else:
28                 continue
29         if len(index_list)==0:
30             print("There are not None values in list.")
31             return
32         else:
33             print(f"{len(index_list)} None items.") # then rows should be dropped
34             return index_list
35
36 def which_low_precision(column_as_list, max_uncertainty):
37     """
38     Return the position of elements in a list
39     for uncertainty greater than
40     """
41     unprecise_occurrences = [i for i, val in enumerate(column_as_list) if val > max_uncertainty]
42     print(f"{len(unprecise_occurrences)} occurrences have uncertainty above {max_uncertainty} m.")
43     return unprecise_occurrences
44
45
46 def drop_rows_by_index(coordinates_df, rows):
47     """
48     Drop rows in the dataframe that have None values.
49     """

```

```

50     filtered_df = coordinates_df
51     if rows is not None:
52         for row in rows:
53             filtered_df = filtered_df.drop(row)
54             print(f"Row {row} removed from data set.")
55     return filtered_df
56 else:
57     print("No rows to remove.")
58 return coordinates_df
59
60
61 def drop_duplicate_coordinates(coordinates_df):
62     """
63     Return a dataframe with no duplicates coordinates
64     for the same species name.
65     """
66     no_duplicates = coordinates_df.drop_duplicates(["species_name", "longitude", "latitude"])
67     print(f"Removed {len(coordinates_df)-len(no_duplicates)} duplicated rows.")
68     return no_duplicates
69
70
71
72
73
74 if __name__ == "__main__":
75
76     # Open CSV file with the occurrences data.
77     base_path = "data/geodata/request/outputs/"
78     filter_hash = filter_hash(geodata_filter, species_list.species_list)
79     csv_file_name = filter_hash + ".geodata.csv"
80     df = create_dataframe_from_csv(base_path+csv_file_name)
81
82     # Extract list with coordinates
83     latitude = column_to_list(df, "decimal_latitude")
84     longitude = column_to_list(df, "decimal_longitude")
85     species_name = column_to_list(df, "species_name")
86     uncertainty = column_to_list(df, "coordinate_uncertainty")
87
88     coordinates_df = pd.DataFrame({
89         "species_name": species_name,
90         "latitude": latitude,
91         "longitude": longitude,
92         "coordinate_uncertainty": uncertainty
93     })
94
95     # Find empty coordinates
96     filtered_df = coordinates_df
97     print("Filter latitude:")
98     filtered_df = drop_rows_by_index(filtered_df, which_none(latitude))
99     print("Filter longitude:")
100    filtered_df = drop_rows_by_index(filtered_df, which_none(longitude))
101    print("Filter uncertainty:")
102    filtered_df = drop_rows_by_index(filtered_df, which_none(uncertainty))
103
104    # Low precision (10 000 m)
105    max_uncertainty = 10000
106    which_low_precision(uncertainty, max_uncertainty)
107    filtered_df = drop_rows_by_index(filtered_df, which_low_precision(uncertainty, max_uncertainty) )
108
109    # Duplicates
110    filtered_df = drop_duplicate_coordinates(filtered_df)
111
112    # Filtered rows
113    print(f"{len(coordinates_df)-len(filtered_df)} rows removed from geodata data set.")
114
115    # Save filtered CSV
116    save_dir = "data/geodata/preprocessing/outputs"
117    if not os.path.exists(save_dir):
118        os.makedirs(save_dir, exist_ok=True)
119
120    csv_file_name = os.path.join(save_dir, "filtered_coordinates.csv")
121    filtered_df.to_csv(csv_file_name, sep = ",", header = True, index = None, encoding="utf-8")
122    print(f"{csv_file_name} created.")
123
124    # Plot filtered occurrences location
125    destination_path = os.path.join(save_dir, "filtered_occurrences_per_class.png")
126    shape_file = "data/geodata/uk_maps/GBR_adm0.shp" # UK map
127    plot_occurrences_map(filtered_df, shape_file, destination_path)
128
129    # Plot occurrence count
130    path_to_plot = os.path.join(save_dir, "filtered_distribution_map.png")
131    geodata = filtered_df
132    species_name = unique(get_species_list(geodata))
133    occurrence_count = get_occurrence_count(get_species_list(geodata))
134    plot_coordenates_count(occurrence_count, species_name, path_to_plot)
135
136    # Plot coordinates histogram
137    destination_path = os.path.join(save_dir, "filtered_histogram.png")
138    plot_coordinates_frequency(coordinates_df, destination_path)

```

Apéndice E

Código de los clasificadores

image_classifier_sample

June 25, 2020

1 Template

```
[8]: import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.utils import plot_model
import os
import numpy as np
import matplotlib.pyplot as plt
from contextlib import redirect_stdout
import pandas as pd
import pydot_ng as pydot
```

1.1 Getting and saving the data

```
[9]: def get_local_repository_path(repository_name):
    """
    Return local absolute path from home directory
    to the repository folder (including it).
    Arg.: Name of the repository.
    """
    wd_path = os.getcwd()
    split_wd_path = wd_path.split("/")
    tfm_position = split_wd_path.index(repository_name)
    local_path_split = split_wd_path[:tfm_position+1]
    return "/".join(local_path_split)

[10]: # CONFIGURE
local_path = get_local_repository_path("tfm")
model_name = "example"
```

1

```
[11]: # DATA SET DIRECTORIES
source_dir = "data/images/image_preprocessing/processed_images_train_val_test/"
train_dir = os.path.join(local_path, source_dir, "train")
val_dir = os.path.join(local_path, source_dir, "val")
test_dir = os.path.join(local_path, source_dir, "test")

[12]: # OUTPUTS
save_dir = os.path.join(os.path.abspath(os.getcwd()), "outputs", model_name)
# Create outputs folder
if not os.path.exists(save_dir):
    os.makedirs(save_dir, exist_ok=True)

[13]: # LABELS
class_names = sorted(os.listdir(train_dir))
print(class_names)

['Acer_campestre', 'Alnus_glutinosa', 'Betula_pendula', 'Betula_pubescens',
'Buxus_sempervirens', 'Carpinus_betulus', 'Cornus_sanguinea',
'Corylus_avellana', 'Crataegus_laevigata', 'Crataegus_monogyna',
'Euonymus_europaea', 'Fagus_sylvatica', 'Frangula_alnus', 'Fraxinus_excelsior',
'Ilex_aquifolium', 'Juniperus_communis', 'Malus_sylvestris', 'Pinus_sylvestris',
'Populus_nigra', 'Populus_tremula', 'Prunus_avium', 'Prunus_padus',
'Prunus_spinosa', 'Pyrus_cordata', 'Quercus_petraea', 'Quercus_robur',
'Rhamnus_cathartica', 'Salix_caprea', 'Salix_cinerea_subsp._oleifolia',
'Salix_fragilis', 'Salix_pentandra', 'Salix_viminalis', 'Sambucus_nigra',
'Sorbus_aucuparia', 'Sorbus_rupicola', 'Sorbus_torminalis', 'Taxus_baccata',
'Tilia_cordata', 'Tilia_platyphyllus', 'Tilia_x_europaea', 'Ulmus_glabra',
'Ulmus_procera', 'Viburnum_opulus']

[14]: # EDIT FOR EACH MODEL
# Model description
model_description = f"""
{model_name}

"""

# Save model description
with open(os.path.join(save_dir, "model_description.txt"), "w") as file:
    with redirect_stdout(file):
        print(model_description)
```

2

1.2 Image decodification

64

```
[15]: # CONFIGURATION ImageDataGenerator
img_height = 224
img_width = 224
color_mode= "rgb"
class_mode="categorical"
shuffle=True
seed = 1234

[16]: def plot_images(images_arr):
    fig, axes = plt.subplots(1, 6, figsize=(15,15))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

1.2.1 Without augmentation

```
[17]: train_datagen_no_aug = ImageDataGenerator(rescale=1./255)
train_array_no_aug = train_datagen_no_aug.flow_from_directory(directory = train_dir,
                                                               target_size=(img_width, img_height),
                                                               color_mode = color_mode,
                                                               shuffle = shuffle,
                                                               class_mode = class_mode,
                                                               subset = "training",
                                                               seed=seed
                                                               )
```

Found 6842 images belonging to 43 classes.

```
[18]: sample_training_images, _ = next(train_array_no_aug)
plot_images(sample_training_images[:6])
```



3

1.2.2 Applying augmentation

Aim: increase the number of examples by randomly applying transformations to the original images.
It also prevents overfitting of the model.

Fill mode for empty pixels when rotating the image is set to "reflect", so that, being the letters between brackets the pixels of the image, the area outside is filled as follow: abcdcdba|abcd|dcbaabcd

```
[19]: # Data augmentation in train dataset
train_datagen = ImageDataGenerator(rescale=1./255,
                                    brightness_range = [0.2,1.5],
                                    zoom_range = [0.5,1.0],
                                    rotation_range=45,
                                    horizontal_flip=True,
                                    vertical_flip=True,
                                    shear_range = 0.2,
                                    fill_mode = "reflect")

train_array = train_datagen.flow_from_directory(directory = train_dir,
                                                target_size=(img_width, img_height),
                                                color_mode = color_mode,
                                                shuffle = shuffle,
                                                class_mode = class_mode,
                                                #subset = "training",
                                                seed=seed
                                                )
```

Found 6842 images belonging to 43 classes.

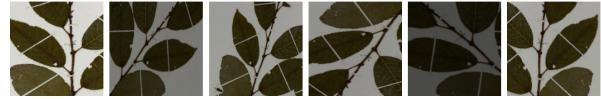
```
[20]: # Validation dataset
val_datagen = ImageDataGenerator(rescale=1./255)

validation_array = val_datagen.flow_from_directory(val_dir,
                                                    target_size=(img_width, img_height),
                                                    color_mode = color_mode,
                                                    class_mode= class_mode,
                                                    #subset='validation',
                                                    seed=seed)
```

Found 2282 images belonging to 43 classes.

```
[21]: augmented_images = [train_array[1][0][0] for i in range(6)]
plot_images(augmented_images)
```

4



1.3 Creating and training the model

1.3.1 Creating the model with Keras

In Keras the model is defined with the `Sequential` method as a linear stack of layers. The `input` layer is implicit in the first layer (a network with 3 layers will have 2 in `keras Sequential` method).

The `input shape` is into the first layer. The model inputs are the tensors or arrays. Images have 3 dimensions: `width, height` and `channels`. The width and the height are measured in pixels and the channels reference the color values (the channel value is 1 if it is in black and white and 3 if it is color in RGB (Red, Green, blue) or HSV (hue, saturation, value) formats - 2 and 4 are black and white or color with an alpha channel (transparency)).

The `activation function` that has to be specified in each layer transforms the input data so that the output doesn't have a linear relation with the input.

1.3.2 Transfer learning with Keras

Aim: to use a pre-existing model that has performed well carrying out a similar task.

VGG16 Ref: 2014 ImageNet competition

```
[22]: # load pre-trained model with the weights
vgg16_model = tf.keras.applications.VGG16()

[1]: vgg16_model.summary()

[24]: # Add the layers of vgg16 model to a new sequential model
model = Sequential()
for layer in vgg16_model.layers[:-1]: # remove last layer
    model.add(layer)
# Rename model
model._name = model_name
model.name
```

```
[24]: 'example'
```

5

```
[25]: # Freeze the weights in the layers of blocks 1 and 2
limit_layer = 11
for layer in model.layers[:limit_layer]:
    layer.trainable = False
for layer in model.layers[limit_layer:]:
    layer.trainable = True
# Add last layer for categories
model.add(Dense(len(class_names), activation = "softmax"))

[26]: model.summary()

# Save model summary
with open(os.path.join(save_dir,"model_summary.txt"), "w") as file:
    with redirect_stdout(file):
        model.summary()
```

```
Model: "example"
-----  

Layer (type)          Output Shape         Param #  

-----  

block1_conv1 (Conv2D)   (None, 224, 224, 64)    1792  

-----  

block1_conv2 (Conv2D)   (None, 224, 224, 64)    36928  

-----  

block1_pool (MaxPooling2D) (None, 112, 112, 64)    0  

-----  

block2_conv1 (Conv2D)   (None, 112, 112, 128)   73856  

-----  

block2_conv2 (Conv2D)   (None, 112, 112, 128)   147584  

-----  

block2_pool (MaxPooling2D) (None, 56, 56, 128)    0  

-----  

block3_conv1 (Conv2D)   (None, 56, 56, 256)    295168  

-----  

block3_conv2 (Conv2D)   (None, 56, 56, 256)    590080  

-----  

block3_conv3 (Conv2D)   (None, 56, 56, 256)    590080  

-----  

block3_pool (MaxPooling2D) (None, 28, 28, 256)    0  

-----  

block4_conv1 (Conv2D)   (None, 28, 28, 512)    1180160  

-----  

block4_conv2 (Conv2D)   (None, 28, 28, 512)    2359808  

-----  

block4_conv3 (Conv2D)   (None, 28, 28, 512)    2359808  

-----  

block4_pool (MaxPooling2D) (None, 14, 14, 512)    0
```

6

```

block5_conv1 (Conv2D)      (None, 14, 14, 512)      2359808
block5_conv2 (Conv2D)      (None, 14, 14, 512)      2359808
block5_conv3 (Conv2D)      (None, 14, 14, 512)      2359808
block5_pool (MaxPooling2D) (None, 7, 7, 512)        0
flatten (Flatten)         (None, 25088)            0
fc1 (Dense)              (None, 4096)             102764544
fc2 (Dense)              (None, 4096)             16781312
dense_1 (Dense)          (None, 43)               176171
=====
Total params: 134,436,715
Trainable params: 131,521,067
Non-trainable params: 2,915,648
-----
The non-trainable parameters are no longer 0, since it has been selected to freeze the weights of
blocks 1 and 2.
[ ]: # Plot model architecture and save it as .png
rankdir = "TB" # TB: vertical; LR: horizontal
plot_model(model, to_file = os.path.join(save_dir,"model_plot.png"),
show_shapes=True, show_layer_names = True, rankdir = rankdir)

```

1.3.3 Training the model

```

[27]: # CONFIGURE
batch_size = len(train_array)//4
epochs = 300
steps_per_epoch = 4

[28]: # COMPILING THE MODEL
# SparseCategoricalCrossentropi directly uses classes labels,
## so that they don't need to be numerically encoded.
optimizer = "sgd" # Options: "sgd", "adam"
model.compile(optimizer=optimizer,
              loss = "categorical_crossentropy",
              metrics=['accuracy'])

```

7

```

[ ]: # Early stopping (when loss does not fall anymore to avoid overfitting)
callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience = 30)

# Checkpoint to save model weights and history before it stops training
checkpoint_filepath = os.path.join(save_dir, "/tmp/checkpoint")
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath = u
                                                               <--checkpoint_filepath,
                                                               save_weights_only=u
                                                               <= True,
                                                               monitor=u
                                                               <--"val_acc",
                                                               save_best_only =u
                                                               <True)

[ ]: # TRAINING THE MODEL
history = model.fit_generator(
    train_array,
    #batch_size = batch_size,
    steps_per_epoch= 4,
    epochs=epochs,
    verbose=1, # get a progress bar and ETA
    validation_data=validation_array,
    validation_steps=2, # batch_size
    callbacks = [callback, model_checkpoint_callback]
)

# Save model history to csv
history_df = pd.DataFrame(history.history)
history_df.to_csv(os.path.join(save_dir, "model_history.csv"), sep=",", u
                  <--index=False)

# Save model weights
model.save_weights(os.path.join(save_dir, "weights.h5"))

# Save model
# model.save(os.path.join(save_dir, "model.h5"))

```

1.3.4 Evaluating the model training

Accuracy and loss during training

```

[ ]: # Parameters measured during model training
history_dict = history.history
print(history_dict.keys())

```

8

```
[1]: try:
    acc = history_dict["acc"]
    val_acc = history_dict["val_acc"]
    loss = history_dict["loss"]
    val_loss = history_dict["val_loss"]
except:
    try:
        acc = history_dict["accuracy"]
        val_acc = history_dict["val_accuracy"]
        loss = history_dict["loss"]
        val_loss = history_dict["val_loss"]
    except:
        pass

[1]: def plot_acc_loss(acc, val_acc, loss, val_loss, epochs):
    epochs_range = range(epochs)
    plt.figure(figsize=(8, 8))
    plt.suptitle(model_name)
    # Accuracy plots
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc*100, label="Training Accuracy")
    plt.plot(epochs_range, val_acc*100, label="Validation Accuracy")
    plt.legend(loc="lower right")
    plt.xlabel("Epochs")
    plt.ylabel("%")
    plt.title("Training and Validation Accuracy")
    # Loss plots
    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label="Training Loss")
    plt.plot(epochs_range, val_loss, label="Validation Loss")
    plt.legend(loc="upper right")
    plt.title("Training and Validation Loss")
    plt.xlabel("Epochs")
    plt.savefig(os.path.join(save_dir, "acc_loss_plot.png"))
    plt.show()

[1]: plot_acc_loss(acc, val_acc, loss, val_loss, epochs)
```

Overfitting When the model predicts significantly better the training set than the validation set, it is a sign of overfitting.

To get back the accuracy and loss data:

- Open the CSV with the model history.
- Save it to a dictionary.

```
history_df = pd.read_csv(os.path.join(save_dir, "model_history.csv"))
history_dict = history_df.to_dict()
```

9

```
try: # the key names vary across tf versions
    acc = np.array(list(history_dict["acc"].values()))
    val_acc = np.array(list(history_dict["val_acc"].values()))
    loss = np.array(list(history_dict["loss"].values()))
    val_loss = np.array(list(history_dict["val_loss"].values()))
    epochs_range = np.array(range(epochs))
except:
    try:
        acc = np.array(list(history_dict["accuracy"].values()))
        val_acc = np.array(list(history_dict["val_accuracy"].values()))
        loss = np.array(list(history_dict["loss"].values()))
        val_loss = np.array(list(history_dict["val_loss"].values()))
        epochs_range = np.array(range(epochs))
    except:
        pass
plot_acc_loss(acc, val_acc, loss, val_loss, epochs)
```

1.4 Evaluating the model

1.5 Evaluating the model

```
[3]: # LABELS
local_path = get_local_repository_path("tfm")
source_dir = "data/images/image_preprocessing/processed_images_train_val_test/"
train_dir = os.path.join(local_path, source_dir, "train")
test_dir = os.path.join(local_path, source_dir, "test")
class_names = sorted(os.listdir(train_dir))
print(class_names)

['Acer_campestre', 'Alnus_glutinosa', 'Betula_pendula', 'Betula_pubescens',
'Buxus_sempervirens', 'Carpinus_betulus', 'Cornus_sanguinea',
'Corylus_avellana', 'Crataegus_laevigata', 'Crataegus_monogyna',
'Euonymus_europaea', 'Fagus_sylvatica', 'Frangula_alnus', 'Fraxinus_excelsior',
'Ilex_aquifolium', 'Juniperus_communis', 'Malus_sylvestris', 'Pinus_sylvestris',
'Populus_nigra', 'Populus_tremula', 'Prunus_avium', 'Prunus_padus',
'Prunus_spinosa', 'Pyrus_cordata', 'Quercus_petraea', 'Quercus_robur',
'Rhamnus_cathartica', 'Salix_caprea', 'Salix_cinerea_subsp._oleifolia',
'Salix_fragilis', 'Salix_pentandra', 'Salix_viminalis', 'Sambucus_nigra',
'Sorbus_aucuparia', 'Sorbus_rupicola', 'Sorbus_torminalis', 'Taxus_baccata',
'Tilia_cordata', 'Tilia_platyphylls', 'Tilia_x_europaea', 'Ulmus_glabra',
'Ulmus_proceria', 'Viburnum_opulus']
```

```
[4]: model_name = "vgg16_b1b2b3PT_1000E"
model_weights = model_name + "_weights.h5"
input_dir = os.path.join(local_path, "models/images/model_testing/inputs/")
limit_layer = 11
```

```

input_dir

[4]: '/home/sciapps/Documents/Repos/tfm/models/images/model_testing/inputs/'

[5]: loaded_model = tf.keras.applications.VGG16()
# layers to freeze in model (limit between frozen and not frozen layers)
model = Sequential()
for layer in loaded_model.layers[:-1]: # remove last layer
    model.add(layer)
# Rename model
model._name = model_name
model.name
# Freeze the weights in first blocks
for layer in model.layers[:limit_layer]:
    layer.trainable = False
for layer in model.layers[limit_layer:]:
    layer.trainable = True
# Add last layer for categories
model.add(Dense(len(class_names), activation = "softmax"))

WARNING:tensorflow:From
/home/sciapps/Documents/Repos/tfm/env/lib/python3.6/site-
packages/tensorflow/python/ops/init_ops.py:1251: calling
VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is
deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor

[6]: model.load_weights(os.path.join(input_dir, model_weights))

```

1.5.1 Get the test dataset

```

[7]: # Test dataset
img_height = 224
img_width = 224
color_mode = "rgb"
seed = 1234
test_main_dir = os.path.join(local_path, source_dir)
test_batch_size = len(os.listdir(test_dir))
test_datagen = ImageDataGenerator(rescale=1./255)
test_array = test_datagen.flow_from_directory(directory = test_main_dir,
                                              classes = ["test"],
                                              batch_size = □
                                              ↵test_batch_size,

```

11

```

                                              target_size=(img_width, □
                                              ↵img_height),
                                              color_mode = color_mode,
                                              shuffle = False,
                                              class_mode= None,
                                              seed=seed)

Found 2280 images belonging to 1 classes.

[8]: # TEST LABELS
def get_test_labels(test_files):
    """
    Return a list of labels:
    "Genus_species"
    Arg.: a list of file names with the structure:
        "Genus_species_occurrencenumber.jpg",
        where Genus_species is the class name,
    """
    test_labels = []
    for i in range(len(test_files)):
        test_file_split = test_files[i].split("_")
        # Remove occurrence number and file extension
        class_name splitted = test_file_split[:-1]
        class_name = " ".join(class_name splitted)
        test_labels.append(class_name)
    return test_labels

[9]: def test_labels_to_index(test_labels, class_names):
    """
    Return a 1D array of integers with the corresponding
    number for a class.
    Args.: - A list with the class name of each item in
            the test data set.
            - A sorted list with the possible class names.
    Eg.: test_labels[1] = "Buzus_sempervirens" corresponds to index 4
          in the list of class names.
    """
    test_labels_index = []
    for i in range(len(test_labels)):
        ind = class_names.index(test_labels[i])
        test_labels_index.append(ind)
    return np.array(test_labels_index)

[10]: test_files = os.listdir(test_dir)
test_labels = get_test_labels(test_files)
test_labels[:5]

```

12

```
[10]: ['Prunus_padus',
       'Tilia_platyphylllos',
       'Crataegus_laevigata',
       'Ulmus_proceria',
       'Populus_nigra']

[11]: test_labels_index = test_labels_to_index(test_labels, class_names)
       test_labels_index[:5]

[11]: array([21, 38, 8, 41, 18])
```

69

1.5.2 Predict the probability of classifying each class

```
[1]: # Get the probability of predicting each class for each image
predictions = model.predict_generator(test_array, steps=10, verbose=1)

Predictions is a 2D array with a shape: (number of examples in test, number of classes)

[1]: predictions.shape

[1]: # Get predicted class for each example
def predicted_class(predictions):
    """
    Return a 1D array with the predicted class for each example.
    Arg.: 2D array predictions of shape (number of examples, number of classes)
    """
    pred_class = []
    for i in range(len(predictions)):
        higher_prob = max(predictions[i])
        ind, = np.where(np.isclose(predictions[i], higher_prob))
        pred_class.append(ind[0])
    return np.array(pred_class)

pred_class = predicted_class(predictions)
```

1.5.3 Plot the confusion matrix

```
[1]: test_labels_index

[1]: pred_class

[1]: # Build the confusion matrix
cm = tf.math.confusion_matrix(test_labels_index, pred_class)
# Convert from tensor to array
sess = tf.Session()
```

13

```
conf_mat = sess.run(cm)
conf_mat

[1]: def plot_confusion_matrix(cm, class_names, model_name):
    """
    Returns a matplotlib figure containing the plotted confusion matrix.

    Args:
    cm (array, shape = [n, n]): a confusion matrix of integer classes
    class_names (array, shape = [n]): String names of the integer classes
    """
    figure = plt.figure(figsize=(20, 20))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.BuGn)
    plt.title("Confusion matrix - " + model_name, fontsize = 22)
    plt.colorbar()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names, rotation=90)
    plt.yticks(tick_marks, class_names)
    plt.savefig(os.path.join(save_dir, "conf_matrix.png"))

[1]: plot_confusion_matrix(conf_mat, np.array(class_names), model_name)
```

14

geodata_classifier_sample

June 26, 2020

1 Template

```
[1]: import os
import random
from contextlib import redirect_stdout
import pandas as pd
import descartes
import geopandas as gpd
import shapely
from shapely.geometry import Point, Polygon
from shapely import geometry
```



```
[3]: import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
import tensorflow.keras.layers as layers
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.utils import to_categorical
```



```
[4]: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
from sklearn import preprocessing
```



```
[5]: from tensorflow import feature_column
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
```

1.1 Getting and describing the data

```
[6]: def get_local_repository_path(repository_name):
    """
    Return local absolute path from home directory
```

1

to the repository folder (including it).
Arg. : Name of the repository.

```
wd_path = os.getcwd()
split_wd_path = wd_path.split("/")
tfm_position = split_wd_path.index(repository_name)
local_path_split = split_wd_path[:tfm_position+1]
return "/".join(local_path_split)
```



```
[163]: # CONFIGURE
local_path = get_local_repository_path("tfm")
model_name = "H1_64_adam_200E_30P" # Number of hidden layers, optimizer, ↴epochs, checkout patience.
```



```
[164]: # OUTPUTS
save_dir = os.path.join(os.path.abspath(os.getcwd()), "outputs", model_name)
# Create outputs folder
if not os.path.exists(save_dir):
    os.makedirs(save_dir, exist_ok=True)
```



```
[9]: # DATA SET DIRECTORY
source_dir = "data/geodata/preprocessing/outputs"
# Importing the dataset
full_dataset = pd.read_csv(os.path.join(local_path, source_dir, ↴"filtered_coordinates.csv"))
full_dataset.sample(n=5)
```


	species_name	latitude	longitude	coordinate_uncertainty
2760	Prunus avium	53.385637	-2.883056	8.0
3286	Sambucus nigra	53.379623	-2.898182	8.0
1014	Fagus sylvatica	51.770710	0.124433	60.0
6349	Quercus robur	53.390822	-2.879542	5.0
1508	Betula pendula	52.803570	-1.416270	100.0


```
[10]: dataset = full_dataset.drop(columns="coordinate_uncertainty")
dataset.sample(n=5)
```



```
[10]: species_name    latitude    longitude
7833  Salix pentandra  52.307280   0.653500
2658  Prunus avium   53.772768  -2.627577
3920  Viburnum opulus 52.730100  -1.295820
995   Fagus sylvatica 52.452928  -2.175263
785   Populus tremula   52.449070  -2.303060
```

2

1.1.1 Examples per class

71

```
[ ]: def get_species_list(df):
        """
        Return df["species_name"].tolist()

    def get_occurrence_count(item_list):
        item_count = []
        unique_items = unique(item_list)
        for item in unique_items:
            item_count.append(item_list.count(item))
        return item_count

    def unique(item_list):
        unique_list = []
        for item in item_list:
            if item not in unique_list:
                unique_list.append(item)
        return unique_list

    def plot_coordenates_count(occurrence_count, species_name, path_to_plot):
        """
        Plot number of coordenates per class.
        """
        occurrence_count = list(reversed(occurrence_count))
        species_name = list(reversed(species_name))
        plt.figure(figsize=(10,10))
        plt.barh(species_name, occurrence_count, color="#7cd9b7", height=0.8, u
        ↪edgecolor="grey")
        plt.title("Number of georeferences per class")
        # x axis, species
        xint = range(0,(max(occurrence_count)+1),10)
        plt.xticks(xint)
        plt.xlabel("Number of georeferences")
        # y axis, image count
        plt.ylabel("Species", rotation="vertical")
        plt.yticks(fontsize=8)
        # save file
        plt.savefig(path_to_plot)

    species_name = unique(get_species_list(dataset))
    occurrence_count = get_occurrence_count(get_species_list(dataset))
    plot_coordenates_count(occurrence_count, species_name, os.path.join(save_dir, u
    ↪"occurrences_per_class.png"))
```

3

1.1.2 Histogram

```
[ ]: def column_to_list(df, column):
        """
        Return a list of elements of column from a data frame.
        Args.: a dataframe and a name of a column.
        """
        try:
            return df[column].tolist()
        except:
            print(f"Unable to extract {column} from {df}.")"

    def plot_coordinates_frequency(coordinates_df, destination_path):
        """
        Return an histogram with the frequency of the coordinates.
        Arg.: a data frame with latitude and longitude
        """
        latitude = column_to_list(coordinates_df, "latitude")
        longitude = column_to_list(coordinates_df, "longitude")
        fig, ax = plt.subplots(nrows=2, ncols=1, sharex=True, sharey=True, u
        ↪figsize=(8, 8))
        #fig, axs = plt.subplots(2)
        fig.suptitle('Coordinates distribution')
        plt.subplot(2,1,1)
        plt.hist(latitude, bins = 80, alpha = 0.5,
                 color="lightgreen",
                 edgecolor="grey")
        plt.ylabel("Frequency", fontsize=8)
        plt.xlabel("Latitude (decimal degrees)", fontsize=8)
        plt.xticks(fontsize=8)
        plt.yticks(fontsize=8)
        plt.subplot(2,1,2)
        plt.hist(longitude, bins = 80, alpha = 0.5,
                 color="lightpink",
                 edgecolor="grey")
        plt.ylabel("Frequency", fontsize=8)
        plt.xlabel("Longitude (decimal degrees)", fontsize=8)
        plt.xticks(fontsize=8)
        plt.yticks(fontsize=8)
        plt.legend()
        plt.savefig(destination_path)

    plot_coordinates_frequency(dataset,os.path.join(save_dir, "histogram.png"))
```

4

1.1.3 All occurrences map

72

```
[ 1: # UK map
shape_file = os.path.join(local_path,"data/geodata/uk_maps/GBR_adm0.shp") # UK
↪map

def plot_occurrences_map(coordinates_df, shape_file, destination_path):
"""
    Return an occurrences map in .png format.
    Args.: a shape file for a geographical area
    and a dataframw with at least two columns:
    - latitude
    - longitude
    and a path where to save the map.
"""

# Add a new column to the data frame with a combination of coordinates
coordinate_pair = [Point(xy) for xy in zip(coordinates_df.longitude,coordinates_df.latitude)]
geometry_df = coordinates_df[["species_name"]]
geometry_df["geometry"] = coordinate_pair
crs = {"init": 'epsg:4326'}
occurrences_loc = gpd.GeoDataFrame(geometry_df, crs=crs,geometry=coordinate_pair)
occurrences_loc.shape

shape_map = gpd.read_file(shape_file)
fig, ax = plt.subplots(figsize=(10,10))
occurrences_loc.geometry.plot(marker="d", color="green", markersize=1,ax=ax, label = "Species occurrences")
shape_map.plot(color="grey", ax=ax, alpha = 0.2)

plt.title('Species distribution map')
plt.legend()
plt.savefig(destination_path)

plot_occurrences_map(dataset, shape_file,os.path.join(save_dir,
↪"occurrences_map.png"))
```

1.1.4 Occurrences per species maps

```
[ 1: # UK map
shape_file = os.path.join(local_path,"data/geodata/uk_maps/GBR_adm0.shp") # UK
↪map
from random import randint
```

5

```
def plot_clustered_occurrences_map(coordinates_df, shape_file,destination_path, species_list):
"""
    Return an occurrences map in .png format.
    Args.: a shape file for a geographical area
    and a dataframw with at least two columns:
    - latitude
    - longitude
    and a path where to save the map.
"""

coordinates_df = coordinates_df.loc[coordinates_df["species_name"].isin(species_list)]

# Add a new column to the data frame with a combination of coordinates
coordinate_pair = [Point(xy) for xy in zip(coordinates_df.longitude,coordinates_df.latitude)]
geometry_df = coordinates_df.iloc[:,[0]]
geometry_df["geometry"] = coordinate_pair
crs = {"init": "epsg:4326"}
occurrences_loc = gpd.GeoDataFrame(geometry_df, crs=crs,geometry=coordinate_pair)
occurrences_loc.shape

shape_map = gpd.read_file(shape_file)
fig, ax = plt.subplots(figsize = (15,15))
shape_map.plot(color="grey", ax=ax, alpha = 0.2)

# Generating and array of random colors with length equals to number of
↪unique species
# Based on: https://stackoverflow.com/questions/7827530/
↪array-of-colors-in-python
number_species = occurrences_loc["species_name"].nunique() # Getting the
↪number of unique species
colors = []
color_count = 0
for i in range(number_species):
    colors.append('#%06X' % randint(0, 0xFFFFFF))
for species_name in occurrences_loc["species_name"].unique():
    occurrences_loc[occurrences_loc["species_name"] == species_name].geometry.plot(
        marker="o", color=colors[color_count],
        markersize=15,
        edgecolors="grey",
        ax=ax,
        alpha = 0.5,
        label = species_name)
    color_count += 1
```

6

```

plt.title('Species distribution map')
plt.legend()
plt.savefig(destination_path)

[ ]: species_list = ["Quercus robur", "Quercus petraea"]
plot_clustered_occurrences_map(dataset, shape_file,os.path.join(save_dir,„
„clustered_occurrences_map.png”), species_list)

[ ]: species_list = ["Salix pentandra", "Salix fragilis", "Salix caprea", "Salix_
„cinerea subsp. oleifolia"]
species_list = ["Salix pentandra", "Salix cinerea subsp. oleifolia"]
plot_clustered_occurrences_map(dataset, shape_file,os.path.join(save_dir,„
„clustered_occurrences_map.png”), species_list)

[ ]: species_list = ["Prunus padus", "Prunus avium"]
species_list = ["Tilia cordata", "Tilia platyphyllos", "Tilia x europaea"]
species_list = ["Crataegus laevigata", "Crataegus monogyna"]
species_list = ["Sorbus rupicola", "Sorbus torminalis"]
species_list = ["Malus sylvestris", "Prunus padus", "Prunus avium"]

plot_clustered_occurrences_map(dataset, shape_file,os.path.join(save_dir,„
„clustered_occurrences_map.png”), species_list)

[40]: # LABELS
class_names = list(set(column_to_list(dataset, "species_name")))
print(f"{len(class_names)} classes in dataset.")
print(f"Classes names: {class_names}")

43 classes in dataset.
Classes names: ['Sorbus aucuparia', 'Sambucus nigra', 'Salix caprea', 'Prunus
padus', 'Ulmus procera', 'Acer campestre', 'Alnus glutinosa', 'Tilia x
europaea', 'Buxus sempervirens', 'Salix fragilis', 'Prunus avium', 'Betula
pubescens', 'Juniperus communis', 'Quercus robur', 'Salix cinerea subsp.
oleifolia', 'Cornus sanguinea', 'Tilia cordata', 'Salix viminalis', 'Fagus
sylvatica', 'Euonymus europaea', 'Crataegus laevigata', 'Salix pentandra',
'Corylus avellana', 'Fraxinus excelsior', 'Prunus spinosa', 'Pinus sylvestris',
'Viburnum opulus', 'Taxus baccata', 'Carpinus betulus', 'Populus nigra',
'Crataegus monogyna', 'Quercus petraea', 'Rhamnus cathartica', 'Betula pendula',
'Populus tremula', 'Ulmus glabra', 'Ilex aquifolium', 'Sorbus rupicola', 'Malus
sylvestris', 'Pyrus cordata', 'Frangula alnus', 'Tilia platyphyllos', 'Sorbus
torminalis']

[19]: # EDIT FOR EACH MODEL
# Model description
model_description = f"""
{model_name}

"""

# Save model description
with open(os.path.join(save_dir,"model_description.txt"), "w") as file:
    with redirect_stdout(file):
        print(model_description)

```

7

```

{model_name}

"""

# Save model description
with open(os.path.join(save_dir,"model_description.txt"), "w") as file:
    with redirect_stdout(file):
        print(model_description)

```

1.2 Retrieving the data from the dataset

```

[41]: # Encode class names
class_names.sort()
species_to_number = {species_name:class_names.index(species_name) for_
„species_name in class_names}
species_to_number

[41]: {'Acer campestre': 0,
       'Alnus glutinosa': 1,
       'Betula pendula': 2,
       'Betula pubescens': 3,
       'Buxus sempervirens': 4,
       'Carpinus betulus': 5,
       'Cornus sanguinea': 6,
       'Corylus avellana': 7,
       'Crataegus laevigata': 8,
       'Crataegus monogyna': 9,
       'Euonymus europaea': 10,
       'Fagus sylvatica': 11,
       'Frangula alnus': 12,
       'Fraxinus excelsior': 13,
       'Ilex aquifolium': 14,
       'Juniperus communis': 15,
       'Malus sylvestris': 16,
       'Pinus sylvestris': 17,
       'Populus nigra': 18,
       'Populus tremula': 19,
       'Prunus avium': 20,
       'Prunus padus': 21,
       'Prunus spinosa': 22,
       'Pyrus cordata': 23,
       'Quercus petraea': 24,
       'Quercus robur': 25,
       'Rhamnus cathartica': 26,
       'Salix caprea': 27,

```

8

```
'Salix cinerea subsp. oleifolia': 28,
'Salix fragilis': 29,
'Salix pentandra': 30,
'Salix viminalis': 31,
'Sambucus nigra': 32,
'Sorbus aucuparia': 33,
'Sorbus rupicola': 34,
'Sorbus torminalis': 35,
'Taxus baccata': 36,
'Tilia cordata': 37,
'Tilia platyphyllos': 38,
'Tilia x europaea': 39,
'Ulmus glabra': 40,
'Ulmus procera': 41,
'Viburnum opulus': 42}

[42]: dataset['target']=dataset.apply(lambda r:species_to_number[r['species_name']],axis=1)
dataset.sample(n=5)

[42]:
   species_name  latitude  longitude  target
6995  Sorbus aucuparia  52.590508 -2.141861     33
7984  Salix pentandra  55.655410 -3.320670     30
8987  Taxus baccata  50.983538 -2.752582     36
2665   Prunus avium  52.710885 -1.579997     20
8146  Salix fragilis  55.909090 -3.529250     29

[43]: dataset_num = dataset.drop(columns="species_name")
dataset_num.sample(n=5)

[43]:
   latitude  longitude  target
7414  51.769330 -0.557480     10
6459  51.733840 -3.521880     24
7611  52.019411 -4.826572     35
5063  53.441978 -2.631473      5
7886  55.772940 -4.209470     30

[44]: train, test = train_test_split(dataset_num, test_size=0.2)
train, val = train_test_split(train, test_size=0.2)
print(len(train), 'train examples')
print(len(val), 'validation examples')
print(len(test), 'test examples')

5804 train examples
1451 validation examples
1814 test examples
```

9

```
[45]: train.sample(n=5)

[45]:
   latitude  longitude  target
6247  52.646340 -1.270560     25
5271  53.032710 -3.998980     15
1774  52.392750 -0.224210     22
5467  53.408072 -2.836948     39
591   53.440690 -2.326318     13

[25]: # Predictors (latitude and longitude)
X_train = train.iloc[:, 1:3].values
X_val = val.iloc[:, 1:3].values
X_test = test.iloc[:, 1:3].values
print(f"Train predictor shape: {X_train.shape}")
print(f"Validation predictor shape: {X_val.shape}")
print(f"Test predictor shape: {X_test.shape}")
print(X_train)

Train predictor shape: (5804, 2)
Validation predictor shape: (1451, 2)
Test predictor shape: (1814, 2)
[[ -2.027036  2.        ]
 [ -2.534807  7.        ]
 [ -1.845199 33.        ]
 ...
 [  0.9857   37.        ]
 [ -2.88908 18.        ]
 [ -3.84065  4.        ]]

[26]: y_train = train.iloc[:, 2].values
y_val = val.iloc[:, 2].values
test_labels = test.iloc[:, 2].values

y_train = to_categorical(y_train)
y_val = to_categorical(y_val)
y_test = to_categorical(test_labels)
print(f"Train target shape: {y_train.shape}")
print(f"Validation target shape: {y_val.shape}")
print(f"Test target shape: {y_test.shape}")
print(y_train)

Train target shape: (5804, 43)
Validation target shape: (1451, 43)
Test target shape: (1814, 43)
[[0. 0. 1. ... 0. 0.]
 [0. 0. 0. ... 0. 0.]
 [0. 0. 0. ... 0. 0.]
 ...
 [0. 0. 0. ... 0. 0.]]
```

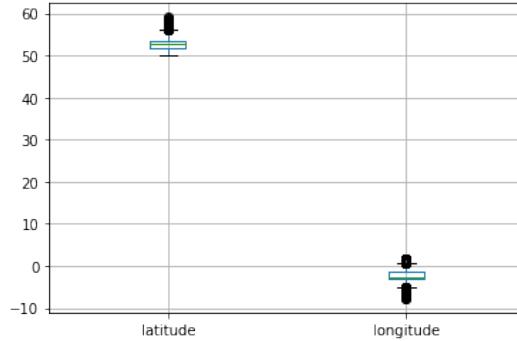
10

```
[0. 0. 0. ... 0. 0. 0.]  
[0. 0. 0. ... 0. 0. 0.]  
[0. 0. 0. ... 0. 0. 0.]]
```

75

1.3 Features normalization

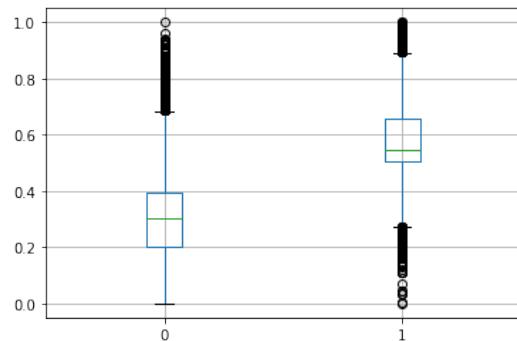
```
[27]: dataset_not_norm = dataset.drop(columns="target")  
dataset_not_norm.boxplot()  
  
[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1f48fcfc88>
```



```
[28]: # Effect of features normalization  
dataset_not_norm = dataset_not_norm.drop(columns="species_name")  
x = dataset_not_norm.values #returns a numpy array  
min_max_scaler = preprocessing.MinMaxScaler()  
x_scaled = min_max_scaler.fit_transform(x)  
dataset_norm = pd.DataFrame(x_scaled)  
dataset_norm.boxplot()
```

```
[28]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1f4483b438>
```

11



```
[29]: X_train_norm = normalize(X_train)  
X_val_norm = normalize(X_val)  
X_train_norm  
  
[29]: array([[-0.71183799,  0.70234371],  
           [-0.34047959,  0.94025191],  
           [-0.05582792,  0.99844041],  
           ...,  
           [ 0.02663109,  0.99964533],  
           [-0.15847612,  0.98736281],  
           [-0.69259282,  0.72132876]])
```

1.4 Creating and training the model

```
[171]: model = Sequential()  
model.add(layers.Flatten(input_shape=(X_train_norm.shape[-1],)))  
model.add(layers.Dense(64, activation='relu'))  
model.add(layers.Dense(len(class_names), activation='softmax'))
```

```
[172]: model.summary()
```

```
Model: "sequential_16"  
-----  
Layer (type)      Output Shape       Param #  
-----
```

12

```

flatten_15 (Flatten)      (None, 2)          0
-----  

dense_46 (Dense)         (None, 64)         192
-----  

dense_47 (Dense)         (None, 43)         2795
-----  

Total params: 2,987  

Trainable params: 2,987  

Non-trainable params: 0
-----  

[173]: # Compiling the ANN  

#opt = tf.keras.optimizers.SGD(learning_rate=0.001)  

model.compile(optimizer = "adam", loss = 'categorical_crossentropy', metrics = u  

↳['accuracy'])  

[174]: # Early stopping (when loss does not fall anymore to avoid overfitting)  

callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience = 30)  

# Checkpoint to save model weights and history before it stops training  

checkpoint_filepath = os.path.join(save_dir, "/tmp/checkpoint")  

model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath = u  

↳checkpoint_filepath,  

save_weights_only=u  

↳= True,  

monitor=u  

↳"val_acc",  

save_best_only =u  

↳True)  

[ 1]: epochs = 200  

history = model.fit(  

    X_train_norm,  

    y_train,  

    validation_data=(X_val_norm, y_val),  

    batch_size = 10,  

    epochs = epochs  

)
[176]: # Save model history to csv  

history_df = pd.DataFrame(history.history)  

history_df.to_csv(os.path.join(save_dir, "model_history.csv"), sep=",",u  

↳index=False)  

# Save model weights  

weights_name = model_name + "_weights.h5"  

model.save_weights(os.path.join(save_dir, weights_name))

```

13

1.4.1 Evaluating the model training

```

Accuracy and loss during training
[177]: # Parameters measured during model training  

history_dict = history.history  

print(history_dict.keys())
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])

[178]: try:  

    acc = history_dict["acc"]  

    val_acc = history_dict["val_acc"]  

    loss = history_dict["loss"]  

    val_loss = history_dict["val_loss"]
except:  

    try:  

        acc = history_dict["accuracy"]  

        val_acc = history_dict["val_accuracy"]  

        loss = history_dict["loss"]  

        val_loss = history_dict["val_loss"]
    except:  

        pass

[179]: def plot_acc_loss(acc, val_acc, loss, val_loss, epochs):
    epochs_range = range(epochs)
    plt.figure(figsize=(8, 8))
    plt.suptitle(model_name)
    # Accuracy plots
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label="Training Accuracy")
    plt.plot(epochs_range, val_acc, label="Validation Accuracy")
    plt.legend(loc="lower right")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.title("Training and Validation Accuracy")
    # Loss plots
    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label="Training Loss")
    plt.plot(epochs_range, val_loss, label="Validation Loss")
    plt.legend(loc="upper right")
    plt.title("Training and Validation Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.savefig(os.path.join(save_dir, "acc_loss_plot.png"))
    plt.show()

[ 1]: plot_acc_loss(acc, val_acc, loss, val_loss, epochs)

```

14

1.5.1 Get the test dataset

```
[185]: test_labels
[185]: array([20, 37, 9, ..., 8, 5, 9])

[186]: X_test
[186]: array([[-2.701556e+00,  2.000000e+01],
           [-3.350930e+00,  3.700000e+01],
           [-2.585800e-02,  9.000000e+00],
           ...,
           [-3.660050e+00,  8.000000e+00],
           [-1.265600e-01,  5.000000e+00],
           [-1.165640e-01,  9.000000e+00]])
```

1.5.2 Predict the probability of classifying each class

```
[187]: # Get the probability of predicting each class for each image
predictions = model.predict(X_test, batch_size = 1)

[188]: predictions.shape
[188]: (1814, 43)

[189]: # Get predicted class for each example
def predicted_class(predictions):
    """
    Return a 1D array with the predicted class for each example.
    Arg.: 2D array predictions of shape (number of examples, number of classes)
    """
    pred_class = []
    for i in range(len(predictions)):
        higher_prob = max(predictions[i])
        ind, = np.where(np.isclose(predictions[i], higher_prob))
        pred_class.append(ind[0])
    return np.array(pred_class)

pred_class = predicted_class(predictions)
```

15

1.5.3 Plot the confusion matrix

```
[190]: test_labels
[190]: array([20, 37, 9, ..., 8, 5, 9])

[191]: pred_class
[191]: array([33, 40, 40, ..., 4, 40, 40])

[192]: # Build the confusion matrix
cm = tf.math.confusion_matrix(test_labels, pred_class)
# Convert from tensor to array
sess = tf.Session()
conf_mat = sess.run(cm)
conf_mat

[192]: array([[24, 13, 0, ..., 0, 0, 0],
           [0, 0, 0, ..., 0, 0, 1],
           [0, 0, 0, ..., 2, 0, 1],
           ...,
           [0, 0, 0, ..., 40, 0, 0],
           [0, 0, 0, ..., 36, 0, 1],
           [0, 0, 0, ..., 42, 0, 0]], dtype=int32)

[193]: def plot_confusion_matrix(cm, class_names, model_name):
    """
    Returns a matplotlib figure containing the plotted confusion matrix.

    Args:
    cm (array, shape = [n, n]): a confusion matrix of integer classes
    class_names (array, shape = [n]): String names of the integer classes
    """
    figure = plt.figure(figsize=(20, 20))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.BuGn)
    plt.title("Confusion matrix - " + model_name, fontsize = 22)
    plt.colorbar()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names, rotation=90)
    plt.yticks(tick_marks, class_names)
    plt.xlabel("Predicted", fontsize = 18)
    plt.ylabel("Actual", fontsize = 18)
    plt.savefig(os.path.join(save_dir,"conf_matrix.png"))

[ ]: plot_confusion_matrix(conf_mat, np.array(class_names), model_name)
```

16

stack_classifier_test

June 26, 2020

1 Test stack image and geodata models

```
[1]: import os
import random
from contextlib import redirect_stdout
import pandas as pd
import descartes
import geopandas as gpd
import shapely
from shapely.geometry import Point, Polygon
from shapely import geometry

[3]: import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras.models import Sequential
import tensorflow.keras.layers as layers
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.utils import plot_model

[4]: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
from sklearn import preprocessing
```

1.1 Getting the data

```
[5]: def get_local_repository_path(repository_name):
    """
    Return local absolute path from home directory
    to the repository folder (including it).
    
```

1

```
Arg.: Name of the repository.
"""
wd_path = os.getcwd()
split_wd_path = wd_path.split("/")
tfm_position = split_wd_path.index(repository_name)
local_path_split = split_wd_path[:tfm_position+1]
return "/".join(local_path_split)

[6]: # CONFIGURE
local_path = get_local_repository_path("tfm")
model_name = "stack_example"

[7]: # OUTPUTS
save_dir = os.path.join(os.path.abspath(os.getcwd()), "outputs", model_name)
# Create outputs folder
if not os.path.exists(save_dir):
    os.makedirs(save_dir, exist_ok=True)

[8]: # INPUTS (for the weights)
input_dir = os.path.join(os.path.abspath(os.getcwd()), "inputs")
geo_weights = "2H_sgd_2000E_100P_weights.h5"
img_weights = "vgg16_b1b2b3PT_1000E_weights.h5"

[9]: # GEOFATA SET DIRECTORY
geo_source_dir = "data/geodata/preprocessing/outputs"
# Importing the dataset
full_dataset = pd.read_csv(os.path.join(local_path, geo_source_dir, "filtered_coordinates.csv"))
dataset = full_dataset.drop(columns="coordinate_uncertainty")

[10]: # DATA SET DIRECTORIES
img_source_dir = "data/images/image_preprocessing/processed_images_train_val_test/"
train_dir = os.path.join(local_path, img_source_dir, "train")
val_dir = os.path.join(local_path, img_source_dir, "val")
test_dir = os.path.join(local_path, img_source_dir, "test")

[11]: # LABELS
img_class_names = sorted(os.listdir(train_dir))
print(f"{len(img_class_names)} classes in dataset.")
print(f"Classes names: {img_class_names}")
```

43 classes in dataset.
 Classes names: ['Acer_campestre', 'Alnus_glutinosa', 'Betula_pendula',
 'Betula_pubescens', 'Buxus_sempervirens', 'Carpinus_betulus',
 'Cornus_sanguinea', 'Corylus_avellana', 'Crataegus_laevigata',
 'Crataegus_monogyna', 'Euonymus_europaea', 'Fagus_sylvatica', 'Frangula_alnus',

2

```

'Fraxinus_excelsior', 'Ilex_aquifolium', 'Juniperus_communis',
'Malus_sylvestris', 'Pinus_sylvestris', 'Populus_nigra', 'Populus_tremula',
'Prunus_avium', 'Prunus_padus', 'Prunus_spinosa', 'Pyrus_cordata',
'Quercus_petraea', 'Quercus_robur', 'Rhamnus_cathartica', 'Salix_caprea',
'Salix_cinerea_subsp._oleifolia', 'Salix_fragilis', 'Salix_pentandra',
'Salix_viminalis', 'Sambucus_nigra', 'Sorbus_aucuparia', 'Sorbus_rupicola',
'Sorbus_torminalis', 'Taxus_baccata', 'Tilia_cordata', 'Tilia_platiphylllos',
'Tilia_x_europaea', 'Ulmus_glabra', 'Ulmus_procera', 'Viburnum_opulus']

[12]: # LABELS
geo_class_names = list(set(full_dataset["species_name"].tolist()))
print(f'{len(geo_class_names)} classes in dataset.')
print(f'Classes names: {geo_class_names}')

43 classes in dataset.
Classes names: ['Salix caprea', 'Corylus avellana', 'Acer campestre', 'Ulmus
glabra', 'Populus nigra', 'Tilia platiphylllos', 'Prunus avium', 'Euonymus
europaea', 'Sambucus nigra', 'Alnus glutinosa', 'Ilex aquifolium', 'Tilia
cordata', 'Crataegus laevigata', 'Salix viminalis', 'Malus sylvestris',
'Carpinus betulus', 'Salix cinerea subsp. oleifolia', 'Quercus robur',
'Crataegus monogyna', 'Ulmus procera', 'Sorbus torminalis', 'Prunus spinosa',
'Pinus sylvestris', 'Pyrus cordata', 'Betula pubescens', 'Fraxinus excelsior',
'Rhamnus cathartica', 'Prunus padus', 'Salix pentandra', 'Taxus baccata',
'Cornus sanguinea', 'Sorbus aucuparia', 'Populus tremula', 'Betula pendula',
'Frangula alnus', 'Fagus sylvatica', 'Sorbus rupicola', 'Viburnum opulus',
'Tilia x europaea', 'Buxus sempervirens', 'Quercus petraea', 'Salix fragilis',
'Juniperus communis']

[13]: # EDIT FOR EACH MODEL
# Model description
model_description = f"""
{model_name}

"""

# Save model description
with open(os.path.join(save_dir,"model_description.txt"), "w") as file:
    with redirect_stdout(file):
        print(model_description)

[14]: # GEODATA SAMPLE
full_dataset = pd.read_csv(os.path.join(local_path, geo_source_dir,
                                         "filtered_coordinates.csv"))
dataset = full_dataset.drop(columns="coordinate_uncertainty")

# Encode class names
geo_class_names.sort()

```

3

```

species_to_number = {species_name:geo_class_names.index(species_name) for
                     species_name in geo_class_names}
dataset['target']=dataset.apply(lambda r:species_to_number[r.
                     species_name],axis=1)
dataset.sample(n=5)
dataset_num = dataset.drop(columns="species_name")
dataset_num.sample(n=5)

# Split in train, val and test subsets
train, test = train_test_split(dataset_num, test_size=0.2)
train, val = train_test_split(train, test_size=0.2)
print(len(train), 'train examples')
print(len(val), 'validation examples')
print(len(test), 'test examples')
# Extract predictors (latitude and longitude)
X_geo_train = train.iloc[:, 1:3].values
X_geo_val = val.iloc[:, 1:3].values
X_geo_test = test.iloc[:, 1:3].values
print(f"Train predictor shape: {X_geo_train.shape}")
print(f"Validation predictor shape: {X_geo_val.shape}")
print(f"Test predictor shape: {X_geo_test.shape}")
print(X_geo_train)

# Encode labels
y_geo_train = train.iloc[:, 2].values
y_geo_val = val.iloc[:, 2].values
y_geo_test = test.iloc[:, 2].values

y_geo_train = to_categorical(y_geo_train)
y_geo_val = to_categorical(y_geo_val)
y_geo_test = to_categorical(y_geo_test)
print(f"Train target shape: {y_geo_train.shape}")
print(f"Validation target shape: {y_geo_val.shape}")
print(f"Test target shape: {y_geo_test.shape}")
print(y_geo_train)

# Normalize train val predictors
X_geo_train_norm = normalize(X_geo_train)
X_geo_val_norm = normalize(X_geo_val)
X_geo_train_norm

5804 train examples
1451 validation examples
1814 test examples
Train predictor shape: (5804, 2)
Validation predictor shape: (1451, 2)
Test predictor shape: (1814, 2)

```

4

```

[[ -2.711519 27.      ]
 [ -2.155155 40.      ]
 [ -2.539228  9.      ]
 ...
 [-0.545627 11.      ]
 [-2.857274  6.      ]
 [-1.54571  40.      ]]
Train target shape: (5804, 43)
Validation target shape: (1451, 43)
Test target shape: (1814, 43)
[[0. 0. 0. .. 0. 0.]
 [0. 0. 0. .. 1. 0. 0.]
 [0. 0. 0. .. 0. 0. 0.]
 ...
 [0. 0. 0. .. 0. 0. 0.]
 [0. 0. 0. .. 0. 0. 0.]
 [0. 0. 0. .. 1. 0. 0.]]
```

```

[14]: array([[-0.099924 ,  0.99499507],
           [-0.05380084,  0.99855169],
           [-0.27153609,  0.96242826],
           ...,
           [-0.04954155,  0.99877206],
           [-0.42994967,  0.90285286],
           [-0.03861393,  0.9992542 ]])
```

```
[15]: # CONFIGURATION ImageDataGenerator
img_height = 224
img_width = 224
color_mode="rgb"
class_mode="categorical"
shuffle=False
seed = 1234

train_datagen = ImageDataGenerator(rescale=1. / 255)
itr = train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    class_mode='categorical')

X_img_train, y_img_train = itr.next()
```

Found 6842 images belonging to 43 classes.

5

1.2 Load models

```

[16]: # GEODATA
# Architecture
geo_model = Sequential()
geo_model.add(layers.Flatten(input_shape=(X_geo_train_norm.shape[-1],)))
geo_model.add(layers.Dense(64, activation='relu'))
geo_model.add(layers.Dense(64, activation='relu'))
geo_model.add(layers.Dense(len(geo_class_names), activation='softmax'))
# Load weights
geo_model.load_weights(os.path.join(input_dir, geo_weights))

WARNING:tensorflow:From
/home/sciapps/Documents/Repos/tfm/env/lib/python3.6/site-
packages/tensorflow/python/ops/init_ops.py:1251: calling
VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is
deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor

[17]: # IMAGES
# Architecture
loaded_model = tf.keras.applications.VGG16()
img_model = Sequential()
for layer in loaded_model.layers[:-1]:
    img_model.add(layer)
img_model.add(Dense(len(img_class_names), activation = "softmax"))
# Load weights
img_model.load_weights(os.path.join(input_dir, img_weights))

[18]: img_model._name = "img_vgg16"
geo_model._name = "geo_dense"

[ ]: img_input = keras.Input(shape=(224,224,3), name="images")
geo_input = keras.Input(shape=(2), name="geodata")

y1 = img_model(img_input)
y2 = geo_model(geo_input)
avg = tf.keras.layers.Average()([y1, y2])
output = tf.keras.layers.Dense(43)(avg)
ensemble_model = keras.Model(inputs=[img_input, geo_input], outputs = output)

[49]: # Define inputs
img_input = keras.Input(shape=(224,224,3), name="images")
geo_input = keras.Input(shape=(2), name="geodata")
```

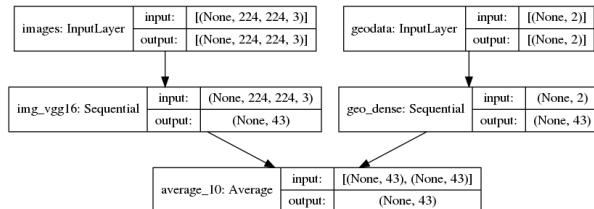
6

```

y1 = img_model(img_input)
y2 = geo_model(geo_input)
output = layers.average([y1,y2])
ensemble_model = keras.Model(inputs=[img_input, geo_input], outputs = output)
keras.utils.plot_model(ensemble_model,show_shapes=True)

```

[49]:



```

[50]: for layer in ensemble_model.layers:
        layer.trainable = False
    ensemble_model.summary()

```

Model: "model_10"

Layer (type)	Output Shape	Param #	Connected to
images (InputLayer)	[(None, 224, 224, 3)]	0	
geodata (InputLayer)	[(None, 2)]	0	
img_vgg16 (Sequential)	(None, 43)	134436715	images[0][0]
geo_dense (Sequential)	(None, 43)	7147	geodata[0][0]
average_10 (Average)	(None, 43)	0	img_vgg16[12][0] geo_dense[12][0]

7

Total params: 134,443,862
Trainable params: 0
Non-trainable params: 134,443,862

[51]: # To train

```

[52]: ensemble_model.compile(optimizer = "adam", loss = 'categorical_crossentropy',  

                           metrics = ['accuracy'])

```

```

[ ]: epochs = 5
history = ensemble_model.fit(
    {"images": X_img_train,
     "geodata": X_geo_train_norm},
    {"img_species": y_img_train,
     "geo_species": y_geo_train},
    batch_size = 10,
    epochs = epochs
)

```