UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

VitiScan: Detección de gotas de producto antifúngico en hojas de vid

Presentado por David Merinero Porres
en Universidad de Burgos — 9 de julio de 2024
Tutor: Carlos Cambra Baseca

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

Expone:

Que el alumno D. David Merinero Porres, con DNI 71312373E, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

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

En Burgos, 9 de julio de 2024

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

Se dispone de una imágenes hiperespectrales de unas hojas de viñedo con diferentes aplicaciones de tratamientos en gotas. Los tratamientos son principalmente de cobre, un tratamiento muy común usado en los viñedos contra cierto tipo de enfermedades. Lo que se pretende es procesar las imágenes y para conseguir saber cuánto porcentaje de hoja se consigue cubrir con la aplicación de estos tratamientos y reducir así el uso de fitosanitarios en los viñedos. Para ello se ha creado una aplicación capaz de procesar estas imágenes hiperespectrales y determinar de esa imagen que superficie corresponde a hoja y cuál a gota. Después de aplicar este procesado, que se podrá realizar tanto de forma individual como por lotes, se podrá visualizar el resultado y compararlo con la imagen original en un editor. Este editor permite visualizar diferentes capas, pudiendo seleccionar cuáles quieres ver y su nivel de transparencia. Las capas que se irán cargando serán normalmente una imagen original de la hoja del viñedo y la imagen procesada de esa misma hoja. Este editor te permite verificar que el resultado del procesamiento ha sido correcto o incorrecto. En caso negativo se volvería a procesar la imagen, pero esta vez cambiando los parámetros según se desee. Tanto las imágenes procesadas como las imágenes creadas en el editor tienen la opción de ser almacenadas. Los datos resultantes del procesado también se almacenan. Las imágenes con las que se trabaja han sido realizadas por investigadores en el marco del proyecto DIG4VITIS.

Descriptores

Imágenes hiperespectrales, Streamlit, Aplicación Web, Procesamiento de Imágenes, Binarización, Python, OpenCV...

Abstract

There are hyperspectral images of vineyard leaves with different applications of antifungal treatments in drops. The treatments are mainly copper and sulfur, a very common treatment used in vineyards against certain types of diseases. The aim is to process the images and to know how much percentage of the leaf can be covered with the application of these treatments and thus reduce the use of phytosanitary products in the vineyards. For this, an application has been created capable of processing these hyperspectral images and determining from that image which surface corresponds to a leaf and which to a drop. After applying this processing, which can be done either individually or in batches, the result can be viewed and compared with the original image in an editor. This editor allows you to view different layers, being able to select which ones you want to see and their level of transparency. The layers that will be loaded will normally be an original image of the vineyard leaf and the processed image of that same leaf. This editor allows you to verify that the processing result has been correct or incorrect. If not, the image would be processed again, but this time changing the parameters as desired. Both processed images and images created in the editor have the option to be stored. The data resulting from processing is also stored. The images we work with have been made by researchers within the framework of the DIG4VITIS project.

Keywords

Hyperspectral images, Streamlit, Web Application, Image Processing, Binarization, Python, OpenCV...

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Se ha creado una aplicación web con tres apartados: Visualizar, Trinarizar y Trinarizar por lotes.

El primero nos permite visualizar una imagen en un editor por capas estilo "Photoshop", donde se permite cargar nuevas capas en diferentes formatos, alternar su visualización, la transparencia de cada capa, eliminar una capa o todas y finalmente poder guardar la imagen resultante.

Por otro lado, la pestaña Trinarizar nos permite subir una imagen en formato hiperespectral, y después se elige la banda para aplicar la primera binarización y detectar la hoja de vid, y la banda y el rango para poder aplicar la segunda binarización para detectar las gotas. Después nos muestra el porcentaje de hoja de vid que hay en la imagen y el porcentaje de gotas que tiene la hoja de vid y nos permite cargar la trinarización en una lista, o exportar la trinarización en formato .png y .csv y los porcentajes en otro .csv.

Finalmente, la pestaña Trinarizar por lotes, que nos permite realizar la trinarización, pero aplicándose a todos las imágenes hiperespectrales que se hayan subido y luego se guardan las imágenes trinarizadas en un .zip.

Ahora se va a pasar a explicar la estructura de la memoria y de los materiales adicionales.

1.1. Estructura de la memoria

1. **Introducción**: Descripción del proyecto y estructura de la documentación.

2. **Objetivos del Proyecto**: Objetivos generales, técnicos y personales que se esperan cumplir en este proyecto.

3. **Conceptos Teóricos**: Explicaciones teóricas sobre los conceptos mas relevantes del proyecto.

4. **Técnicas y Herramientas**: Explicación de las técnicas y herramientas que se han usado a lo largo del desarrollo del proyecto.

5. **Aspectos Relevantes del Desarrollo del Proyecto**: Puntos interesantes e importantes que han ido surgiendo a lo largo del desarrollo del proyecto.

6. **Trabajos Relacionados**: Trabajos relacionados con la temática tratada en este proyecto.

7. **Conclusiones y Líneas de Trabajo Futuras**: Las conclusiones que se han extraído de la realización del proyecto y propuestas e ideas de por donde podría seguirse desarrollando este proyecto.

1.2. Estructura de los anexos

A. **Plan de Proyecto Software**: En este apartado se analiza la planificación temporal y se estudia la viabilidad económica y legal del proyecto.

B. **Especificación de Requisitos**: En esta sección se realiza una descripción del sistema software que se ha desarrollado.

C. **Especificación de Diseño**: En este apartado explica que datos usa la aplicación, su diseño arquitectónico y muestra diferentes diagramas sobre el funcionamiento de la aplicación.

D. **Documentación técnica de programación**: En esta sección se explica la estructura de directorios y archivos que componen el proyecto. También se enseña a crear el entorno de desarrollo que necesita la aplicación en el manual del programador. Además, se detalla paso a paso cómo compilar, instalar y ejecutar la aplicación en diferentes sistemas operativos y utilizando diferentes métodos.

E. **Documentación de Usuario**: En esta sección se explican los requisitos necesarios para ejecutar el proyecto, los detalles de cómo instalarlo, y también explica el funcionamiento del programa.

F. **Anexo de Sostenibilidad Curricular**: En este apartado se realizan unas reflexiones sobre los aspectos de sostenibilidad que se han implementado en el pruyecto.

1.3. Materiales adicionales

Despliegue con Docker

La aplicación aparte de ser ejecutada en modo local haciendo la instalación completa, también se podrá ejecutar dentro de un contenedor Docker.

2. Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

2.1. Objetivos Generales

1. Poder procesar las imágenes hiperespectrales de las hojas de vid con la aplicación en gotas de diferentes productos antifúngicos con base de cobre y azufre, teniendo cada gota una concentración diferente del producto.

2. Poder extraer información de estas imágenes hiperespectrales. Principalmente conocer píxel por píxel si se trata de hoja(01), fondo(00) o gota(10) con producto antifúngico. A esta asignación a cada píxel de uno de los tres valores se ha denominado Trinarización. Después esta información se guarda en un fichero .csv. Lo siguiente es la creación de una imagen pintando cada valor de la trinarización de un color diferente, generándose la imagen trinarizada. Finalmente, se crea otro archivo .csv, este contiene el porcentaje de hoja que hay en la imagen y el porcentaje de gotas con producto que hay en la hoja de vid.

2.2. Objetivos Técnicos

1. Crear un programa con tres pestañas diferentes, una para procesar los datos de una imagen trinarizada que se ha cargado, otra que procesa

las imagenes por lotes y finalmente, otra pestaña que permite visualizar los resultados en editor por capas tipo Photoshop.

2. Permitir subir las imágenes hiperespectrales tanto de forma individual cómo por lotes.

3. Crear un editor que permite seleccionar capas, editarlas(cambiar la transparencia) y guardar el resultado.

4. Seleccionar los datos (bandas espectrales, umbral para aplicar la binarización, etc.) para procesar la imagen hiperespectral, y conseguir la deseada trinarización de los píxeles explicada previamente.

5. Cargar las trinarizaciones en una lista para poder visualizarlas en el editor.

6. Exportar el resultado de la trinarización tanto en formato imagen como en una tabla .csv.

7. Exportar el porcentaje de cubrimiento de las gotas sobre la hoja de vid y el porcentaje de hoja que hay en la imagen en una tabla .csv.

8. Aplicar la trinarización a varias imágenes a la vez, trabajando por lotes, y guardar los resultados en otra carpeta.

2.3. Objetivos Personales

1. Aprender a desarrollar webs en Python y mejorar en el manejo de este lenguaje de programación.

2. Realizar un desarrollo completo de una aplicación desde su inicio hasta su final. Aprender cómo se va evolucionando el código a lo largo del desarrollo.

3. Descubrir nuevas librerías de Python o herramientas que me puedan ser útiles en un futuro.

4. Aprender a documentar correctamente un proyecto y comprender el funcionamiento de LateX y cómo citar las fuentes.

5. Mejorar en la resolución de los errores que van surgiendo mientras que se va desarrollando la aplicación.

3. Conceptos teóricos

La tarea principal que se lleva a cabo en este proyecto es la detección de gotas en hojas de viñedo. Para ello se usan unas imágenes hiperespectrales y se aplican una serie de técnicas. Para comprender correctamente el funcionamiento del proyecto es necesario conocer estos conceptos.

3.1. Imágenes hiperespectrales

Definición

Una imagen hiperespectral [19] en un tipo de imagen digital que recopila una gran cantidad de información en amplio rango del espectro electromagnético. Por ejemplo, las imágenes RGB (rojo, verde, azul) únicamente recogen tres bandas del espectro, mientras que este caso, las imágenes hiperespectrales capturan cientos de ellas, que pueden ir desde el ultravioleta hasta el infrarrojo.

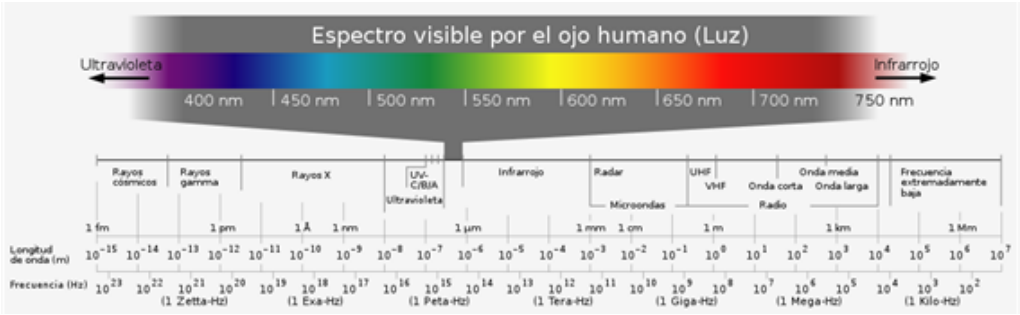

Figura 3.1: Espectro visible por el ojo humano

Obtención de las imágenes hiperespectrales

Las imágenes hiperespectrales se pueden obtener usando sensores especializados, que pueden ser aerotransportados, satelitales o terrestres. Existen diferentes técnicas diferentes para obtener la información de la imagen hiperespectral:

- **Barrido espectral (Spectral Scanning):** También conocido como "pushbroom". En este método, el sensor captura simultáneamente una línea completa de píxeles en una dimensión espacial y en todas las longitudes de onda espectrales, mientras que la plataforma se desplaza en la otra dimensión espacial.
- **Barrido puntual (Point Scanning):** En esta técnica, el sensor captura datos de un solo punto en cada momento y requiere mover el sensor o el objeto de interés para construir la imagen completa. Este método es menos común debido a su lentitud en comparación con otros métodos de escaneo.
- **Barrido en plano focal (Focal Plane Array, FPA):** Utiliza detectores que tienen una matriz de elementos sensibles a diferentes longitudes de onda. Este método permite capturar imágenes hiperespectrales completas en un solo instante, haciendo uso de cámaras hiperespectrales que disponen de un sensor bidimensional.
- **Imágenes instantáneas (Snapshot Imaging):** Este enfoque captura una imagen hiperespectral completa en una sola exposición utilizando dispersión óptica y sensores especiales. Es útil para aplicaciones que requieren alta velocidad de adquisición de datos.

Lo que se obtiene a partir del uso de estas técnicas es una imagen con tres dimensiones, lo que es comúnmente llamado como hipercubo.

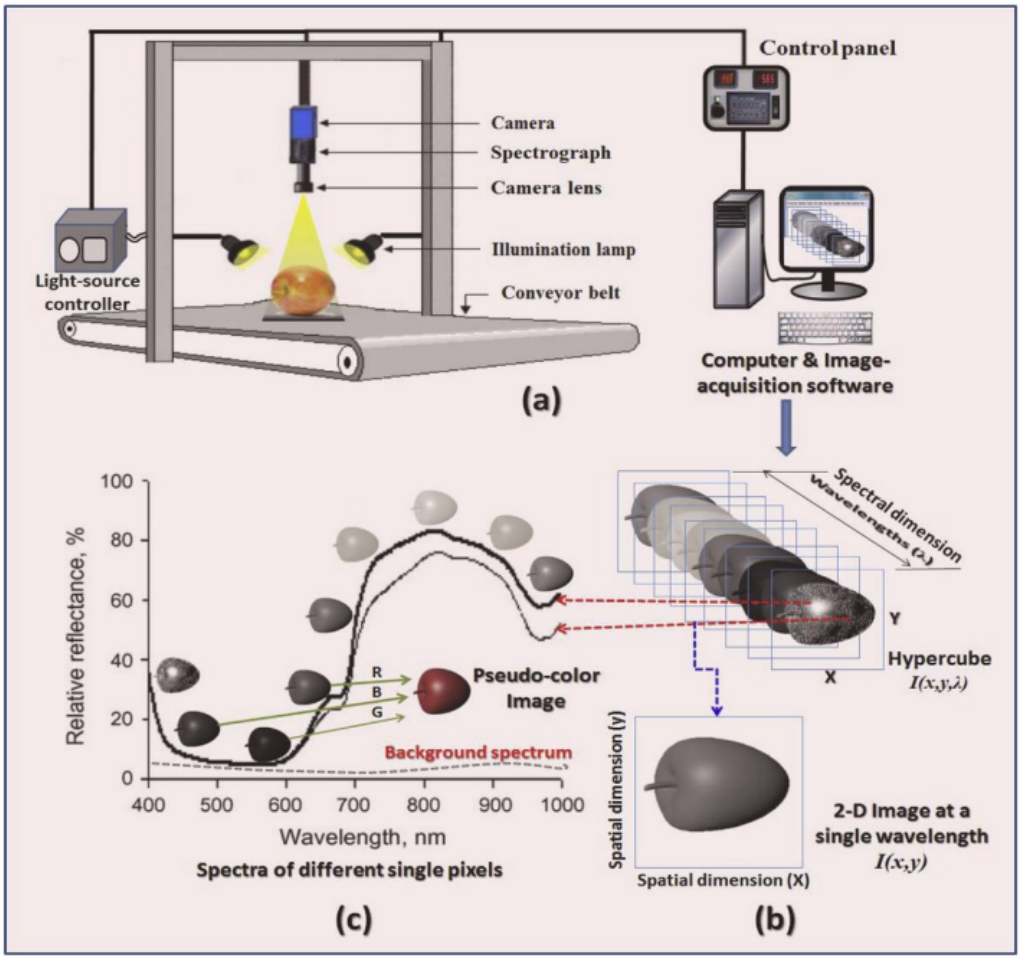

Figura 3.2: (a) Configuración básica de una imagen hiperespectral para adquirir una estructura de datos hiperespectrales "hipercubo". El “hipercubo” podría visualizarse como subimágenes 2D individuales I(x,y) en cualquier longitud de onda determinada (b) o como espectros I(λ) en cualquier píxel determinado de la imagen (c). [16]

Estructura de una imagen hiperespectral

En este proyecto se trabajan con una serie de imágenes hiperespectrales extraídas de un dataset disponible en Riubu. Al tratarse de una imagen hiperespectral con 300 bandas tiene un tamaño considerable, ocupando unos 600 Mb cada una. Estas imágenes vienen divididas en dos ficheros, uno con extensión .bil y otro con extensión .bil.hdr El archivo .bil contiene la información y es el fichero que más espacio ocupa, en cambio, el archivo .bil.hdr es una cabecera con información y ocupa unos pocos KB únicamente.

La información del archivo .bil.hdr incluye características de la imagen hiperespectral como son las dimensiones (líneas, muestras, bandas), detalles de la captura (tipo de dato, intervalo espectral, velocidad de obturación), características del sensor (número de serie, tamaño de píxel), ajustes de calibración radiométrica y detalles espectrales (longitudes de onda y unidades).

En este proyecto se ha trabajado con imágenes hiperespectrales con una resolución de 1200x900x300, lo que viene siendo un hipercubo. Las 300 bandas espectrales van desde una longitud de onda desde 388 hasta 1024, aumentando 2 ese valor en cada nueva banda aproximadamente. En el proyecto software se da la opción de elegir una banda desde 0 hasta 299, pero el valor de longitud de onda de cada una en realidad se corresponde a lo anteriormente descrito.

Análisis de las imágenes hiperespectrales

El análisis de imágenes hiperespectrales implica varios pasos para extraer y procesar la información contenida en el hipercubo de datos. Una de las primeras tareas es la preprocesamiento de los datos, que puede incluir corrección geométrica de la imagen, eliminación de ruido y normalización de los espectros. Después de esta etapa, se pueden usar diferentes técnicas para interpretar y clasificar los datos. Aquí se citan una serie de posibles técnicas a aplicar:

- **Clasificación espectral:** Identificación y categorización de materiales en la imagen basada en sus valores espectrales.
- **Análisis de componentes principales (PCA):** Reducción del número de dimensiones de los datos para resaltar las características más significativas.
- **Análisis de mezcla espectral:** Descomposición de los píxeles en función de sus bandas espectrales y tipo de material.
- **Algoritmos de aprendizaje automático:** Aplicación de técnicas de aprendizaje supervisado y no supervisado para tareas de clasificación y segmentación de los datos.

Aplicaciones de las imágenes hiperespectrales

Las imágenes hiperespectrales se usan en una gran variedad de campos [1] y en tareas diferentes que se detallan a continuación:

- **Agricultura:** Monitorización de la salud de los cultivos, detección de plagas y enfermedades, y evaluación del contenido de nutrientes en el suelo.
- **Medio ambiente:** Evaluación de la calidad del agua, monitorización de la deforestación y gestión de desastres naturales.
- **Geología:** Identificación y mapeo de minerales y recursos naturales.
- **Defensa y seguridad:** Detección de camuflaje y materiales peligrosos, vigilancia y reconocimiento.
- **Medicina:** Detección de tumores y diagnóstico de enfermedades a través de la imagen médica no invasiva.
- **Industria alimentaria:** Inspección de calidad y seguridad de los alimentos, detección de contaminantes y autenticación de productos.

3.2. Binarización

Para poder discriminar entre lo que es hoja, gota o fondo de la imagen es necesario el uso de la binarización. La binarización se aplica sobre imágenes en escala de grises, que justamente así es como procesamos las bandas de la imagen hiperespectral, al tratarse de valores decimales que en este caso van del 0 al 1.

La binarización consiste en elegir un valor umbral [18] a partir del cuál se convierte la imagen de escala de grises en una imagen binaria o máscara. Los valores superiores al umbral se convierten en 1 y los inferiores en 0, de esta forma consiguiendo así detectar el objeto del fondo, en el caso de que se diferencien claramente. Por suerte, se está trabajando con imágenes de laboratorio sobre un fondo blanco, lo cual facilita más la detección entre la hoja y el fondo. Aunque en este proyecto, hemos utilizado la media de los valores como umbral, para que funcione correctamente con fotos con diferente nivel de luminosidad.

Figura 3.3: Ejemplo de binarización de una manzana.

Posteriormente se realiza una segunda binarización, entre las gotas y la hoja, lo cual es más complejo de discriminar. Por ello, se va a utilizar un rango de valores en vez de un valor umbral a la hora de binarizar entre la hoja y las gotas.

La combinación de estas dos binarizaciones es lo que se ha denominado trinarización.

3.3. Normalización

Para poder trabajar con las bandas de la imagen hiperespectral como imágenes es necesario pasarlas al formato RGB, donde los valores que hay van del 0 al 255. Sin embargo, los datos que se procesan de cada capa de la imagen hiperespectral son valores con decimales que van del 0 al 1.

$$y = \frac{(x - \text{mín}(x)) \cdot (valor\_max - valor\_min)}{\text{máx}(x) - \text{mín}(x)} + valor\_min$$

Donde:

- **x** es el valor de entrada (Número decimal del 0 al 1).
- **min(x)** es el valor mínimo de los valores de entrada.
- **max(x)** es el valor máximo de los valores de entrada.
- **valor_min** es el valor mínimo del rango de salida (0).

- **valor_max** es el valor máximo del rango de salida (255).
- **y** es el valor ya normalizado.

Por eso mismo es necesario aplicar una normalización, es decir, transformar los valores decimales en enteros que vayan de 0 al 255 pero de forma proporcional para no perder información. De esta forma somos capaces de visualizar estas bandas en formato de imagen estándar, en este caso PNG.

Todo esto se realiza en el proyecto usando esta línea de código:

```
imagen = cv2.normalize(banda, None, 0, 255,
cv2.NORM_MINMAX, dtype=cv2.CV_8U)
```

Lo que se hace aquí, es aplicar la normalización minmax [23] de la librería OpenCV, que utiliza la fórmula anteriormente descrita. Además cada valor obtenido se transforma al formato de píxel sin signo de 8 bits (unsigned char), que almacena valores del 0 al 255. Esto también se hace usando la librería OpenCV.

3.4. Técnicas de transformación de imágenes

OpenCV es una biblioteca de código abierto ampliamente utilizada para el procesamiento de imágenes y visión por computadora. Entre las muchas técnicas que ofrece, las transformaciones morfológicas [17] destacan por su utilidad en la modificación y análisis de estructuras en imágenes binarias, un tipo de imágenes que utilizamos en este proyecto en el proceso de trinarización. Las técnicas más útiles y utilizadas son: erosionar, dilatar, abrir y cerrar. Luego hay más variantes como la función morphologyEx(), que es capaz de combinar algunas de las anteriores para realizar transformaciones morfológicas.

Figura 3.4: Imagen binaria a la que se van a aplicar transformaciones.

Ahora se va a pasar a explicar las técnicas que se han utilizado en el proyecto para procesar las imágenes hiperespectrales, y así poder filtrar por ejemplo, el ruido que se genera al binarizar las gotas de producto fúngico de las hojas de vid. Estas son:

Erosión (Erode)

La erosión reduce el tamaño de los objetos del primer plano, eliminando los píxeles de los bordes. Esto se logra haciendo que un píxel se considere 1 solo si todos los píxeles debajo del kernel son 1.

Figura 3.5: Imagen binaria después de aplicarle una erosión.

Esta transformación se utiliza en el proyecto para reducir el tamaño de las gotas de producto detectadas en las hojas de vid y reducir así el ruido.

Dilatación (Dilate)

La dilatación es opuesta a la erosión, realizando la acción de aumentar el tamaño de los objetos.

Figura 3.6: Imagen binaria después de aplicarle una dilatación.

En este proyecto se han dilatado las gotas para recuperar el tamaño original después de aplicar la erosión.

Cierre (MORPH_CLOSE)

El cierre es una transformación que consiste en la dilatación seguida de erosión. Esta operación es útil para cerrar pequeños agujeros dentro de los objetos, es decir, rellenarlos completamente. Es una opción de las varias que ofrece la función morphologyEx().

Figura 3.7: El antes y el después de aplicar el morphologyEx con la opción de cerrar.

En este proyecto se ha usado la transformación morfológica de cierre para conseguir rellenar las gotas de producto antifúngico, ya que el agua reflejaba y algunas gotas no se conseguían cerrar del todo.

4. Técnicas y herramientas

En este apartado se detallan las técnicas y herramientas utilizadas a lo largo del desarrollo del proyecto. Para cada una de ellas se realizará un pequeño resumen de sus características y los motivos de por qué se han elegido en cada caso respecto a sus alternativas.

4.1. Aplicaciones y Servicios

Zube

Zube [14] es una plataforma de gestión de proyectos que combina un tablero Kanban y los Sprints de la metodología Scrum, proporcionando una solución integral para equipos que buscan administrar sus proyectos de manera ágil. Se ha elegido sobre ZenHub, siendo más famoso y su principal competidor porque ahora es de pago. Las funciones que ofrece Zube respecto a ZenHub son similares, así no ha supuesto una gran diferencia. Zube además te permite la opción de conectarte con GitHub, pudiendo crear y administrar los milestones e issues directamente desde su plataforma. Además, al poseer un tablero Kanban y hacer uso de Sprints es mucho más visual y fácil de administrar el proyecto que si se hiciera directamente desde GitHub.

GitHub

GitHub [4] es una plataforma que permite alojar y gestionar repositorios de código utilizando el sistema de control de versiones Git. Se ha elegido por ser la opción más conocida y ampliamente usada. Para realizar los diferentes commits en nuestro repositorio se ha usado el cliente de git, utilizando la

opción de consola y usando los comandos. El uso de GitHub o similares hoy en día es algo esencial a la hora de realizar cualquier desarrollo de software.

Visual Studio Code

Visual Studio Code (VSCode) [13] es un editor de código fuente desarrollado por Microsoft. Es bastante potente, ligero y se puede personalizar y mejorar con las diferentes extensiones que va creando la comunidad. Te permite usar una gran variedad de lenguajes de programación, depurar el código directamente desde el editor y además tiene integración directa con GitHub. Aunque en nuestro caso no se ha usado esta opción, ya que como se ha comentado previamente se ha usado el cliente de Git para realizar los commits. Se ha elegido esta opción respecto a otras como Pycharm o Jupyter Notebook debido a la familarización con el programa, ya que lo he usado en algunas asignaturas de la carrera y me ha gustado como funciona. Además, al ser uno de los editores de código más usados si surgiera algún problema me resultaría más fácil de resolver, ya que otro usuario podría haberlo resuelto previamente.

Overleaf

Overleaf [7] es una plataforma de escritura y colaboración en línea para crear documentos LaTeX. Es bastante conocido en la comunidad académica, ya que bastante gente lo usa para editar sus documentos científicos o de investigación. Algunas características interesantes por lo que se ha elegido esta plataforma de edición de documentos LaTeX es la capacidad de compartir el proyecto con el tutor y que pueda ir revisando la documentación a lo largo de su desarrollo. Otra característica muy útil es la posibilidad de ir compilando el código para poder ir viendo como va quedando el documento. Pero no todo son cosas buenas, ya que la opción de enlazarlo con GitHub que poseía previamente se ha vuelto de pago, por lo que me he visto obligado a descargar el proyecto y subir al repositorio haciendo un commit con el cliente Git de forma local. A pesar de esta desventaja seguiría recomendando su uso, y si se usa con mucha asiduidad, incluso plantearse pagar la versión premium.

ChatGPT

El famoso ChatGPT [2] es un modelo de inteligencia artificial desarrollado por la empresa OpenAI. Su principal función es la de generar texto a partir de una orden que tú le mandes. El uso que se le ha dado en la realización

de este proyecto ha sido la de resolver alguna duda que iba surgiendo a lo largo del desarrollo y para poder detectar algún error de programación.

Es una herramienta muy útil para tareas sencillas y te ahorra el tiempo de tener que navegar por las páginas para buscar cierta información, ya que al estar entrenado con millones de datos esas cuestiones sencillas te las resuelve sin problema. Sin embargo, para un trabajo más preciso o más complejo no es recomendable su uso, ya que muchas veces se inventa la información con tal de satisfacer tu petición, aparte de que no te referencia el origen de donde saca esa información.

Docker

Docker [3] es un software que automatiza el proceso de desplegar aplicaciones, creando un contenedor donde se incluyen las dependencias y el contenido de la aplicación, y gracias a esto se puede ejecutar en cualquier servidor Linux.

En este proyecto se ha utilizado Docker para el despliegue de la aplicación. Gracias a estos contenedores, el usuario que quiere ejecutar la aplicación se ahorra todo el trabajo de bajarse el proyecto del repositorio, instalar las dependencias y demás tareas. Me parece una solución ideal para poder ejecutar diferentes aplicaciones de forma fácil y rápida en entornos que utilizan Linux.

4.2. Lenguajes de programación

Python

Python [21] es un lenguaje programación de alto nivel, interpretado y de propósito general. Se elegido en este caso debido a las librerías que posee y el tipo de proyecto que se iba a desarrollar, ya que es común utilizar Python para el desarrollo de aplicaciones web. Además, al haberlo utilizado en varias asignaturas a lo largo de la carrera me pareció una buena elección.

CSS

CSS(Cascading Style Sheets) [20] es un lenguaje de diseño gráfico que se usa comúnmente junto al lenguaje HTML, y sirve para dar estilo y formato a las páginas web, documentos o interfaces que se creen con un lenguaje de estilo marcado. Su uso en este proyecto ha sido algo anecdótico, ya que

sólo se ha usado para modificar el aspecto de algunos elementos que vienen integrados en la librería *Streamlit*.

XML

XML(Extensible Markup Language) [22] es un lenguaje marcado que define reglas para codificar un documento y que sea legible tanto por humanos como por máquinas. Para lograr todo esto se describe la estructura del contenido usando etiquetas. En este proyecto se utiliza XML para cargar los datos de las bandas de las imágenes hiperespectrales a trinarizar, es decir, estos datos se cargan desde un fichero XML.

4.3. Librerías

Streamlit

Streamlit [11] es un framework de Python que permite la creación de aplicaciones web interactivas para la visualización de datos de forma rápida y sencilla, aunque se pueden realizar tareas más complejas en ella. Gracias a esta herramienta he sido capaz de crear una aplicación interactiva y con un buen diseño. Para poder usar este framework es necesario importar su librería en el proyecto.

Spectral

Spectral Python (SPy) [10] es una biblioteca de Python diseñada para trabajar con datos de imágenes hiperespectrales. Permite la lectura, el análisis y la visualización de estos datos. Las imágenes hiperespectrales contienen información detallada a través de un amplio rango del espectro electromagnético, lo que las hace útiles en aplicaciones que van desde la teledetección y la agricultura de precisión hasta la detección de anomalías y la clasificación de materiales. SPy soporta varios formatos de archivo hiperespectral y proporciona herramientas para manipular y analizar estos datos de manera eficiente.

OpenCV

OpenCV (Open Source Computer Vision Library) [6] es una biblioteca de software de código abierto enfocada en la visión artificial y el aprendizaje automático. Incluye una amplia variedad de funciones para el procesamiento

de imágenes y videos. OpenCV es ampliamente utilizado en aplicaciones como el reconocimiento facial, la detección de objetos, la realidad aumentada y el análisis de movimiento. En este proyecto su uso principal ha sido el procesamiento de imágenes con el uso de técnicas como la binarización, la normalización de valores, erosionar y dilatar objetos o rellenar espacios cerrados en las diferentes máscaras usadas.

Numpy

Numpy [5] es una de las librerías más usadas y conocidas de Python. Se usa principalmente para el trabajar con datos numéricos y permite trabajar con grandes volúmenes de datos. Permite crear arrays, usar funciones matemáticas y lógicas, generar números y un largo etcétera de funcionalidades más. En este proyecto se ha usado para cambiar el formato de los datos, crear arrays de llenos de ceros, comparar datos o leer datos del buffer. Es una librería muy útil y polivalente, de ahí que sea tan común su uso.

Pandas

Pandas [8] es otra de las librerías más utilizadas de Python. Esta librería proporciona estructuras de datos como los conocidos dataframes o series de datos, permite leer,escribir o modificar datos, etc. Normalmente los datos trabajados con Pandas se analizan o visualizan con otras librerías gráficas. En este proyecto se utiliza el dataframe para almacenar los porcentajes calculados durante la trinarización y que posteriormente se guardan en un archivo .csv.

SKimage

La librería SKimage o Scikit-Image [9] ofrece una colección de algoritmos para poder procesar imágenes. En este proyecto se ha utilizado para eliminar ruido a la hora de detectar las gotas, midiendo y eliminando aquellas regiones que fueran superiores o inferiores a unos valores fijados.

streamlit_image_zoom

La librería streamlit_image_zoom [12] se ha usado en el proyecto para añadir la funcionalidad de hacer zoom a la imagen resultante en la pestaña Visualizar de la aplicación. Principalmente, te permite hacer zoom con la rueda del ratón y también tiene la opción de desplazarte por la imagen usando el cursor.

5. Aspectos relevantes del desarrollo del proyecto

5.1. Primeros pasos del proyecto

Como en todo proyecto, el resultado final puede variar mucho respecto a las primeras ideas que se van teniendo. En este proyecto pasó algo por el estilo. Se partió de la idea de utilizar Streamlit como base para desarrollar una aplicación web y se pensó en la librería *OpenCV* para el procesado de imágenes. Siempre se tuvo en mente las imágenes hiperespectrales, pero se comenzó a trabajar con unas imágenes de ciertas bandas de estas donde se apreciaban mejor las gotas. Estas imágenes tenían aplicados unos valores en los canales RGB que potenciaban aún más el contraste entre las gotas y la hoja. Entonces se empezó el desarrollo de la aplicación con estas imágenes en formato .tiff.

Durante las primeras semanas se desarrolló un prototipo, que me ayudó a familiarizarme con las herramientas que ofrecen tanto OpenCV como Streamlit. En esta primera aproximación se empezó a trabajar con las binarizaciones para detectar la hoja, la búsqueda de bordes, aplicar mapas de color, etc.

Todo esto se aplicaba sobre una imagen en formato .tiff que se subía a la aplicación y se le aplicaban las diferentes transformaciones una tras otra pudiendo ver el resultado que se obtenía en cada caso. Gracias a este trabajo tan visual con cada cambio que hacía podía ir viendo los cambios que sucedían en la imagen y poder decidir que técnica o herramienta me ayudaba más a llegar al objetivo. Tras llegar al punto de conseguir eliminar el fondo de la imagen y permitir cambiar los colores de la imagen con unas

bandas que permitían cambiar los canales RGB se decidió hacer un cambio radical al proyecto.

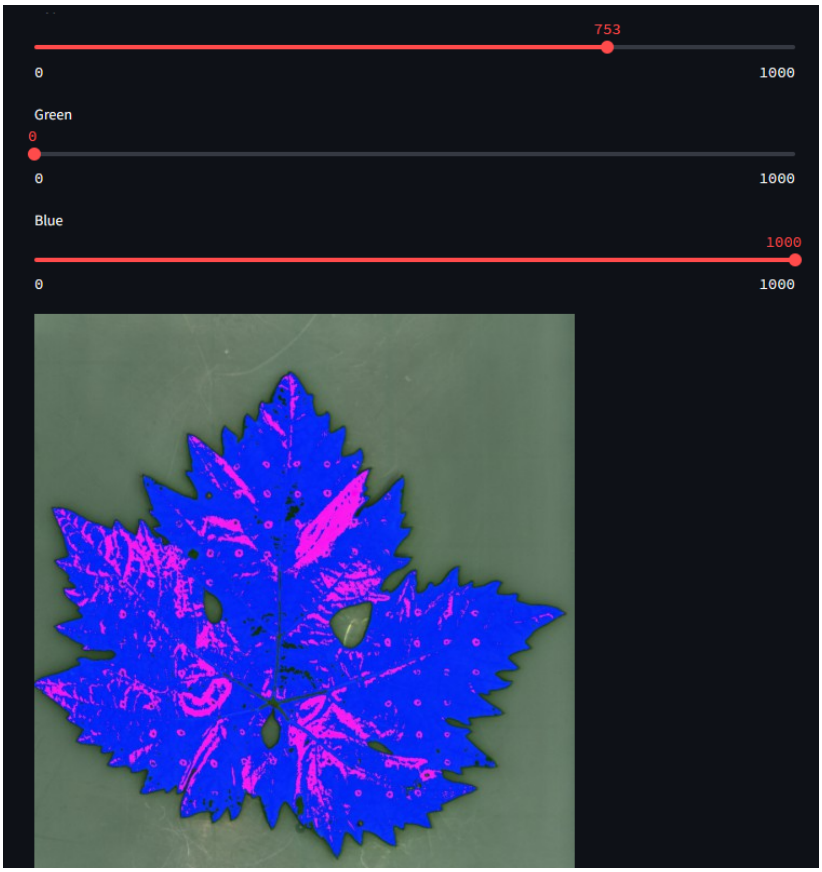

Figura 5.1: Captura del prototipo.

Se celebró una reunión a principios de Mayo donde se llegó a la conclusión que era mejor trabajar directamente con las imágenes hiperespectrales, poder extraer una banda, y tras los procesamientos de trinarizar ser capaces de determinar qué era hoja, que era gota y qué era fondo. También se decidió crear dos pestañas diferentes, una para realizar la trinarización y otra para visualizar los resultados en una especie de editor por capas tipo "Photoshop"que permitiera comparar el resultado obtenido respecto a la foto original cambiando la transparencia.

Esto supuso un antes y un después en el desarrollo de esta aplicación, porque ya se estaba hablando de trabajar con imágenes hiperespectrales, procesarlas y guardar sus resultados, y además, de crear un editor para visualizar los resultados. Una vez que se especificaron bien el diseño y

diferentes funcionalidades del proyecto ya se pudo empezar a trabajar de forma más firme y segura.

5.2. Metodología Scrum

Para la realización del proyecto se ha seguido la metodolgía Scrum, un modelo de gestión de proyectos ágil. Con esta metodología se van realizando sucesivos Sprints, tras los cuales se hace una reunión donde se valora lo que se ha hecho hasta la fecha y se fijan nuevos objetivos a realizar. La duración de estos Sprints ha variado en función de la tarea a realizar o la disponibilidad tiempo en la agenda. Estos Sprints han tenido una duración desde una semana hasta unos 15 días.

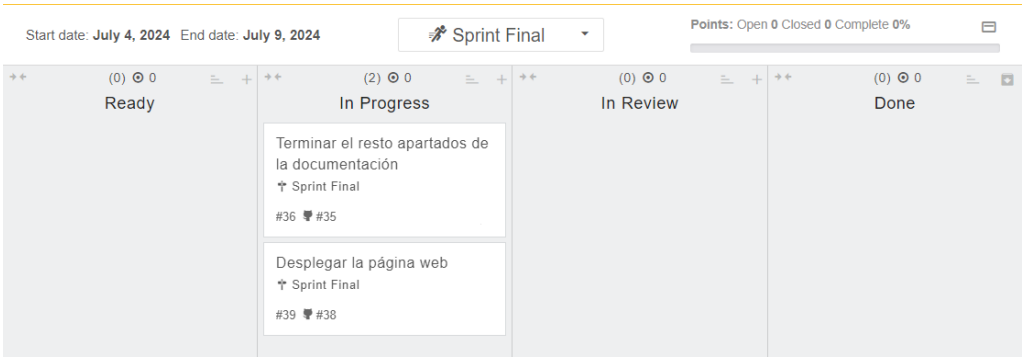

Figura 5.2: Tablero del Sprint Final con las issues pendientes.

Para la planificación de estos Sprints se ha usado el servicio Zube, dónde se han ido creando estos Sprints con un milestone de GitHub asociado y con la creación de las issues para cada uno de ellos.

Adicionalmente se ha usado un tablero Kanban en el día a día, donde se podía ver las tareas pendientes, en desarrollo o finalizadas de ese Sprint. Esta funcionalidad también viene incluida en Zube.

5.3. Estructura del proyecto con *Hydralit*

Una vez que se planteó una aplicación con diferentes pestañas, empecé a barajar las opciones que tenía para implementarlo. Finalmente, opté por la librería *Hydralit*, que te permite implementar una barra de navegación con diferentes pestañas y está integrado con Streamlit. Esta librería añade

además algunas funcionalidades extras como barras de progreso, animaciones de carga, widgets de información, etc.

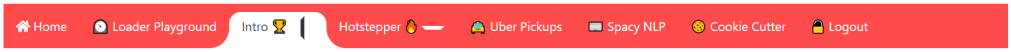

Figura 5.3: Ejemplo de barra de navegación de Hydralit.

Cada pestaña de la barra de navegación es, de facto, una aplicación diferente, por lo que en este proyecto contamos con tres aplicaciones diferentes dentro de la principal. Esto supone tener que guardar algunos datos y estados para que se guarden al cambiar entre una pestaña y otra, ya que sino se eliminarían al ejecutar otra aplicación. Para ello, se han utilizado unos estados de sesión (`st.session_state`), es decir, unas variables que almacenan la información mientras se ejecuta la aplicación principal. Luego está la opción de exportar los resultados, que te guarda los resultados de forma local.

5.4. Procesado de imágenes hiperespectrales

Para poder realizar la trinarización correctamente se han debido realizar varias transformaciones y procesados a las imágenes hiperespectrales. Para empezar como hemos dicho previamente se debe extraer una banda de la imagen hiperespectral para trabajar con ella. Esta banda la debe elegir el usuario, tanto la de detectar la hoja, como la de detectar la gota. Una vez elegidas las bandas se procede a la binarización de esas imágenes. Ya en este paso se debe realizar una normalización para pasar los valores decimales de la banda a valores del 0 al 255, para ser capaces de visualizarlos. Esa dos binarizaciones no dejan de ser máscaras de 1 y 0, por lo que se comparan y se añaden los colores en función del valor que tengan ambas máscaras píxel por píxel.

```
banda = img.read_band(paso1)
banda = cv2.normalize(banda, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)
_, hoja_bin = cv2.threshold(banda, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

banda2 = img.read_band(paso2)
banda2 = cv2.normalize(banda2, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)
gotas_bin = cv2.inRange(banda2, int(paso2_min * 255/1000), int(paso2_max * 255/1000))


trinarizada = np.zeros((banda.shape[0], banda.shape[1], 3), dtype=np.uint8)

trinarizada[(hoja_bin == 0) & (gotas_bin == 0)] = [255, 0, 0]
trinarizada[(hoja_bin == 0) & (gotas_bin == 255)] = [0, 255, 0]
trinarizada[hoja_bin == 255] = [0, 0, 0]
```

Figura 5.4: Fragmento de código encargado de la trinarización.

Después de tener la imagen trinarizada con los colores negro(fondo), verde(hoja) y rojo(gotas), se procede a eliminar el ruido que hay en la imagen. Para ello, primero se aplican unas transformaciones de dilatación, rellenado y reducción. Con esto se consigue rellenar las gotas que no estuvieran cerradas.

```
mascara_gotas = (trinarizada[:, :, 0] == 255) & (trinarizada[:, :, 1] == 0) & (trinarizada[:, :, 2] == 0)

mascara_dilatada = cv2.dilate(mascara_gotas.astype(np.uint8), np.ones((5, 5), np.uint8), iterations=1)

mascara_rellenada = cv2.morphologyEx(mascara_dilatada, cv2.MORPH_CLOSE, np.ones((5, 5), np.uint8))

mascara_final = cv2.erode(mascara_rellenada, np.ones((3, 3), np.uint8), iterations=1)
```

Figura 5.5: Fragmento de código encargado de las transformaciones morfológicas.

La otra transformación que se hace es eliminar aquellas regiones de gota que sean más pequeñas o más grandes de un valor dado, ya que conocemos el tamaño de las gotas.

```
etiquetas = measure.label(mascara_final, connectivity=2)
propiedades = measure.regionprops(etiquetas)

area_min_umbral = 100
area_max_umbral = 500

trinarizada[(mascara_final == 1) & ~mascara_gotas] = [255, 0, 0]

for prop in propiedades:
    if prop.area > area_max_umbral or prop.area < area_min_umbral:
        for coord in prop.coords:
            trinarizada[coord[0], coord[1]] = [0, 255, 0]
```

Figura 5.6: Fragmento de código encargado de eliminar regiones fuera del rango.

Gracias a todas estas transformaciones se obtiene una imagen trinarizada bastante precisa, al haber escogido las bandas donde mejor se ven las gotas y haber eliminado el ruido de la hoja, ya que al ser irregular es más difícil de discriminar entre gota u hoja.

5.5. Ficheros resultantes

Este proyecto permite exportar los resultados al disco local, para poder utilizar posteriormente los datos que se han conseguido extraer de las imágenes hiperespectrales.

Figura 5.7: Pestañas disponibles en la aplicación.

Pasamos a ver pestaña por pestaña qué datos o imágenes se pueden exportar:

Pestaña *Visualizar*

En la pestaña *Visualizar*, tras elegir las diferentes capas y sus transparencias, se da la opción de guardar la imagen resultante. Esta imagen tiene la

misma resolución que las imágenes originales que se han subido y se guarda en formato .png.

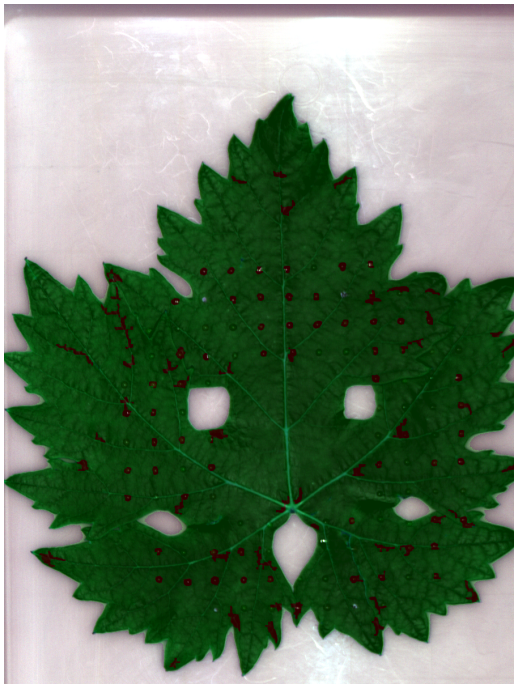

Figura 5.8: Ejemplo de imagen resultante tras la edición por capas, tras fusionar una imagen a color de la hoja con una imagen trinarizada de la misma.

Pestaña *Trinarizar*

En la pestaña *Trinarizar*, tras seleccionar las bandas y el rango se obtiene una imagen con tres colores diferentes. Esta imagen representa la trinarización, con el fondo de negro, la hoja de verde y las gotas que se han detectado en la hoja, de rojo. Otra información se obtiene tras realizar la trinarización son los píxeles de hoja y los píxeles de gota que hay. También se muestran unos porcentajes, el del total de hoja respecto a la imagen y el del total de gotas respecto a la hoja.

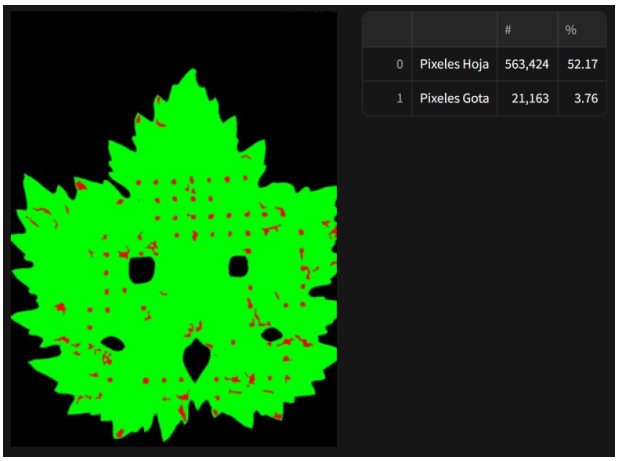

Figura 5.9: Captura del resultado de aplicar la trinarización, se observa la imagen trinarizada y los porcentajes.

Finalmente se da la opción de guardarlos resultados, que son: la imagen trinarizada en formato .png, el archivo .csv con el número de píxeles y los porcentajes y otro archivo .csv que almacena la información de cada píxel de forma codificada. La codificación de la trinarización es esta: 00 corresponde a fondo, 01 corresponde a hoja y 10 corresponde a gota.

Figura 5.10: Captura con algunos de los valores del .csv correspondiente a la trinarización de los píxeles.

Pestaña *Trinarizar Por Lotes*

Por último, en esta pestaña se trinarizan todas las imágenes que hay en un directorio. Para ello se selecciona una carpeta de origen y una de destino donde guardar los ficheros resultantes. En este caso, se ha optado por sólamente guardar la imagen trinarizada en formato .png de cada imagen hiperespectral en la carpeta destino, sin crear ningún archivo .csv como en el paso anterior. Este modo nos permite aplicar la trinarización de forma más rápida y posteriormente poder visualizar los resultados para cada hoja en la pestaña *Visualizar*.

Figura 5.11: Captura del resultado de aplicar la trinarización por lotes, se aprecian varias imágenes trinarizadas en una carpeta.

6. Trabajos relacionados

En este apartado se presentan algunos trabajos y tesis que utilizan el mismo tipo de datos que en este proyecto (imágenes hiperespectrales) y comparten la idea de detectar diferentes características para después poder clasificarlas. Además todos los trabajos y estudios seleccionados tienen relación con la botánica.

6.1. Remote sensing of vegetation and crops using hyperspectral imagery and unsupervised learning methods.

El objetivo que busca este proyecto es la detección de cultivos y vegetación a partir de unas imágenes hiperespectrales utilizando métodos de entrenamiento tanto supervisados como no.

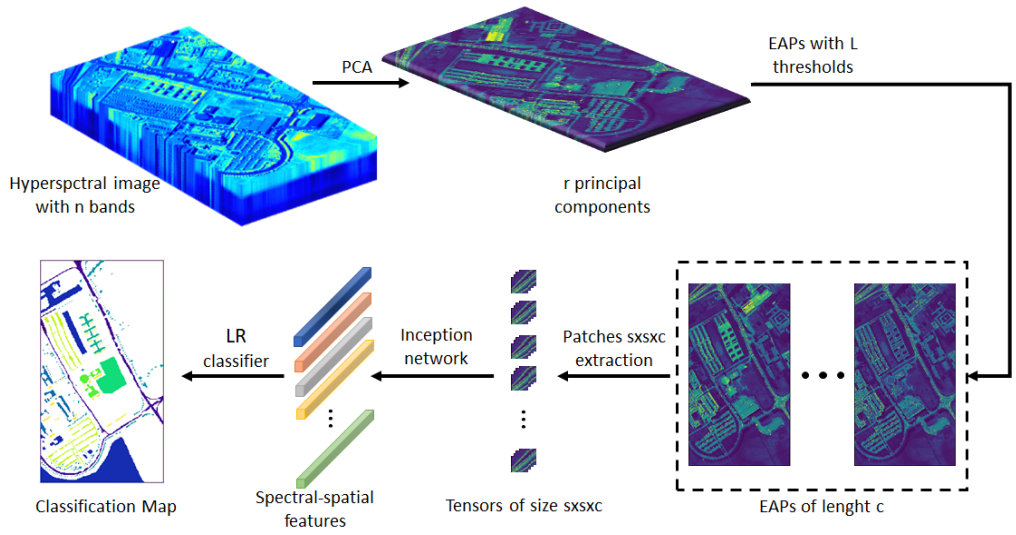

Figura 6.1: Diagrama del proceso de reconocimiento y clasificación basado en EAP y una red tipo inception.

Como se puede observar en la figura 6.1 esta aplicación tiene varias similitudes con la que se ha desarrollado en este proyecto. Para empezar, se utilizan umbrales (thresholds) para seleccionar las diferentes bandas espectrales. Otro aspecto que tiene este trabajo en común con mi proyecto es la clasificación final en una imagen, dónde se colorea de forma diferente cada tipo de zona detectada en la imagen original. La diferencia es que este trabajo utiliza adicionalmente diferentes redes y modelos no supervisados a la hora de detectar las diferentes características.

Se puede acceder al repositorio del proyecto a través del siguiente enlace: https://github.com/davidruizhidalgo/unsupervisedRemoteSensing

6.2. Hyperspectral image applied to determine quality parameters in leafy vegetables.

Esta tesis doctoral [15] utiliza las imágenes hiperespectrales para otra labor diferente, determinar la calidad de vegetales de hoja y su nivel de frescura.

La primera tarea que se llevó a cabo fue la de tomar diferentes fotografías hiperespectrales de hojas de espinacas, berros y lechugas a lo largo de 21 días, mientras estaban conservadas a 4ºC en un ambiente refrigerado.

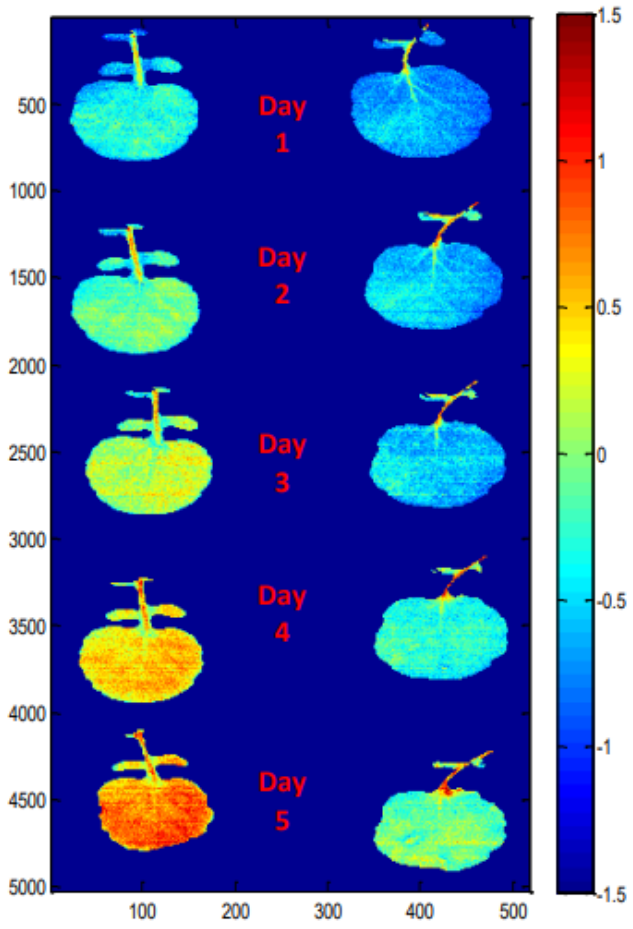

Figura 6.2: Imagen con la puntuación de dos hojas de berros a lo largo de 5 días de pruebas.

La segunda tarea fue procesar esas imágenes, en función de la reflectancia y otros parámetros, y clasificar el nivel de frescura de esos alimentos. Para esa clasificación se usaron diferentes modelos. Finalmente, se pudo concluir que este método tiene mucho potencial, ya que permite clasificar la calidad de los alimentos frescos sin destruirlos ni sacarlos de su envase.

6.3. PlantCV: Plant phenotyping using computer vision

Por último, vamos a explicar PlantCV, una librería de Python basada en OpenCV, capaz de detectar el fenotipo de la plantas. Este software de análisis de imágenes utiliza diferentes técnicas a partir de una gran variedad de algoritmos y paquetes. Al poseer una arquitectura modular, se pueden añadir nuevos métodos de forma rápida y sencilla.

Esta librería tiene relación con el proyecto sobretodo a la hora de detección de hojas, ya que es su principal funcionalidad. Sin embargo, en este proyecto se detecta el contorno de la hoja únicamente para detectar las gotas posteriormente, no porque la forma de la hoja sea algo relevante a la hora de extraer la información que nos interesa.

Se puede acceder al repositorio de la librería a través del siguiente enlace: https://github.com/danforthcenter/plantcv

7. Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

Tras la realización del proyecto se puede concluir que se han cumplido gran parte de los objetivos marcados al inicio del mismo. La aplicación es capaz de detectar las hojas, las gotas y guardar los datos e imágenes con los resultados. Todo esto se puede hacer tanto de forma individual como por lotes. Además se pueden visualizar los resultados en el editor y comprobar la efectividad de las trinarizaciones, pudiendo también exportar los resultados. Para llegar este resultado han ido surgiendo algunos problemas que se han ido resolviendo a lo largo del desarrollo y que se van a exponer en los siguientes apartados.

Sobre la superposición de capas en el editor

Un problema que no se ha podido solventar de manera completamente eficaz es el de superponer las diferentes capas en el editor. La idea original como ya se ha comentado, era la de implementar un editor de capas que funcionase tal y cómo lo hace Photoshop, pero no se ha podido desarrollar de forma completa.

Debido a la complejidad de la tarea y a que no se ha conseguido encontrar una librería en Python que implemente esas funciones, la única salida que se ha visto es la utilizar una funcionalidad de OpenCV para conseguir un resultado parecido.

La funcionalidad de OpenCV que se ha usado es la de addWeighted. Lo que hace esta función es fusionar dos imágenes, pero con diferentes transparencias en función del valor que se escoja. Ese valor es el que se modifica en el Slider del editor de imágenes del programa.

Por lo que no se ha conseguido trabajar con capas superpuestas de forma independiente, sino que se ha realizado una fusión de imágenes pero eligiendo el peso de cada una.

Pero en un intento de asemejarnos a la funcionalidad de Photoshop, se tomó la decisión de que capa inferior siempre feuse opaca, es decir, sin ninguna transparencia, para así implementar algo más o menos parecido.

Sobre la exportación de datos

Durante el desarrollo del proyecto, no se consideró que durante el despliegue de la aplicación, no sería posible acceder a los directorios del usuario final. Esto impediría guardar los archivos de forma local en las carpetas elegidas por la aplicación.

Este problema surgió a la hora de desplegar la aplicación como es evidente, y se ha tomado la decisión de que los ficheros resultantes se guarden en la carpeta de descargas que tenga asignada el usuario en su navegador.

No es una solución ideal, ya que cuando se ejecutaba en modo local, al guardarse en diferentes carpetas se conseguía tener los ficheros mejor ordenados. Para solventar este nuevo inconveniente, se ha modificado el nombre de los ficheros resultantes poniendo unas iniciales diferentes para cada tipo de fichero a exportar, facilitando así la localización de cada tipo de fichero exportado.

Sobre la carga de imágenes hiperespectrales por lotes

Otro gran problema que nos hemos encontrado ha sido la carga de ficheros por lotes, ya que como se ha comentado, en las aplicaciones web modernas no se deja acceder directamente a los directorios del usuario por motivos de seguridad.

La solución que se ha encontrado ha sido la de abrir la carpeta que se quiera trinarizar por lotes y arrastrarla al widget de subida de ficheros, consiguiendo así que se suban todos los archivos de una vez.

El mayor incoveniente de esto es que tiene que cargar cada imagen hiperespectral, con el tiempo de espera que esto conlleva, ya que cada imagen

ocupa por lo menos 600 MB. Anteriormente se procesaban directamente desde la carpeta de origen y se guardaban en la carpeta que tú elegías.

Ahora se guardan todas las imágenes trinarizadas en un .zip, algo menos eficiente que con la versión anterior.

Sobre el uso de Streamlit

La elección de Streamlit como base el desarrollo del proyecto se tomó en las primeras reuniones, dado que al principio se pensó en usar OpenCV para el procesamiento de imágenes, y para mostrar los resultados obtenidos, Streamlit funciona bastante bien.

Sin embargo, a lo largo del desarrollo se fue pensando en añadir cada vez más funcionalidades a la aplicación, y las herramientas que ofrece Streamlit se me fueron quedando cortas. Un problema que tuve fue el tema de editar los componentes frontend que ofrece Streamlit, ya que directamente no ofrece la opción de editarlos con HTML o CSS. Para sortear estos problemas tuve que inyectar código CSS y HTML usando un elemento externo de Streamlit.

Otra cuestión que me ocurrió fue a la hora de crear una ventana emergente, que por suerte actualizando a la última versión de Streamlit, habían añadido como función beta. Pero el tema de que aún esté en desarrollo y cuestiones tan básicas como una ventana emergente todavía no estuviese implementada es un problema a la hora de desarrollar aplicaciones sobre ella.

En definitiva, si hubiera sabido desde un principio que la sencillez de Streamlit me iba a suponer un problema a la hora de escalar la aplicación, probablemente hubiera optado por su alternativa Flask. Las ventajas de Flask es que se puede escalar mejor, te permite una mayor personalización y que lleva más tiempo en desarrollo, siendo más estable.

7.2. Líneas de trabajo futuras

De cara a continuar con el desarrollo de este proyecto, me gustaría hacer algunas sugerencias mediante las cuáles se podrían mejorar las funcionalidades y capacidades de la aplicación.

Detección de gotas con Inteligencia Artificial

Mi primera sugerencia sería la adicción de la inteligencia artificial a la hora de detectar las gotas en las hojas. Lo bueno de utilizar inteligencia

artificial sería ir entrenando un modelo que se especializara en detectar gotas y, en caso de que no fuesen regulares, como las que se usan en este proyecto, fuese capaz de detectarlas igualmente.

Uno de los grandes problemas que se nos han planteado es la cuestión de eliminar el ruido a la hora de detectar las gotas. Aunque en este caso contábamos con la ventaja de que las gotas se habían precipitado en un laboratorio usando una plantilla, el problema seguía ahí. La cosa es que cuando se trate de detectar las gotas en un futuro, las gotas serán irregulares, ya que se estará pulverizando todo el viñedo de una vez, no hoja por hoja como en este caso.

De ahí que se plantee el uso de inteligencia artificial, tanto para facilitar la detección de gotas irregulares como para omitir las irregularidades o defectos de las hojas que causan ruido en el resultado final.

Mejorar el visualizador de imágenes de capas

El visualizador de imágenes que se ha implementado es bastante simple en cuanto a funcionalidades se refiere. Este visualizador te permite principalmente añadir capas, mostrarlas o no, elegir su nivel de transparencia y hacer zoom sobre la imagen resultante.

Lo que se sugiere es añadir la posibilidad de poder mover la capa superior y cambiarla el tamaño. Esto sería para tener la opción de subir imagen de una gota y poder cuadrarla en la imagen de la hoja, para comprobar que coinciden correctamente.

También, se podrían añadir más funcionalidades si se desea, tomando de referencia el conocido editor de imágenes Photoshop.

Detectar la cantidad de producto antifúngico que cubre la hoja una vez secadas las gotas

La tercera sugerencia para continuar el desarrollo del proyecto consiste en conseguir detectar el cubrimiento de producto antifúngico que dejan las gotas con diferentes concentraciones sobre las hojas de vid una vez secas. Este es un punto fundamental a la hora de reducir el uso de fertilizantes en los viñedos.

La tarea consistiría en ir a la zona donde se detectó la gota, y usando la imagen de la hoja una vez seca, determinar cuánta superficie de la gota ha sido cubierta por el producto antifúngico. Para ello se debe saber la

concentración de producto que se ha aplicado en cada gota, y una vez tenemos esa información ya se podría calcular el porcentaje de cubrimiento de la hoja de vid, y saber de forma concreta cuanto fertilizante está haciendo efecto.

Esta sería la línea de trabajo principal en un futuro, ya que es el objetivo final que se está buscando con este proyecto de investigación.

Bibliografía

[1] Atriainnovation — Tecnología hiperespectral. urlhttps://atriainnovation.com/blog/tecnologia-hiperespectral/, 2024. [Internet; accedido el 5-julio-2024].

[2] Chatgpt — chatgpt.com. https://chatgpt.com/, 2024. [Internet; accedido el 2-julio-2024].

[3] Docker — docker.com. https://www.docker.com/, 2024. [Internet; accedido el 8-julio-2024].

[4] Github — github.com. https://github.com/, 2024. [Internet; accedido el 2-julio-2024].

[5] Numpy — numpy.org. https://numpy.org/, 2024. [Internet; accedido el 4-julio-2024].

[6] Opencv — opencv.org. https://opencv.org/, 2024. [Internet; accedido el 4-julio-2024].

[7] Overleaf — overleaf.com. https://es.overleaf.com/, 2024. [Internet; accedido el 2-julio-2024].

[8] Pandas — pandas.pydata.org. https://pandas.pydata.org/, 2024. [Internet; accedido el 4-julio-2024].

[9] scikit-image — scikit-image.org. https://scikit-image.org/, 2024. [Internet; accedido el 5-julio-2024].

[10] Spectral pyhton(spy) — spectralpython.net. https://www.spectralpython.net/, 2024. [Internet; accedido el 4-julio-2024].

[11] Streamlit — streamlit.io. https://streamlit.io/, 2024. [Internet; accedido el 4-julio-2024].

[12] streamlit-image-zoom — github.com/vgilabert94/streamlit-image-zoom. https://github.com/vgilabert94/streamlit-image-zoom, 2024. [Internet; accedido el 5-julio-2024].

[13] Visual studio code — code.visualstudio.com. https://code.visualstudio.com/, 2024. [Internet; accedido el 2-julio-2024].

[14] Zube — zube.io. https://zube.io/, 2024. [Internet; accedido el 2-julio-2024].

[15] Miguel Ángel Lara Blas. Hyperspectral image applied to determine quality parameters in leafy vegetables. No Publicado, 2016.

[16] Gamal M. ElMasry and Shigeki Nakauchi. Image analysis operations applied to hyperspectral images for non-invasive sensing of food quality – a comprehensive review. *Biosystems Engineering*, 142:53–82, 2016.

[17] OpenCV. Image Processing in OpenCV — Morphological Transformations. https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html, 2024. [Internet; accedido el 8-julio-2024].

[18] OpenCV. Image Processing in OpenCV — Thresholding. https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html, 2024. [Internet; accedido el 8-julio-2024].

[19] Wikipedia. Hiperespectral — wikipedia, la enciclopedia libre, 2023. [Internet; descargado 2-junio-2023].

[20] Wikipedia. Css — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=CSS&oldid=159796485, 2024. [Internet; descargado 3-julio-2024].

[21] Wikipedia. Python — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Python&oldid=161085729, 2024. [Internet; descargado 3-julio-2024].

[22] Wikipedia. Xml — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Extensible_Markup_Language&oldid=160549383, 2024. [Internet; descargado 3-julio-2024].

[23] Wikipedia contributors. Feature scaling — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Feature_scaling&oldid=1228410225, 2024. [Online; accessed 8-July-2024].