MÁSTER UNIVERSITARIO EN INGENIERÍA

Escuela Politécnica Superior de Jaén


Universidad de Jaén

Escuela Politécnica Superior (Jaén)


# ESTIMACIÓN EN CONTINUO DEL NIVEL DE SUCIEDAD DE ACEITUNAS


Alumno/a: Oya Monzó, Marina

Tutor/a: Pablo Cano Marchal

Dpto.: Ingeniería Electrónica y Automática

# Contenido

**RESUMEN**

En este Trabajo de Fin de Máster se pretende obtener información a partir de imágenes procedentes de una almazara. Para tal fin, se va a emplear el lenguaje de programación Python y diversas librerías, aunque principalmente, la más conocida en el ámbito de procesado de imagen: OpenCV.

La información que se va a obtener de las imágenes va a ser la cantidad de hoja presente en cada imagen, además de la clasificación de la aceituna, es decir, el estado en el que se encuentra. Toda esta información es de gran utilidad para determinar la calidad del aceite, ya que el estado de la aceituna es uno de los parámetros principales que lo determinan.

Por último, se va a presentar una pequeña interfaz para facilitar la utilización del programa que se desarrolla en este trabajo para cualquier posible usuario.

**ABSTRACT**

The aim of this Master's thesis is to obtain information from images taken in an oil mill. To this end, the Python programming language has been used, in addition to a wide range of libraries of programming functions, focusing on the the most well-known and applied in the field of image processing – OpenCV.

The information to be obtained from the aforementioned images will consist on the quantity of olive leaves, and the correct classification according to the condition of the harvested olives. Since the state of these olives is one of the main parameters to determine the quality of the olive oil, this information, together with the leaf quantity, will be extremely valuable.

Furthermore, an user-interface will be provided to make the implementation of this programme more feasible.

# 1. INTRODUCCIÓN

## 1.1. Justificación

Jaén es la región en la que se produce mayor cantidad de aceite de oliva en España y en el mundo, contando con 550.000 hectáreas de olivar, generando un 20% de la producción mundial de aceite de oliva. Estas cifras exigen un control de este sector en la provincia que requiere de multitud de recursos humanos, materiales e intelectuales dedicados al desarrollo de medios que aseguren la calidad del aceite.

La provincia de Jaén se autoproclama en España como “Capital Mundial del Aceite de Oliva”, por lo que ejerciendo esta capitalidad mundial, la Universidad de Jaén acoge un gran número de estudios relacionados con el olivo, la aceituna y el aceite de oliva. (Esencia de olivo, 2011)

En este Trabajo de Fin de Máster se va a realizar un seguimiento del estado de la aceituna a la llegada a una almazara para obtener información sobre la calidad de la aceituna, que determinará la calidad del aceite que se obtenga de ella.

La realización del seguimiento se va a realizar a partir de imágenes tomadas en una almazara, y mediante visión por computador, se va a extraer y analizar la información. Para este proceso se va a utilizar la librería de programación *OpenCV* (Open Source Computer Vision Library), con la idea de crear una herramienta potente para solventar los objetivos de este Trabajo de Fin de Máster.

## 1.2. Objetivos del proyecto

El objetivo principal de este Trabajo de Fin de Máster es detectar el estado de la aceituna en el momento de llegada a la almazara, que es relevante para conocer la calidad del aceite que se obtiene a partir de ella. La

calidad de la aceituna depende fundamentalmente de si ésta se recolecta del suelo o del olivo, aunque hay otros factores que pueden afectar que no se va a estudiar en este documento.

Otro objetivo del proyecto consiste en cuantificar también la cantidad de hoja presente en cada imagen, ya que es otro parámetro relevante para estudiar.

Por último, se va a diseñar e implementar un entorno fácil e intuitivo para que cualquier usuario pueda trabajar con el programa resultante de manera práctica y rápida, mediante una interfaz.

El lenguaje de programación con el que se va a desarrollar este trabajo es Python, empleándose la herramienta Anaconda, debido a que contiene la librería *OpenCV* y otras complementarias que se necesitan para conseguir los objetivos.

La herramienta diseñada está destinada para facilitar la clasificación de la aceituna para una mejor selección para la elaboración de aceite de oliva.

## 1.3. Metodología

Para conseguir el objetivo de este proyecto, se va a seguir una serie de pasos que se resumen a continuación:

- Seleccionar los diferentes modelos reproductivos del color que se van a implementar y estudiar los resultados obtenidos con cada uno de ellos.
- Procesar las imágenes para evitar trabajar con aquellas que sean de la cinta transportadora vacía. Para ello, se va a trabajar con la textura de la imagen para desechar éstas.
- Identificar las dos familias principales presentes en una imagen modelo, que son la hoja y la aceituna, empleando el clasificador *K-means*.
- Extraer la información del clasificador, obteniendo así una máscara a partir de la umbralización de las imágenes para localizar las dos familias en todas las imágenes.

- Contabilizar el porcentaje de hoja presente en la imagen a partir de los pixeles clasificados.
- Localizar los núcleos de las aceitunas de las imágenes usando un umbral para localizar los brillos que generan tanto la forma de la aceituna como el sistema de iluminación de la cámara.
- Segmentar cada aceituna de la imagen con ayuda del modelo *Active Contours Without Edge,* el modelo de *Chan-Vese* empleado para detectar contornos sin gradientes.
- Estudio de cada aceituna para detectar su proveniencia: aceituna buena, aceituna del suelo, aceituna estropeada. Para este paso, se utilizarán herramientas tanto de umbralización, como de detección de bordes.
- Resumir la información obtenida en una aplicación Web con ayuda de Dash by Plotly, para facilitar su utilidad mediante una interfaz usuario sencilla y simple.

## 1.4. Contenido de la memoria

En la introducción se ha realizado la justificación de este proyecto y se explica de manera breve los objetivos principales que persigue, así como las pautas a seguir para su desarrollo. Además, se resume el contenido de la memoria y su estructura.

Por su parte, en el punto Estado del Arte, se va a resumir el estado actual en el que se encuentran los estudios relacionados con este tema. También se van a explicar las bases teóricas sobre las que se fundamenta el proyecto.

En el siguiente punto, titulado metodología empleada, se ha empleado para facilitar la comprensión del texto, realizando una descripción de las técnicas usadas en el desarrollo del código.

En el Desarrollo del Proyecto, se trata de explicar y detallar las funciones y modelos principales empleados. Se presenta un diagrama de flujo que se detalla a lo largo de todo el apartado.

Por su parte, en los resultados obtenidos, se van a comentar ejemplos de la información obtenida en el proyecto, resaltando los más concluyentes. También se comentarán los problemas que se han presentado en el desarrollo del código

En el apartado de conclusiones y líneas de futuro, se encuentra un resumen explicativo de las conclusiones obtenidas de todo el proceso, así como la conformidad de los resultados a los objetivos tomados. Por otro lado, también se va a comentar las líneas de futuro que abre este Trabajo de Fin de Máster para posteriores estudios

En la bibliografía se presentan las referencias que se han empleado para facilitar y complementar el desarrollo del documento del proyecto.

Por último, se encuentra el apartado de anexos, donde se encuentra el código del programa con el que se han obtenido los objetivos del proyecto. Además, también se presenta la interfaz realizada para dicho programa.

# 2. ESTADO DEL ARTE

## 2.1. Visión por computador

La visión por computador es aquella ciencia que desarrolla el entorno teórico y algorítmico con el que se extrae y analiza información mediante secuencia de imágenes, considerándose un sub-campo de la inteligencia artificial (IA). (Alberto, 2013)

El objetivo de la visión artificial es que un computador, mediante programación, sea capaz de extraer las características de interés de una imagen para su descripción e interpretación.

Un área estrechamente ligada a este campo, es el procesamiento de imágenes, mediante el cual se mejora la calidad de las imágenes para su posterior interpretación.

Existen múltiples aplicaciones de la visión computacional en la actualidad, siendo destacables en los campos:

- Robótica y automoción, con ayuda de sensores y cámaras para generar modelos tridimensionales mediante mapeo de una escena.
- Manufactura, para localizar e identificar piezas.
- Medicina, para analizar e interpretar imágenes.
- Astronomía, para localizar objetos en el espacio.
- Compresión de imágenes.
- Química, física y biología, para interpretar la información de imágenes microscópicas.
- Dibujos y planos, para reconocimiento de textos e interpretación de dibujos y mapas.
- Detección y reconocimiento de caras humanas.
- Búsqueda de imágenes digitales por su contenido

El entorno *OpenCV* no utiliza todos los campos que recoge la visión por computador, ya que se enfoca principalmente en el procesado de imágenes.

## 2.2. Entorno *OpenCV*

*OpenCV* (Open Source Computer Vision) es una biblioteca libre que implementa operaciones de infraestructura y procesamiento de imágenes y visión. Inicialmente, fue desarrollada por Intel, siendo su primera aparición en 1999. Su publicación se consigue bajo licencia BSD, que es una familia de licencias de software libre permisiva.

Dispone de una gran cantidad de algoritmos que permiten el análisis de imagen y el aprendizaje automático.

Esta librería es multiplataforma, ya que se encuentra disponible para Windows, Mac, Linux, Android, Maemo, FreeBSD, OpenBSD, iOs, BlackBerry 10 y OS X. Su interfaz principal se encuentra en C++, aunque en la actualidad, hay interfaces en diversos lenguajes, como en C, Python, Java y Matlab/Octave. (Alberto, 2013)

## 2.3. Estudios relacionados anteriores

Los sistemas de visión por computador se han utilizado mucho en procesos de clasificación e inspección de productos alimenticios, debido a la mejora de la calidad que supone un método que, consumiendo menos recursos humanos, conlleva a una alta confianza de los resultados.

La utilización de esta herramienta en el sector olivarero no está extendida en la actualidad. Los pocos estudios que existen sobre el tema, se centran en la detección de aceitunas defectuosas.

Entre todos estos estudios, destacan los siguientes:

En (R. Diaz, Comparison of three algorithms in the classification of table olives by means of computer vision, 2004), se realiza un sistema de inspección automática que, empleando visión por computación, detecta la presencia de defectos superficiales en aceitunas para mesa. Para este fin, se probaron tres

algoritmos: Bayesian, análisis PLS y redes neuronales. Con la utilización de las redes neuronales se obtuvo una precisión de más del 91%.

En (R. Furferi, 2010) se empleó la visión artificial con dos objetivos: detectar defectos superficiales y evaluar el índice de maduración de la aceituna en función del color. En este trabajo, se empleó el umbral K-Means para establecer el color.

En (E. Guzmán, 2013), se desarrolló un método para clasificar aceitunas en base un algoritmo de segmentación a partir de un sistema de visión del espectro infrarrojo.

La visión por computación también se ha empleado para estimar el índice de maduración de la aceituna en (E. Guzmán V. B.-M., 2015) empleando diferentes muestras de aceitunas. En este estudio, los autores también presentaron una metodología para calcular el índice de maduración en cada aceituna de manera individual.

Mediante redes neuronales, se estimó el contenido de aceite de oliva en aceitunas intactas, con modelos de predicción basados en análisis de imágenes visuales y regresiones lineales. En (T. Ram, 2010) se alcanzaron los mejores resultados usando ANNs, obteniendo correlaciones lineales de 0.87.

# 3. METODOLOGÍA EMPLEADA

En este apartado se va a exponer todos los elementos empleados en el desarrollo del Trabajo de Fin de Máster en los siguientes bloques principales: Bibliotecas y funciones principales empleadas, breve introducción al entorno OpenCV y los diferentes modelos representativos del color valorados.

## 3.1. Librerías principales empleadas en Python

- ***OpenCV***

Es la principal librería que se ha empleado en este Trabajo de Fin de Máster, cuyo fin es explotar los recursos que ofrece *OpenCV* para conseguir los objetivos definidos con anterioridad.

Las características e información de esta librería, se han comentado con anterioridad, por lo que se pasará a explicar otras librerías adicionales que se han empleado en el desarrollo del trabajo.

- ***NumPy***

NumPy es una librería de Python que significa “Numerical Python”, siendo el paquete fundamental para la computación científica con Python.

Esta librería proporciona: objetos de matriz N-dimensional, funciones de radiodifusión, herramientas para integrar código C / C ++ y Fortran, además de álgebra lineal y transformada de Fourier.

Esta librería, agrega un mayor soporte a Python en el uso de vectores y matrices, formando una biblioteca de funciones con un alto nivel matemático para realizar operaciones, modificar datos o almacenarlos de una manera eficiente.

- ***Pandas***

Debido a que Microsoft Excel es un programa muy potente empleado para la visualización y análisis de datos, la existencia de recursos para extraer datos

y guardarlos para facilitar el procesado de imagen es un punto clave. La biblioteca *pandas* es una biblioteca de código abierto en Python que posibilitan el análisis y manipulación de datos en la programación.

La parte más importante en *pandas* es el *DataFrame*, donde se almacenan y modifican los datos. *DataFrame* es una herramienta similar a una tabla SQL o a una hoja de cálculo en Excel.

Una vez recogidos los datos en *DataFrame*, se pueden guardar en un archivo Excel, y posteriormente acceder a ellos para su reutilización.

- ***Seaborn***

*Seaborn* es una biblioteca de visualización de datos estadísticos de Python basada en *matplotlib* (biblioteca de trazado 2D en Python) y ligado a la estructura de datos de *pandas*.

*Seaborn* emplea la visualización de manera que facilita la exploración y compresión de los datos mediante trazado estadístico de éstos. (Waskom, 2012)

- ***Scikit – learn***

*Scikit - learn* es una colección de algoritmos para el procesado de imagen y visión por computador. Es una biblioteca de uso y descarga libre, generada por una comunidad de voluntarios.

Ha sido necesario el empleo de esta biblioteca debido a que el algoritmo *Chan-Vese* para detectar objetos no se encuentra en *OpenCV*.

El algoritmo de segmentación *Chan-Vese* tiene como objetivo detectar objetos cuyos límites no están definidos de manera clara.

La implementación del algoritmo en *scikit-learn* solo se puede aplicar para imágenes en escala de grises, debido a que no se emplea el factor de área, simplificando el método.

Gracias a que el algoritmo devuelve una lista de valores que corresponden a la energía de cada iteración, facilita el ajuste de los diversos parámetros para un ajuste más óptimo. (Scikit-image)

## *3.2.* Entorno *OpenCV*

En este apartado, se presentan los métodos de procesado de imagen empleados en el desarrollo del Trabajo de Fin de Máster y que se encuentran disponibles en el entorno OpenCV.

### 3.2.1. Algoritmos de clasificación

Existen diversos tipos de algoritmos de clasificación que se pueden aplicar en el campo de la visión por computador. En este Trabajo de Fin de Máster, el principal empleado ha sido *K-means*, que implementa el aprendizaje no supervisado. *K-means*, agrupa datos en clases (*clusters*) de manera que los puntos de una clase sean similares entre ellos y diferentes de los puntos de los otros grupos. El objetivo de este algoritmo es realizar la clasificación de un conjunto de datos sin etiquetar.

Este algoritmo permite determinar los centroides de las k clases existentes en la imagen y estimar su posición tomando regiones que contengan pixeles que posean valores similares. Estos valores se obtienen mediante el cálculo de la media, estando cada clase caracterizada por su centroide, es decir, por el valor de la media que se ha asignado a dicha clase. El número de clases existentes k que posee la imagen es un dato a introducir en el algoritmo.

El algoritmo *K-means* se procesa en cuatro pasos:

- Se seleccionan las K clases existentes en las imágenes (si no se selecciona un número de clases, trabaja con 9 clases por defecto).
- Asignar cada elemento al centroide más cercano (se asigna el más próximo al objeto)
- Se recalculan los centroides de las clases usando:

$$c_j = \frac{1}{|C_j|}\sum z$$

donde Cj corresponde al número de elementos totales en la clase, z representa cada elemento del conjunto de datos y cj es un centroide

- Iterar las etapas 2 y 3 hasta que no se realicen más reasignaciones para intentar obtener la solución óptima.

A pesar de existir diversos modelos de clasificación, como *KNN*, *RTREE* o *SVM*, en este Trabajo de Fin de Máster se ha empleado el modelo de clasificación *K-Means* debido a su simplicidad y a que es el único modelo implementado en las librerías utilizadas no supervisado, por lo que es más sencillo y rápido de implementar. (Irene Gómez Moreno)

### 3.2.2. Morfología de imagen

La morfología se basa en operaciones de teoría de conjuntos, siendo útiles para simplificar imágenes, conservando las principales características de los objetos.

Las dos operaciones morfológicas principales son la erosión y la dilatación. Estas funciones requieren de un elemento estructurante (*kernel*) para su aplicación, siendo determinante el tamaño y forma del *kernel* para obtener el resultado óptimo de la operación.

Para generar un elemento estructurante en ***OpenCV***, se emplea la función:

**cv2.getStructuringElement(***Forma*, *Tamaño***)**

*Forma*: **MORPH_RECT**, **MORPH_CROSS**, **MORPH_ELLIPSE** (determina la forma del elemento estructurante)

*Tamaño*: define el tamaño del elemento estructurante, indicándose como una matriz (ej: (5,5); (3,3)).

Dependiendo de la forma y tamaño del elemento estructurante, se obtendrán diferentes resultados al realizar la operación.

La dilatación y la erosión son funciones similares que se complementan pudiéndose formular como una relación de dualidad: la erosión hace al objeto lo que la dilatación hace al fondo.

- **Dilatación**

La operación de dilatación genera el efecto de expandir la región de la imagen con la forma y tamaño del *kernel* utilizado. La dilatación, dada una imagen IM y un elemento estructurante K, se define como:

$$IM \oplus K = \{x | (\hat{K}_x \cap IM \neq \emptyset\}$$

La dilatación puede representarse como la unión de los trasladados. Esta función en *OpenCV* se programa empleando:

**cv2.dilate(***scr*, *dst*, *kernel***)**

*scr*: imagen de entrada

*dst*: imagen de salida

*kernel*: elemento estructurante generado con anterioridad

- **Erosión**

La operación de erosión genera el efecto de adelgazamiento de la región de la imagen con la forma y tamaño del *kernel* utilizado. La erosión, dada una imagen IM y un elemento estructurante K, se define como:

$$IM \ominus K = \{x | (\hat{K}_x \subseteq IM\}$$

La erosión puede representarse como la intersección de los trasladados negativos. Esta función en *OpenCV* se programa empleando:

**cv2.erode(***scr*, *dst*, *kernel***)**

*scr*: imagen de entrada

*dst*: imagen de salida

*kernel*: elemento estructurante generado con anterioridad

A partir de las dos funciones básicas de la morfología, es posible realizar operaciones de apertura y cierre. Las operaciones de apertura y cierre no se encuentran disponibles en *OpenCV* de manera directa.

- **Apertura**

La apertura suaviza los contornos de los objetos de una imagen, eliminando regiones de menor tamaño que el elemento estructurante. La apertura de una imagen IM por un elemento estructurante K se forma a partir de un proceso de erosión, y su posterior dilatación, quedando definida como:

$$IM \circ K = (IM \ominus K) \oplus K$$

**cv2.morphologyEx(***img*, *Función*, *kernel***)**

*img*: imagen de entrada

*Función*: **cv.MORPH_OPEN**

*kernel*: elemento estructurante generado con anterioridad

- **Cierre**

El cierre elimina huecos de menor tamaño que el elemento estructurante, haciendo que forme parte del objeto. El cierre de una imagen IM por un elemento estructurante K se forma a partir de un proceso de dilatación, seguido por un proceso de erosión, quedando definida como:

$$IM \cdot K = (IM \oplus K) \ominus K$$

**cv2.morphologyEx(***img*, *Función*, *kernel***)**

*img*: imagen de entrada

*Función*: **cv.MORPH_CLOSE**

*kernel*: elemento estructurante generado con anterioridad

### 3.2.3. Umbralización de la imagen

La binarización de una imagen mediante umbralización es un ejemplo de operación puntual. Una operación puntual es aquella en la que se obtiene una imagen de salida de manera que cada pixel de ésta depende únicamente de cada pixel de la imagen de entrada.

La umbralización es una técnica empleada para segmentación rápida, con un coste computacional bajo. Para obtener una imagen binaria, se toma un valor umbral T (threshold), de manera que cada pixel de la imagen de salida solo pueda tomar dos valores: 0 y 1, negro y blanco, en función de si está por encima o por debajo del valor umbral.

Con esta técnica se puede conseguir separar el objeto de interés del resto de la imagen, siendo el problema la determinación del valor umbral.

### 3.2.4. Segmentación Chan-Vese

El modelo de detección de contornos activos sin bordes de Chan-Vese es un modelo para detectar objetos en una imagen dada. Este modelo está basado en técnicas que se fundamentan en la evolución de la curva para segmentar mediante niveles de energía.

Este modelo es capaz de detectar contornos sin necesidad de definir gradientes, ya que en este caso, el problema se convierte en un flujo de curvatura media que se detendrá en el límite deseado. Este término de

detención se basa en las técnicas de segmentación de Mumford-Shah, obteniendo un modelo que es capaz de detectar contornos con o sin gradiente, objetos con límites mal definidos o incluso límites discontinuos.

Definiendo una curva inicial, que puede posicionarse en cualquier zona de la imagen y tomar cualquier forma sin que varíe el resultado obtenido por el proceso. A partir del proceso de minimizar una función concreta que posee el modelo, se puede detectar el contorno en una imagen.

Para mejorar el método anterior, se han añadido también dos parámetros para tener mayor control del algoritmo: la longitud y el área de la curva.

Este proceso no se va a explicar en detalle debido a que el método está implementado en Python de una manera más simple y directa. (Tony F. Chan, 2001)

## 3.3. Color

El color es un fenómeno perceptual que el ser humano es capaz de detectar a diferentes longitudes de onda del espectro visible (400 – 700 nm). Debido a que el ojo posee unos sensores que pueden dividir las señales en tres clases, cuya respuesta depende de la banda de longitudes de onda a las cuales son más sensibles.

La identificación de la información cromática se realiza mediante la combinación de estas tres señales, que conforman la variedad de colores que se pueden distinguir. Estos son los conocidos como colores primarios (rojo, verde y azul). De la suma de estos tres colores, se forma el blanco, mientras que de la combinación aditiva en parejas y partes iguales de estos colores, se forman los secundarios (amarillo, magenta y cian). Si se combinan los secundarios, se obtiene el negro. Esto se puede aprecian en la figura.


**Imagen 1. Diagrama cromático para el sistema**

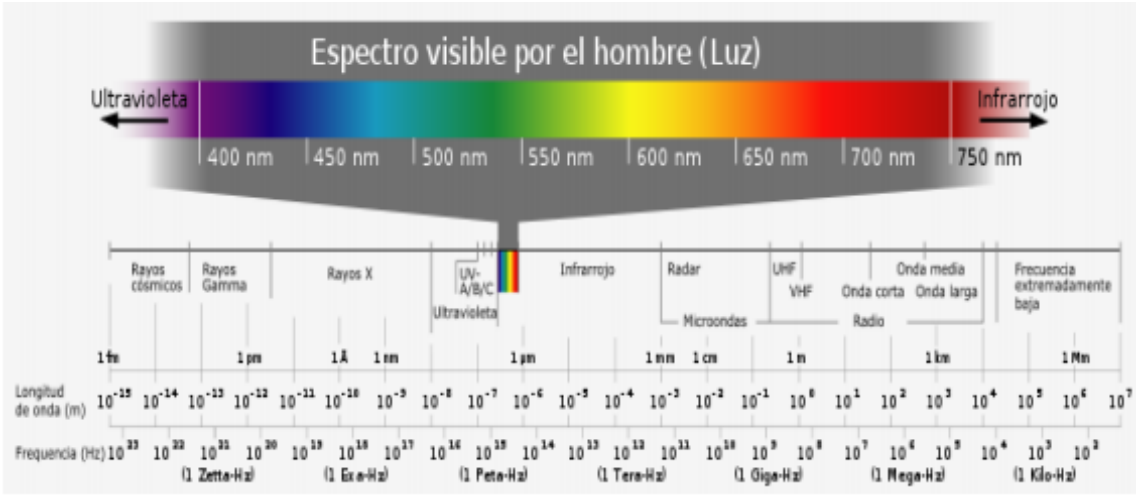


**Imagen 2. Espectro visible por el hombre**

El color en visión es relevante porque puede facilitar la extracción de características para la identificación de objetos en una imagen. En contraste, respecto a las imágenes monocromáticas, el ojo humano puede distinguir infinidad de colores, mientras que solo distingue alrededor de una veintena de niveles de gris, por lo que poder disponer de imágenes a color es un punto muy importante para el reconocimiento de objetos.

Hay diversas formas de organizar los diferentes colores a partir de componentes básicas, conocidas como espacios de color. Se han realizado diversas pruebas respecto al modelo reproductivo del color a seleccionar para clasificar las dos familias principales de las imágenes: la aceituna y la hoja. A

continuación, se va a explicar brevemente algunos espacios de color que se han empleado en el desarrollo de este Trabajo de Fin de Máster.

Han sido tres opciones las estudiadas, todas ellas trabajan con espacios de color tridimensionales, debido a la disponibilidad de imágenes a color en este proyecto: RGB (Red, Green, Blue), HSV (Hue, Saturation, Value) y CIE 1976 L*a*b (que de manera abreviada se conoce como CIELAB). (Guerrero, 2015)

### 3.3.1. Modelo RGB

Este modelo se basa en los tres sensores humanos, considerando que todos los colores son una combinación de tres colores primarios de la luz: R (red), G (green), B (blue). Es un modelo basado en la síntesis aditiva, por lo que se obtienen los colores del espectro como combinación de los diferentes valores de intensidad de los tres colores de la luz primarios.

Para visualizar el modelo se puede emplear el cubo RGB, que compone su representación tridimensional, en cuyos vértices se encuentran los componentes primarios. La combinación lineal de los primarios forman todos los demás colores, situándose el color blanco en el centro del cubo, con la misma proporción de cada vértice. Un ejemplo del cubo RGB se puede apreciar en la imagen.

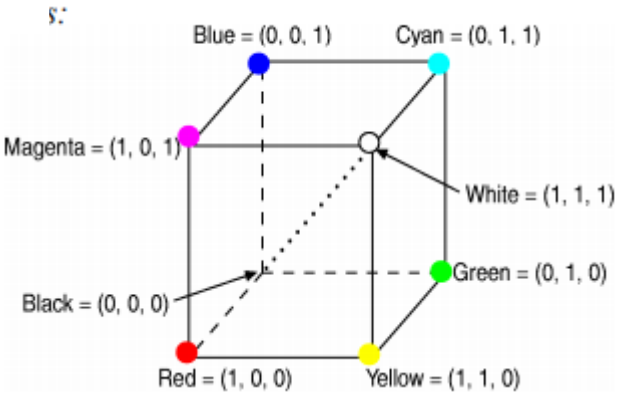


**Imagen 3. Cubo RGB**

### 3.3.2. Modelo HSV

El modelo reproductivo del color HSV se trata de una transformación no lineal del espacio de color RGB, utilizándose como progresiones de color. Este modelo, conforma el color en base a tres componentes: H (Hue), S (Saturation), V (Value), que también conforman tres matrices en los programas de procesado de imagen.

Se puede representar con un cono hexagonal del sistema de coordenadas cilíndricas, en el que cada arco de la circunferencia superior lo conforma un color primario puro y las mezclas puras que se forman entre ellos. El pigmento blanco se encuentra en el centro de la circunferencia, por lo que la saturación se consigue disminuyendo la saturación en valor constante. El pigmento negro se encuentra en la cúspide el cono, por lo que las sombras se obtienen disminuyendo el valor a saturación constante.

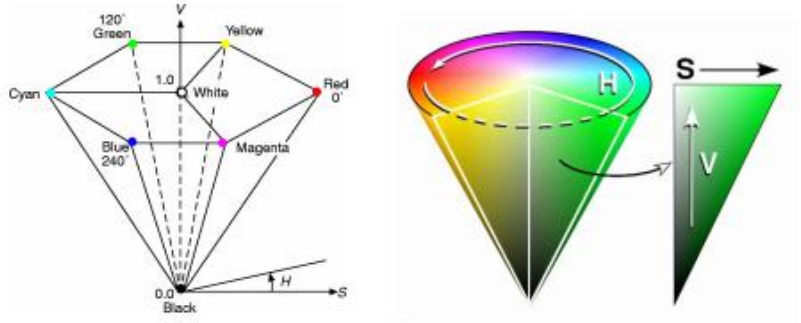


**Imagen 4. Modelo de color HSV con un único cono hexagonal**

### 3.3.3. Modelo CIELAB

La ‘Comisión Internacional de Iluminación’ (CIE) estandarizó como colores primarios el azul, verde y rojo. El objetivo de esta selección es evitar las componentes negativas en la representación del espacio de color.

Finalmente, el modelo seleccionado para la clasificación de la hoja ha sido CIELAB, debido a que se realiza una clasificación más aproximada que en

los anteriores casos. CIELAB es modelo cromático cuya función es representar el espacio de color que percibe el ojo humano. Los tres espacios de color representan la luminosidad del color (L), la posición entre el rojo y el verde (a) y la posición entre el amarillo y el azul (b). Este espacio de color se puede representar mediante un diagrama. (Entendiendo el espacio de color CieLAB, 2014)

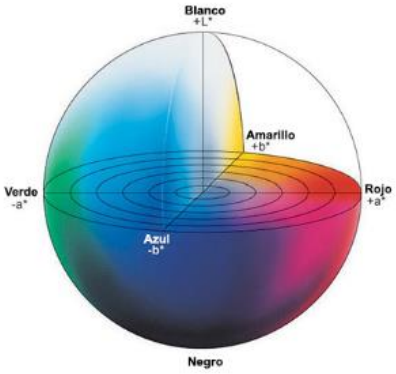


**Imagen 5. Diagrama CIELAB**

# 4. DESARROLLO DEL PROYECTO

En este apartado se va a desglosar toda la información necesaria para comprender el desarrollo de este Trabajo de Fin de Máster.

## 4.1. Diagrama de flujo del proceso

En la imagen 6 se muestra un diagrama de flujo que resume el proceso realizado por el código fuente adjuntado en el primer anexo.

En primer lugar, se ha clasificado la aceituna y la hoja para calcular el porcentaje de hoja presente en cada imagen.

Una vez clasificado este dato, se ha obtenido individualmente las aceitunas presentes en sub-imágenes para llevar a cabo la clasificación del tipo de aceituna.

En la clasificación del tipo de aceituna, que es el paso final del código fuente, se ha desglosado en tres: aceituna buena, aceituna rugosa y aceituna del suelo.

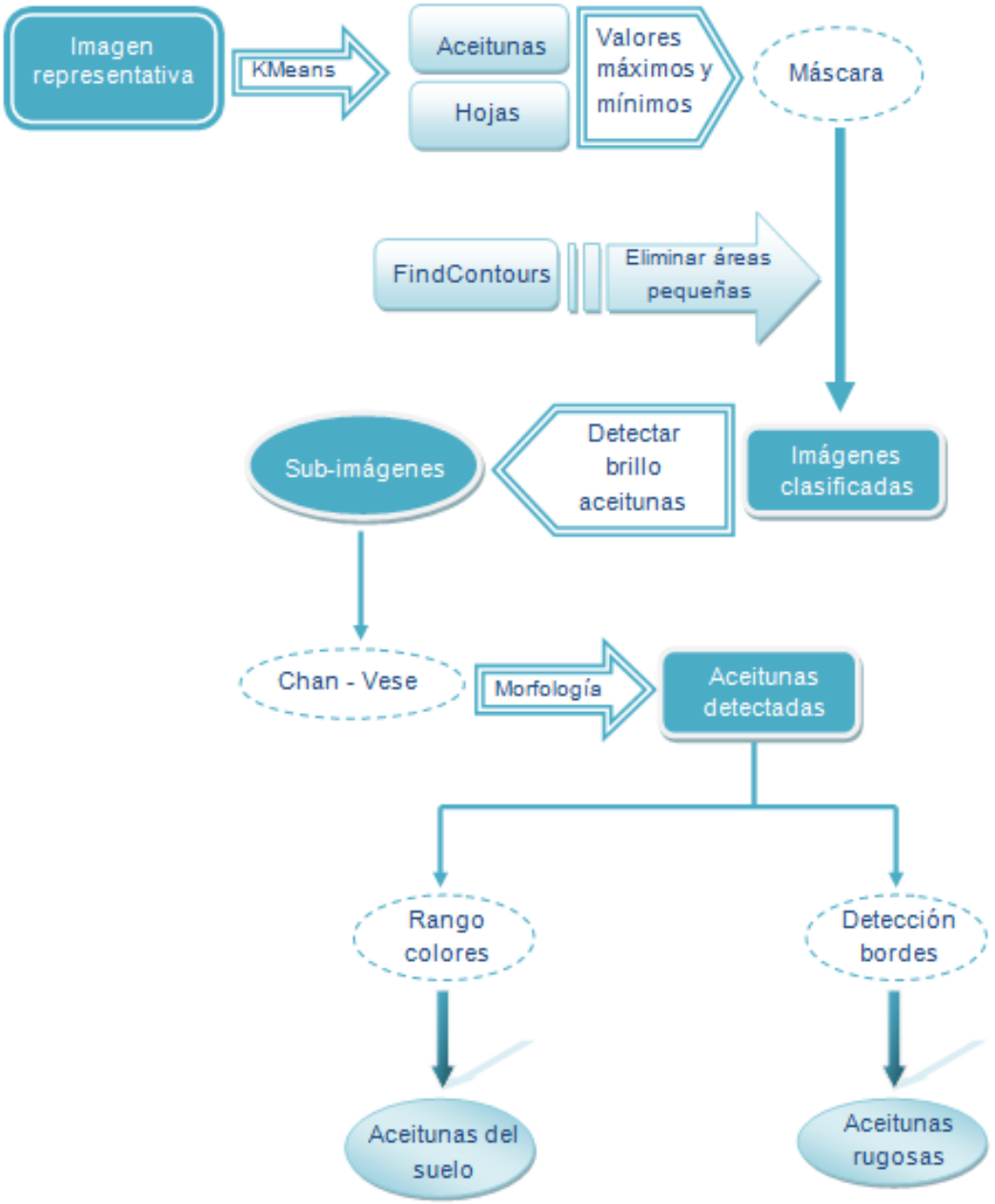


**Imagen 6. Diagrama de flujo del proceso**

## 4.2. Selección del modelo reproductivo del color para detectar la hoja

Para detectar la hoja se han realizado pruebas con los diferentes modelos reproductivos del color y con ayuda del clasificador *K-Means*. Las diferentes pruebas se resumen a continuación:

### 4.2.1. RGB

El empleo de este modelo para conseguir el objetivo del Trabajo de Fin de Máster, a pesar de ser el más intuitivo a nivel usuario, no ha sido posible debido a que los resultados obtenidos no son concluyentes para la clasificación.

En *OpenCV* se consigue el formato BGR por defecto al leer la imagen de la siguiente manera:

**cv2.imread(***'nombrearchivo.jpg'***,** *Función***)**

*Función*: **cv2.IMREAD_COLOR** (por defecto, el formato es BGR)

Para clarificar este hecho, abajo se tiene un ejemplo de clasificación de las imágenes del proyecto usando este modelo. Se han hecho pruebas con diferentes números de clases: 3, 4,5 y 6, para intentar obtener el número de clases óptimo.

**Imagen 7. Imagen original**

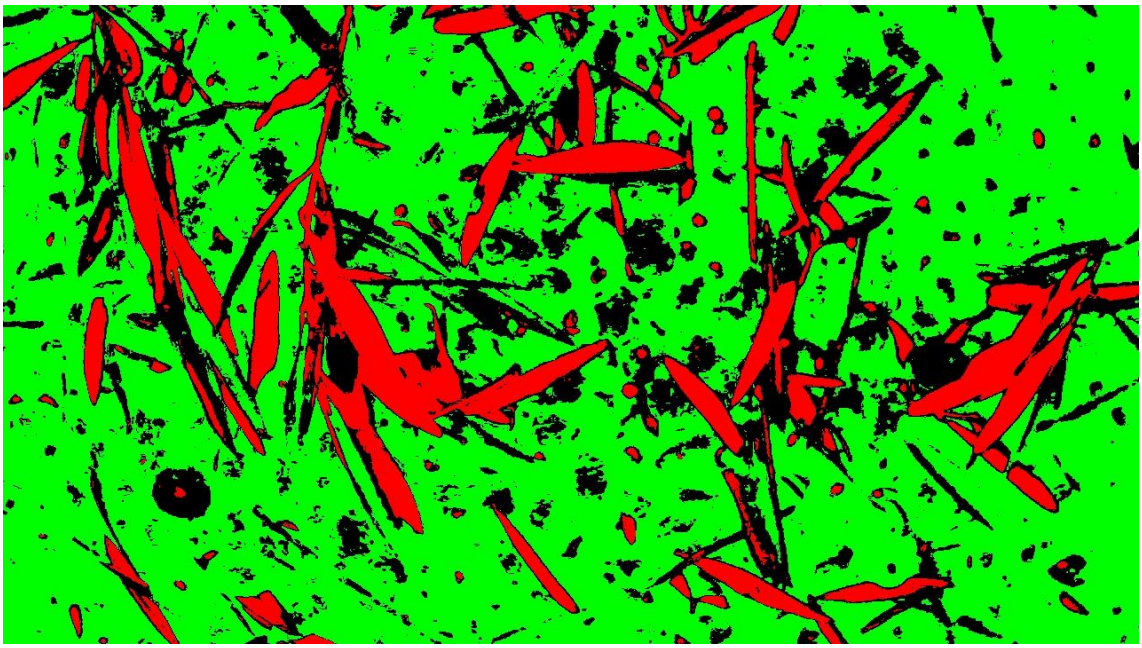

**Imagen 8. Imagen clasificada con 3 clusters**

La imagen que se ha clasificado mediante *K-Means* con 3 familias distintas, selecciona a la perfección las hojas claras de la imagen, mientras que

las hojas oscuras se mezclan con el fondo, perdiendo dicha información. Por otra lado, la tercera familia (negra) la forman algunos brillos de las aceitunas y algunas hojas de tonos más oscuros.

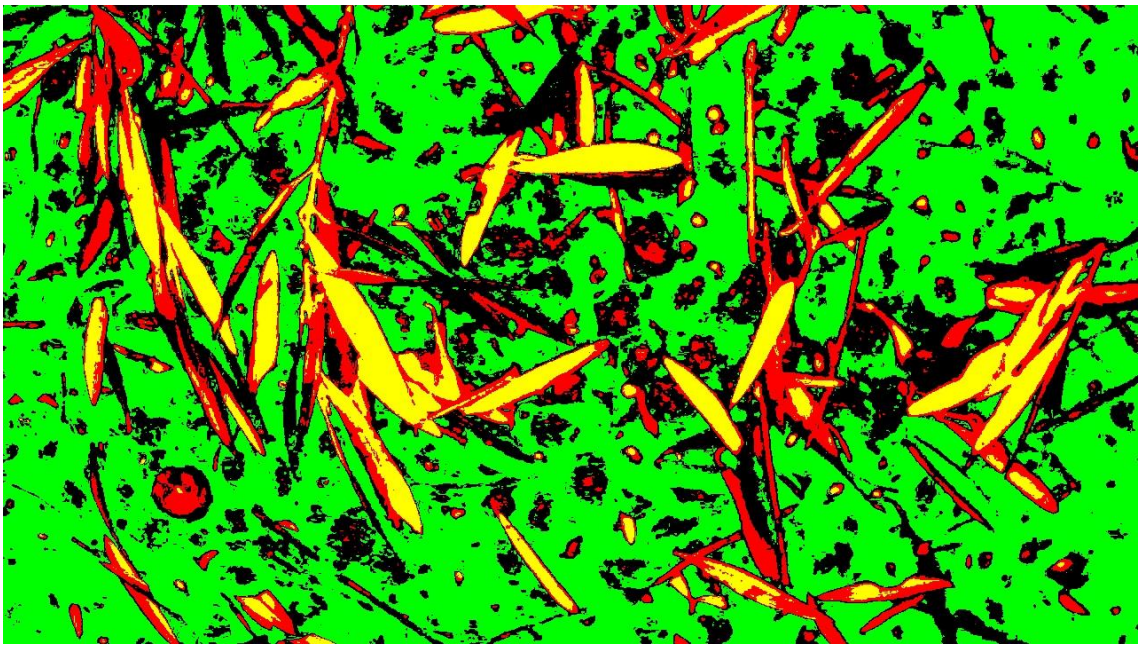

**Imagen 9. Imagen clasificada con 4 clusters**

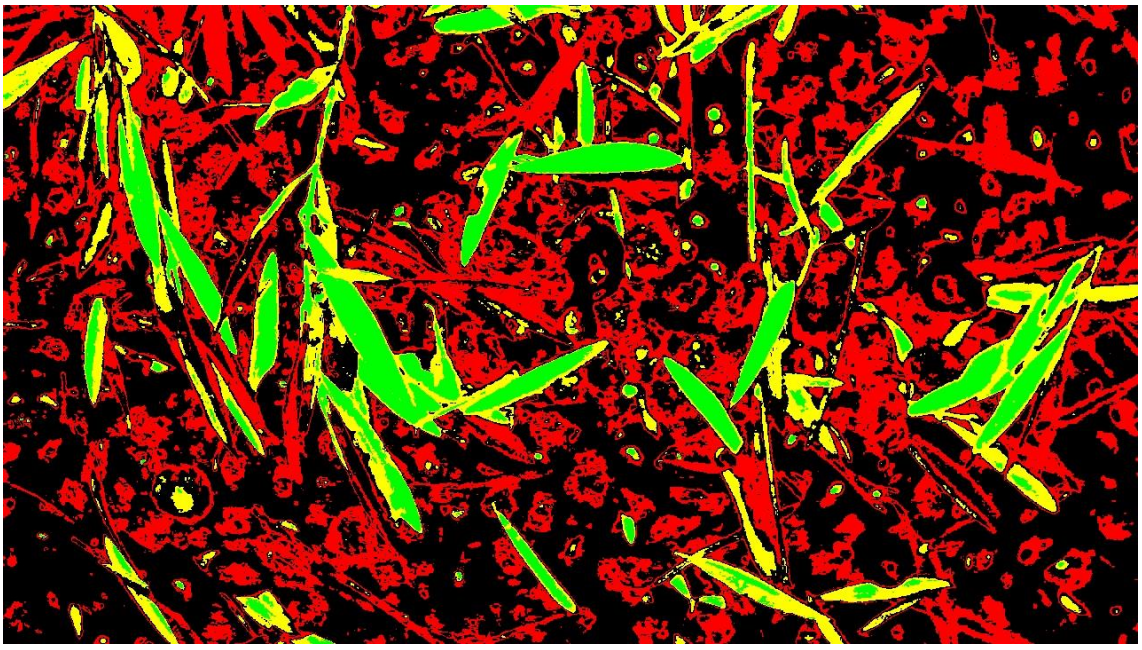

**Imagen 10. Imagen clasificada con 5 clusters**

Al introducir 5 familias en el clasificador, se detectan los bordes que forman las aceitunas por sus caras rugosas, por lo que se obtiene una información que no facilita el proceso de selección de ambas familias.

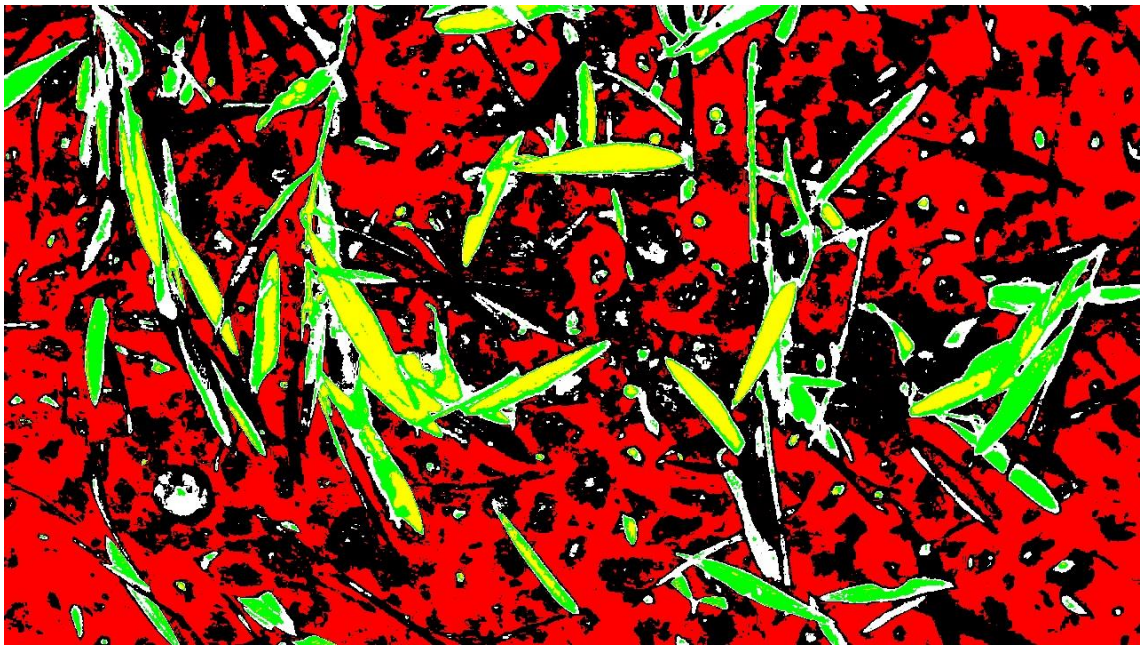

**Imagen 11. Imagen clasificada con 6 clusters**

Como se puede apreciar, la clasificación facilitada por *K-Means* no es óptima, ya que mezcla las familias que se pretenden separar. Aunque las hojas más claras sí son fácilmente clasificadas, las hojas más oscuras de la imagen se mezclan con aquellas aceitunas con tonos más grises. También se detectan los brillos que causan las aceitunas por la forma circular y rugosa que poseen, facilitando tal cantidad de información mezclada que no se obtienen resultados concluyentes.

Otros resultados que se tienen de este modelo han sido obtenidos mediante procesos de umbralización, cuyo resultado tenemos en la Imagen 12. Estos resultados han sido peores que los obtenidos con el clasificador *K-Means*, a pesar de ser un método que implica menor coste computacional, por lo que no se ha tenido en cuenta esta opción para la solución al problema planteado en este trabajo.

**Imagen 12. Imagen obtenida por rango de colores en una máscara**

### 4.2.2. HSV

Para el empleo del modelo HSV, se requiere un cambio de formato en *OpenCV* que se implementa de la siguiente manera:

**cv2.cvtColor(***Imagen***,** *Función***)**

*Función*: **cv2.COLOR_BGR2HSV** (cambia el formato de color de BGR a HSV)

Se han detectado problemas que suelen aparecer en *OpenCV* en la representación de imágenes en este formato. Se produce una distorsión de la imagen, apareciendo un efecto pixelado que imposibilita su utilización, pues se pierde gran cantidad de información. Estos problemas aparecen debido a que las imágenes estudiadas tienen formato *jpg*, además de un histograma de color desplazado a la izquierda, es decir, con tonalidades muy parecidas, como se aprecia el histograma generado de una imagen aleatoria de la muestra.

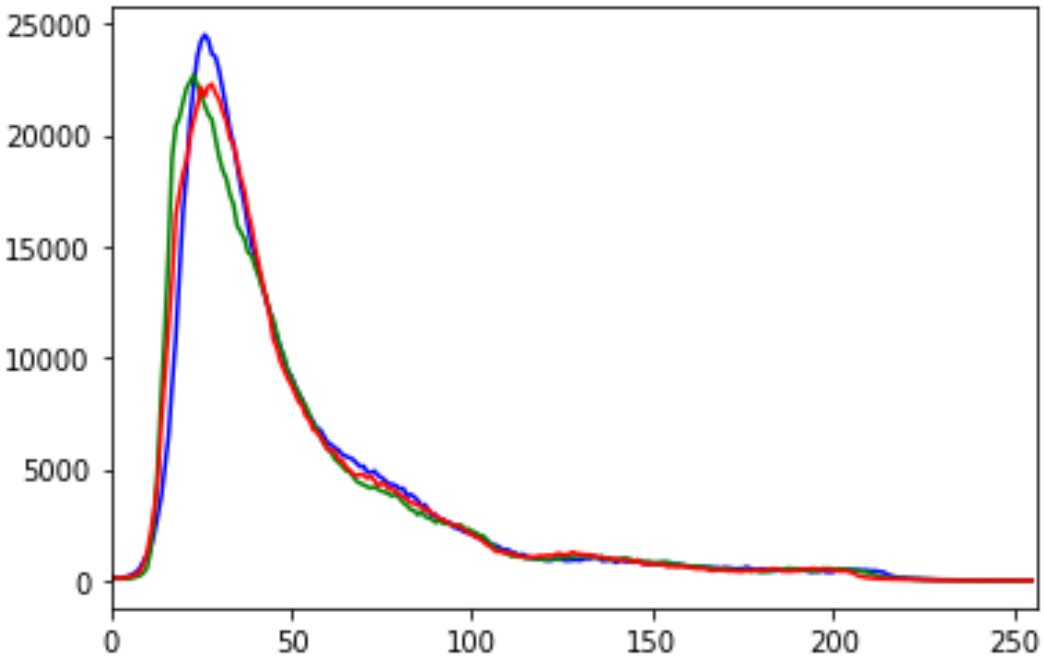


**Imagen 13. Histograma de una imagen aleatoria**

A continuación se muestra la imagen anterior una vez que se ha procesado mediante el clasificador *K-Means*. Se aprecia el defecto comentado.

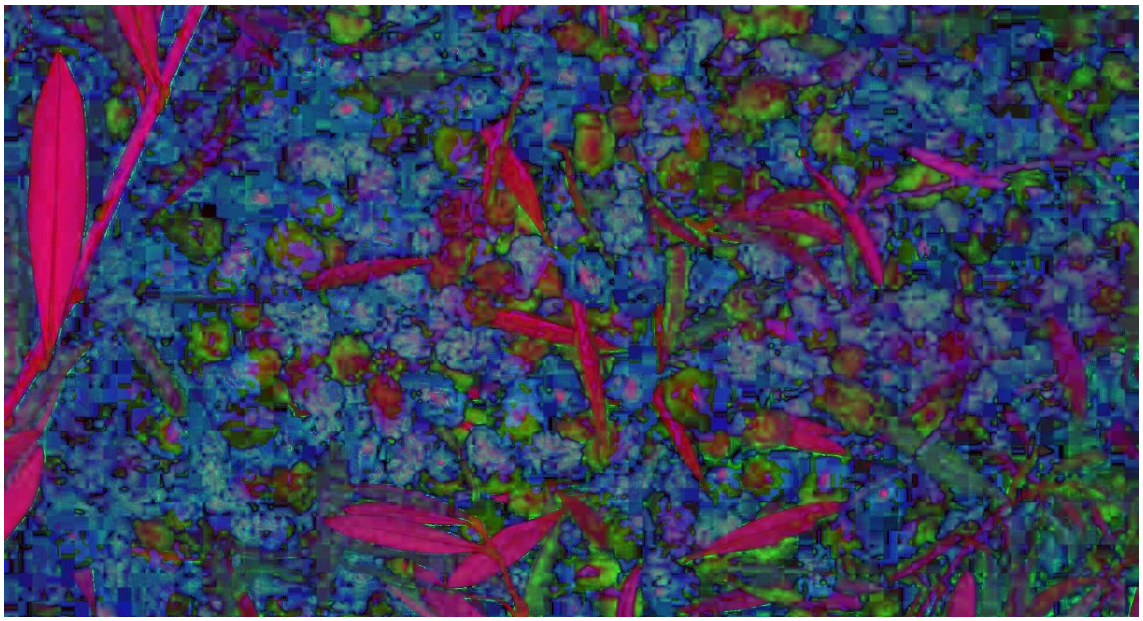

**Imagen 14. Imagen en formato HSV**

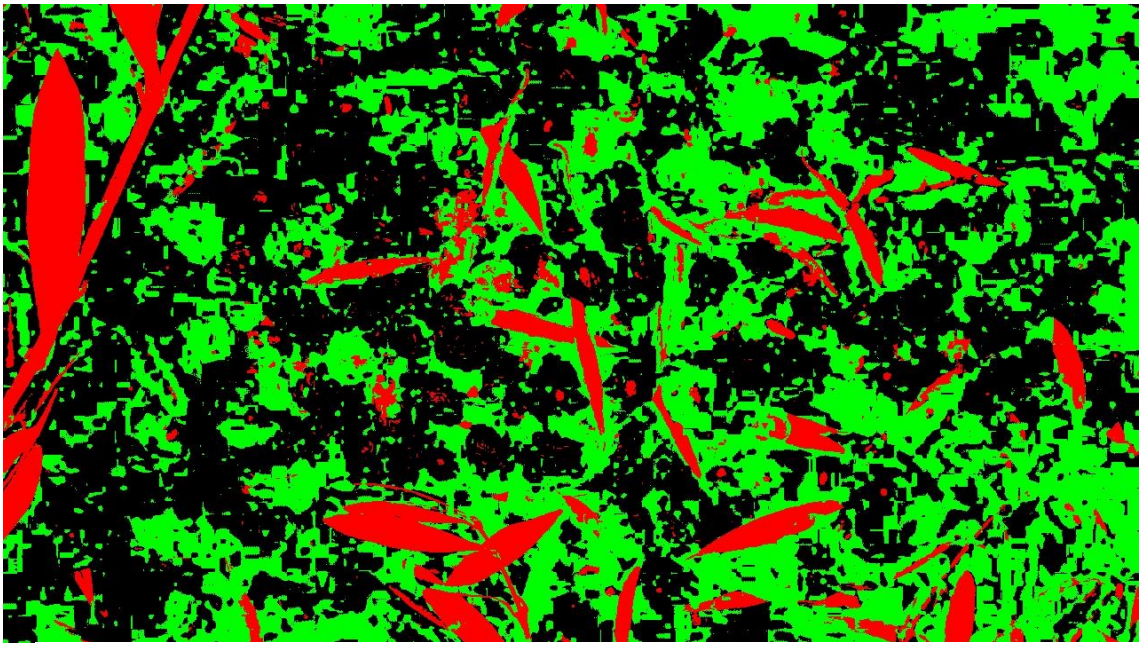

**Imagen 15. Imagen clasificada mediante K-Means en formato HSV**

Para comprobar el canal en el que se ha generado esta distorsión, se ha representado individualmente cada canal del modelo. El defecto aparece en el canal *Hue*, como puede apreciarse en las imágenes.

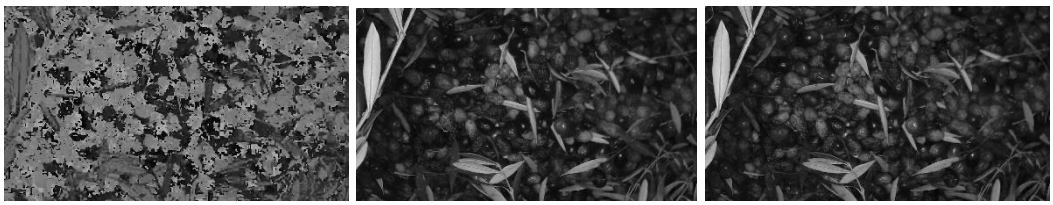

**Imagen 16. Canales del modelo HSV**

### 4.2.3. CIELAB

Este formato se consigue en *OpenCV* de la siguiente manera:

**cv2.cvtColor(***Imagen***,** *Función***)**

*Función*: **cv2.COLOR_BGR2Lab** (cambia el formato de color de BGR a Lab)

A continuación, se puede ver un ejemplo de una de las imágenes con las que se ha realizado este Trabajo de Fin de Máster en formato CIELAB.

**Imagen 17. Imagen en formato CIELAB**

La clasificación con esta opción ha sido con la que mejores resultados se han obtenido. Para ejemplificar este hecho, se muestra la imagen 16, de donde se han extraído los datos mediante *pandas* para generar la clasificación general en las demás imágenes. La clasificación se ha realizado con *K-Means*, seleccionando 10 clases distintas y agrupándolas en las dos familias principales (hoja y aceituna) de manera manual, para conseguir los valores buscados.

El resultado de esta clasificación se puede ver en la imagen 18.

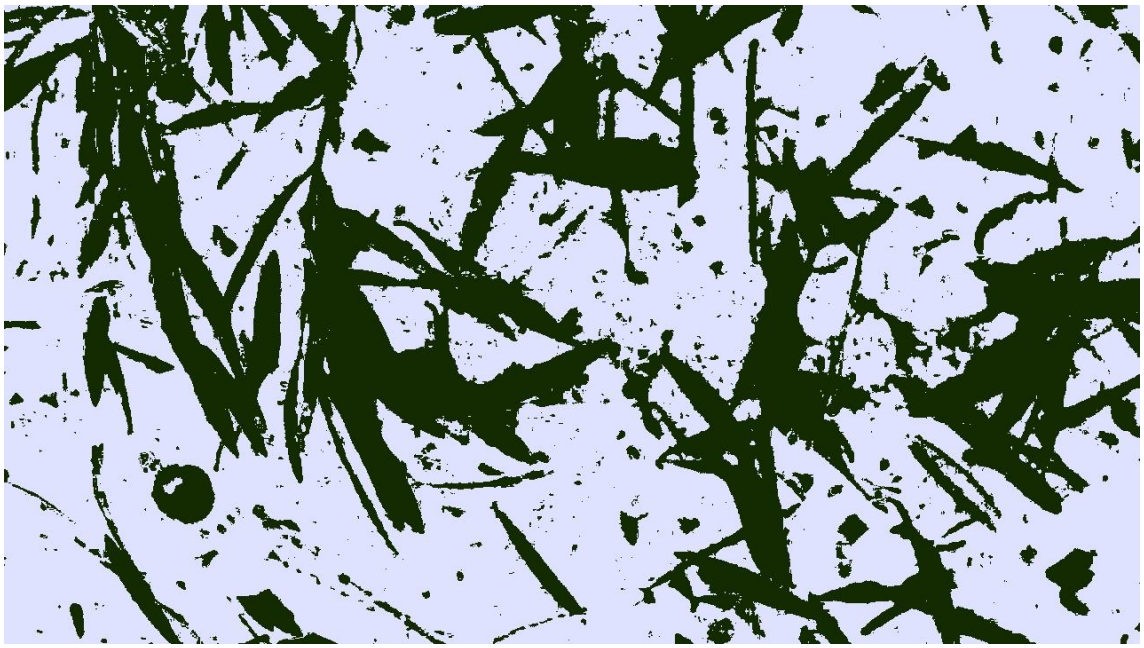

**Imagen 18. Clasificación K-Means con formato CIELAB**

## 4.3. Detección de la hoja

Una vez que se ha seleccionado CIELAB como modelo representativo del color, el siguiente paso ha sido guardar los datos de cada una de las clases en Excel con ayuda de *pandas.* Para ello, se ha realizado un proceso de umbralización *y* se ha aplicado al resto de imágenes para corroborar la efectividad del programa.

A continuación se puede ver un diagrama de flujo que resume el funcionamiento de este fragmento de código.

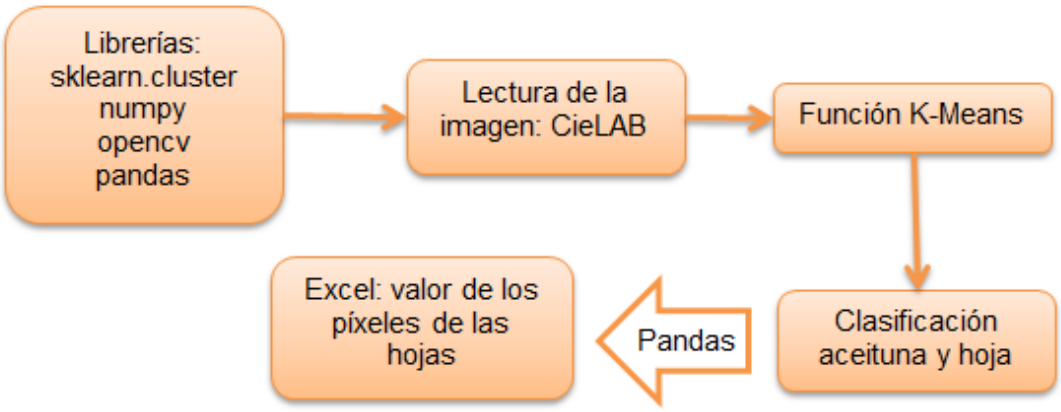


**Imagen 19. Diagrama de flujo: clasificación K-Means**

Una vez que se ha guardado en Excel los valores prototipos de hojas, se extraen los valores máximos y mínimos de la lista de valores de píxeles de hoja y se genera una máscara con la siguiente función:

Mascara = **cv2.inRange(***Imagen***,** *valor máximo, valor mínimo***)**

Esta función genera imágenes binarias en las que selecciona en color blanco los valores encontrados entre el máximo y el mínimo facilitado.

A continuación, se tienen algunos ejemplos de este paso.

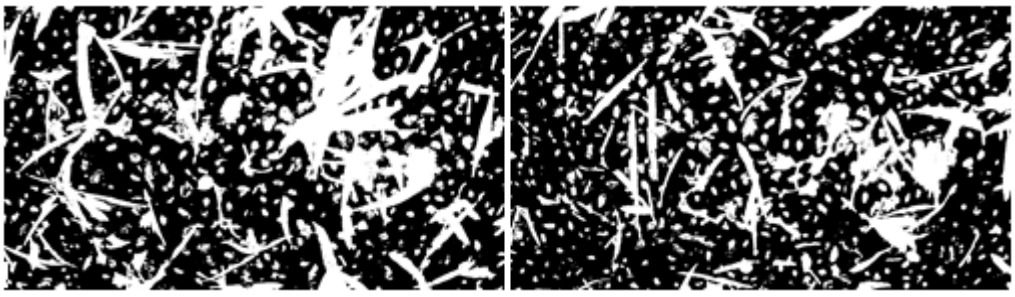

**Imagen 20. Imágenes binarias obtenidas de la máscara**

Como el proceso de umbralización facilita una imagen binaria, se ha empleado dicha imagen para realizar un proceso de detección de contorno. Antes de dibujar el contorno se han eliminado los contornos de áreas más

pequeñas para evitar que el algoritmo cuente los brillos de las aceitunas detectados por el clasificador.

La función empleada para encontrar contornos permite encontrar los contornos externos e internos de una imagen binaria, lo que facilita la detección de objetos en una imagen.

El resultado de esta función, genera un vector en el que se guardan todos los contornos encontrados. Otra variable de salida es la jerarquía existente entre los contornos encontrados, de manera que a cada contorno se le asocia un índice. Este método permite diferenciar entre contornos externos e internos.

Los contornos externos se llaman padres (*parents*) y los contenidos en los anteriores se llaman hijos (*children*). De manera que puede establecerse relación entre los contornos encontrados, representando esta relación la *Jerarquía* de la función.

Cada contorno tiene su propia información sobre a qué jerarquía pertenece, representándose en *OpenCV* mediante un vector: [Next, Previous, First_Child, Parents].

**Next**: indica el siguiente contorno en el mismo nivel jerárquico.

**Previous**: indica el anterior contorno en el mismo nivel jerárquico.

**First_Child**: indica el primer contorno hijo.

**Parents**: indica el índice del contorno padre.

Para contabilizar el porcentaje de hoja, se ha sumado el área encerrada en cada contorno padre, y se le ha restado el área encerrada en cada contorno hijo, ya que representa a las aceitunas que hay entre las hojas presentes. La *Jerarquía* se obtiene mediante el modelo cv2.RETR_TREE.

Como se puede detectar en las imágenes inferiores, aparecen dos colores diferentes para representar los contornos. El contorno rojo implica que son contornos padres y el contorno verde representa los contornos hijos.

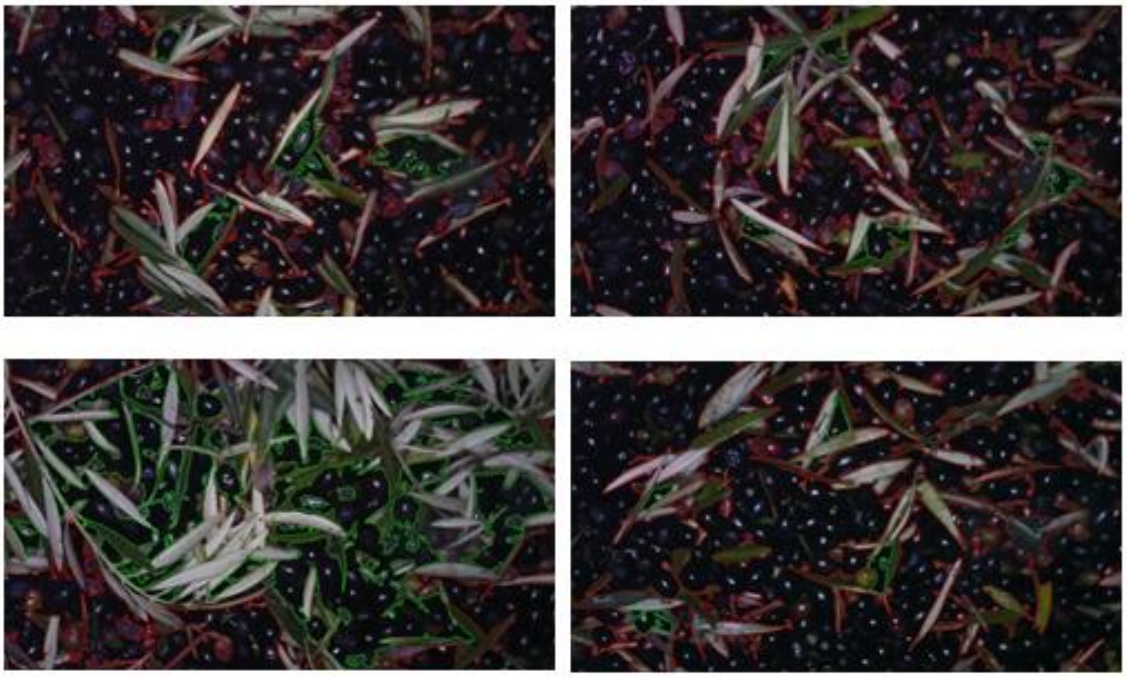

**Imagen 21. Imágenes con contorno en las hojas**

En el código se puede especificar un número en las opciones de dibujo, obteniendo así las imágenes que se muestran arriba, con grosor de línea el número seleccionado. Sin embargo, si se coloca un -1, tal y como está el código ejemplo, se obtienen los contornos rellenos como se muestran en las imágenes inferiores. A la izquierda, tenemos las imágenes originales, mientras que a la derecha, tenemos el resultado de cada imagen original después de procesarla como se ha concretado con anterioridad.

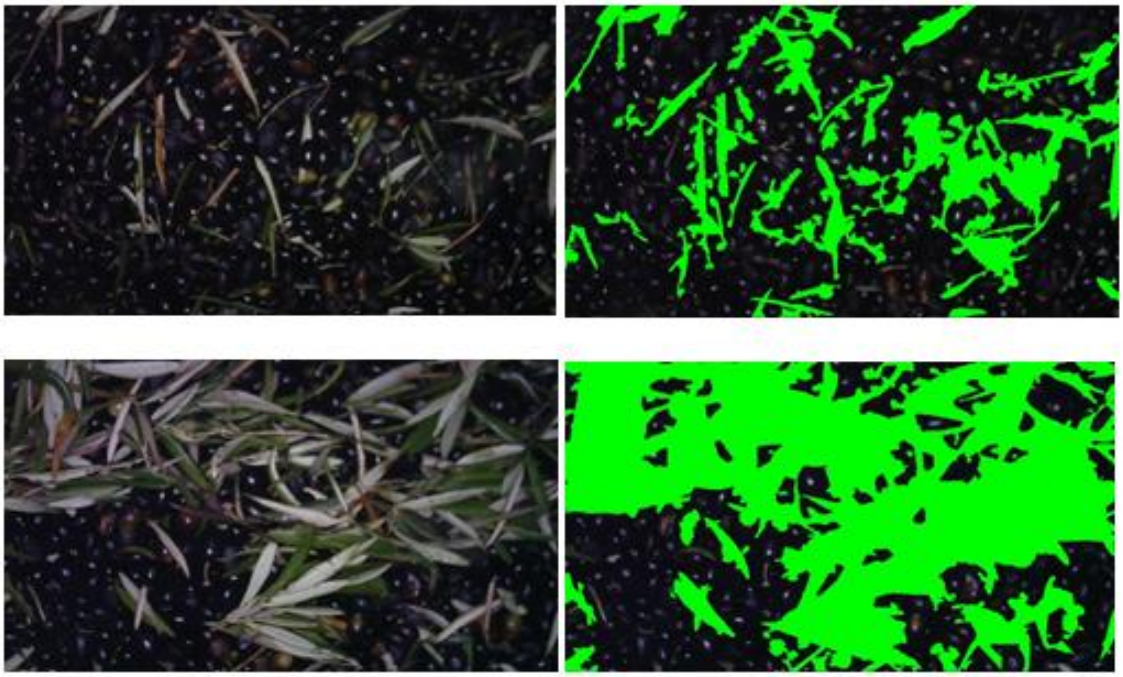

**Imagen 22. Ejemplos de la selección de la hoja**

El porcentaje de hoja presente en la aceituna se ha calculado mediante una función que contabiliza el área total de contorno seleccionado, dividiendo entre el área total de la imagen y multiplicando por cien. Se ha obtenido así un vector que contiene cada porcentaje de hoja en todas las imágenes.

Con el siguiente diagrama de flujo se puede resumir la función **seleccion_hojas** que se adjunta en el Anexo:

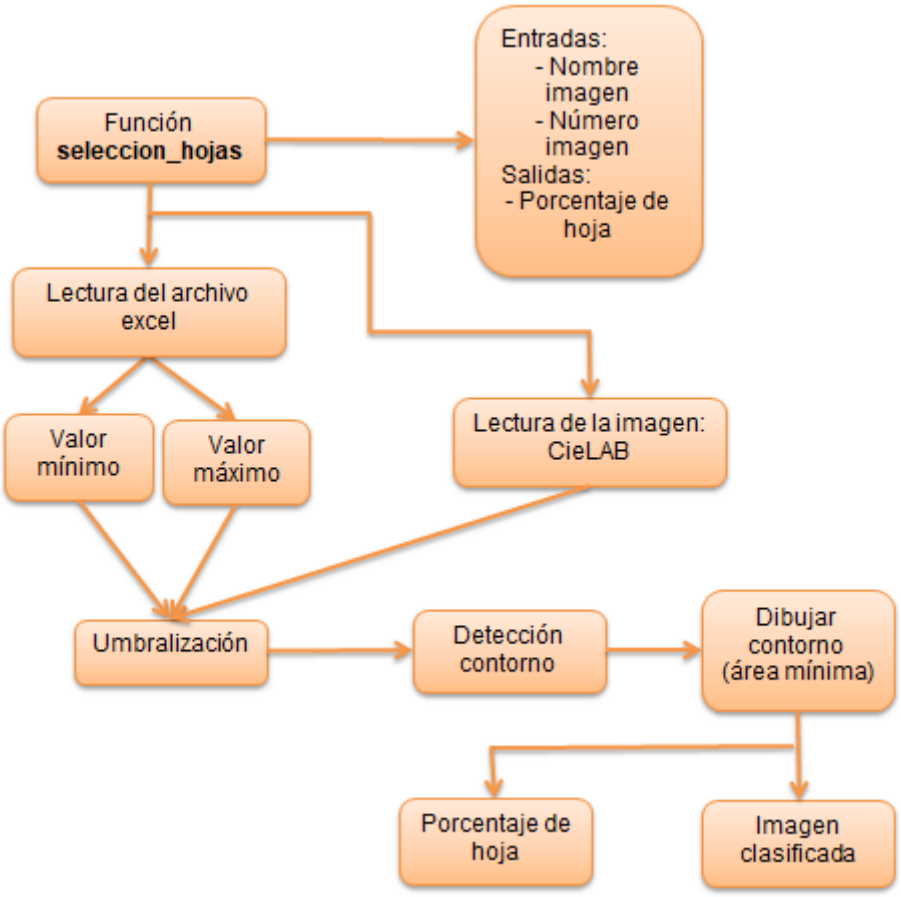


**Imagen 23. Diagrama de flujo: selección hojas**

## 4.4. Detección de la aceituna

### 4.4.1. Detección de centros aceituna

Para la obtención de cada aceituna de manera individual se va a probar el método de *Chan-Vese* (active contours without edges).

Para la implementación de este algoritmo se va a requerir el uso de otra biblioteca de procesado de imagen existente en *Python* (*SCIKIT*) ya que *OpenCV* no cuenta con la implementación del modelo.

Para poder trabajar con este método, primero es necesario activar la librería:

```
from skimage.segmentation import chan_vese
```

Este algoritmo tiene como finalidad segmentar objetos sin límites claramente definidos, ya que es un modelo que trabaja con la energía de la imagen, no requiere el uso de gradientes para la detección de bordes. (Dr. Vicente Candela Pomares, 2014)

Para detectar las semillas de cada uno de los objetos y trabajar con áreas más reducidas en la imagen, se ha umbralizado la imagen para obtener los puntos brillantes que causa el sistema de iluminación y la forma de la aceituna en el centro de esta. Como resultado de esta umbralización, se ha obtenido una máscara como la que se muestra a continuación en la Imagen 24.

**Imagen 24 - Máscara resultante (derecha) de la umbralización de la imagen original (izquierda)**

Sin embargo, la imagen original no es la que se ha umbralizado para conseguir la máscara que ejemplifica esta parte del código, sino la obtenida en el paso anterior, que es en el proceso de eliminación de hojas (Imagen 20).

Para encontrar los centros de cada área blanca en la máscara se va a recurrir al cálculo de momentos. Para ello, se ha aplicado la función para detectar contornos para poder obtener los momentos en cada contorno.

Los momentos son una medida capaz de especificar la dispersión de una nube de puntos, donde $(x, y)$ son las coordenadas de un píxel y la función $I(x, y)$ representa su valor de intensidad. Se calcula de la siguiente manera:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

Como la imagen con la que se está trabajando es una máscara binaria, la intensidad de la imagen solo podrá tener dos valores (0,255).

Para calcular el centroide de cada área blanca, se requiere obtener tres momentos específicos: M00, M01 y M10, que sustituyendo en la Ecuación 1:

$$M_{00} = \sum_x \sum_y x^0 y^0 I(x,y) = \sum_x \sum_y I(x,y)$$

$$M_{10} = \sum_x \sum_y x^1 y^0 I(x,y) = \sum_x \sum_y x\, I(x,y)$$

$$M_{01} = \sum_x \sum_y x^0 y^1 I(x,y) = \sum_x \sum_y y\, I(x,y)$$

El momento M00 aporta información sobre la localización de cada área blanca de la máscara. Los momentos M10 y M01 se van a emplear para calcular las coordenadas del centroide de cada área blanca de la máscara, de la siguiente manera:

$$\frac{M_{10}}{M_{00}} = \frac{\sum_x \sum_y x\, I(x,y)}{\sum_x \sum_y I(x,y)} = \bar{x}$$

$$\frac{M_{01}}{M_{00}} = \frac{\sum_x \sum_y y\, I(x,y)}{\sum_x \sum_y I(x,y)} = \bar{y}$$

De esta forma, ya se tienen localizadas las coordenadas centrales de cada área, por lo que se tiene la posición de cada aceituna. (Robologs, 2019)

### 4.4.2. Obtención de sub-imágenes

Antes de la implementación de este modelo se va a realizar una subdivisión de cada imagen para poder trabajar con menos área e información y obtener mejores resultados.

Sin embargo, si se seleccionan todos centros de la imagen, se obtendrán sub-imágenes duplicadas, ya que algunos de ellos están muy cerca unos de otros. Para evitar este hecho, que conduciría a conclusiones erróneas y poco concluyentes tras el procesado del programa, se ha hecho una lista guardando únicamente los centros que no solapen entre ellos. Esto se consigue, calculando la mínima distancia Euclídea existente entre los centros, guardándose únicamente aquellos valores que sean superiores a un número representativo para eliminar los duplicados.

Para implementar el modelo Chan-Vese, es necesario tener imágenes en escala de grises, ya que la función solo está implementada para una imagen en 2D. Por este motivo, como se apreciará a continuación, se obtienen las sub-imágenes en un modelo representativo del color (RGB) y en escala de grises.

Esta parte del código, presente en el Anexo, se ha resumido en la función **todas_sub_imagenes** y se puede aclarar con el siguiente diagrama de flujo:

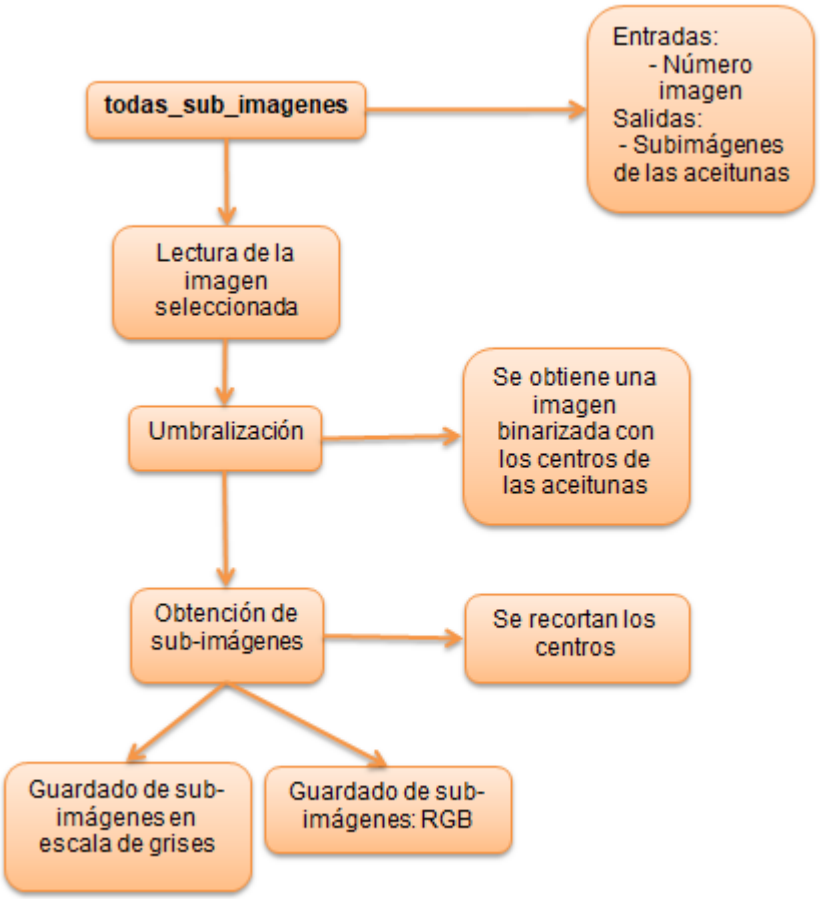


**Imagen 25. Diagrama de flujo: obtención sub-imágenes**

### 4.4.3. Aplicación del modelo Chan-Vese

Cuando se han obtenido las sub-imágenes, se aplica el modelo Chan-Vese a cada una de ellas.

La función implementada en Python en la librería *skimage* tiene la siguiente forma:

```
cv = chan_vese(im, mu, lambda1, lambda2, tol, max_iter, dt, init_level_set = "checkerboard", extended_output)
```

Los parámetros de entrada son:

- **mu**: parámetro 'longitud del borde' (los valores más altos producen un borde redondeado, mientras que los más cercanos a cero detectan objetos más pequeños).
- **lambda1**: parámetro 'variación del promedio' para la región cuya salida tome el valor 'Verdadero'.
- **lambda2**: parámetro 'variación del promedio' para la región cuya salida tome el valor 'Falso'.
- **tol**: tolerancia aceptable establecida entre iteraciones.
- **max_iter**: número máximo de iteraciones permitidas.
- **dt**: un factor de multiplicación que se aplica a los cálculos para acelerar el algoritmo.
- **init_level_set**: define el conjunto de niveles de inicio utilizado por el algoritmo. Los valores de cadena aceptados son: '**checkerboard**' '**disk**', '**small disk**'.
- **extended_output**: True: devuelve una tupla con tres valores de retorno / False: solo devuelve la matriz de segmentación (valor por defecto)

Los parámetros de salida son:

- **Segmentación**: resultado de la segmentación
- **phi**: conjunto de niveles calculado por el algoritmo
- **Lista de energías**: evolución de la 'energía' para cada iteración

La librería ***os*** permite comprobar la existencia de archivos en la ruta seleccionada. En este caso, para evitar un error del código, debe comprobarse primero la existencia de archivos (para leer únicamente las sub-imágenes existentes generadas en el paso previo).

La segmentación que devuelve este proceso es una matriz cuyos objetos detectados aparecen con el valor 'True' y el resto aparece como 'False'. (Dr. Vicente Candela Pomares, 2014)

Para poder trabajar con estos valores, se van a procesar para asignarle los valores blanco o negro y formar una máscara binaria. Después, para conseguir mejores resultados, se procesará mediante morfología de la imagen con procesos de erosión y dilatación, con elementos estructurantes acordes al tamaño de la aceituna. Posteriormente, se aplica un proceso de detección de contorno donde se eliminan las áreas obtenidas que sean menores al tamaño de una aceituna.

El resultado de este proceso se puede comprobar con las imágenes inferiores:

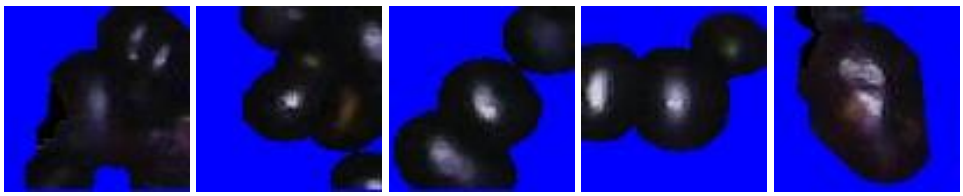

**Imagen 26. Muestra de sub-imágenes tras el proceso de Chan-Vese**

Como puede apreciarse en las imágenes, no ha sido posible detectar una sola aceituna en cada sub-imagen debido a la variabilidad de su forma y su tamaño. Por ejemplo, la sub-imagen de la derecha la ocupa una sola aceituna, mientras que en los demás casos, tenemos tres o más elementos. No es posible generalizar el tamaño, por lo que para dar información sobre el porcentaje resultante, se recurrirá al área detectada por el modelo.

## 4.5. Clasificación de la aceituna

Hay que recordar que uno de los objetivos principales de este Trabajo de Fin de Máster es la clasificación de la aceituna, este punto es el más importante y el concluyente de este documento.

Para clasificar la aceituna, hay que tener en cuenta tanto el color como la rugosidad de ésta. El color marrón implica que la aceituna es del suelo, mientras que la rugosidad se puede deber a dos causas, a que la aceituna es

del suelo, o a la falta de lluvia en la temporada. En este caso, se tomará como 'aceituna estropeada' la que presente clase rugosa.

Para evitar que se dupliquen las sub-imágenes que detectamos por ambos caminos, se incluye una condición en la que primero se detecten las del suelo, que es la opción más fiable, y las 'estropeadas' se sacarán de aquellas que no se han procesado como 'suelo'.

Como se puede apreciar en el fragmento de código superior, todo está ligado a un condicional inicial que verifica que la media de la sub-imagen es inferior a 200 en su canal azul. Esto se debe a que se han eliminado algunas sub-imágenes que el programa detectaba y aportaban poca información por la falta de área trabajable.

Algunos de los ejemplos eliminados son como los que se muestran en la imagen 27.

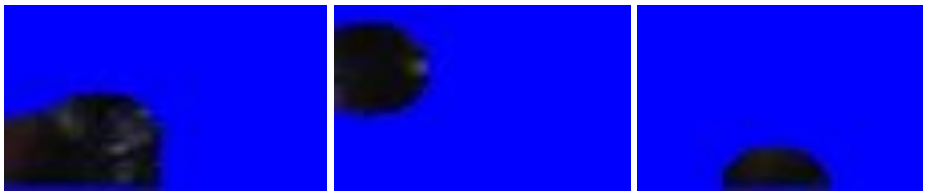

**Imagen 27. Sub-imágenes eliminadas en el proceso**

Para ejemplificar este código, tenemos una serie de sub-imágenes, tanto las detectadas como tipo suelo, como las detectadas como el tipo rugosas.

- Aceitunas suelo:

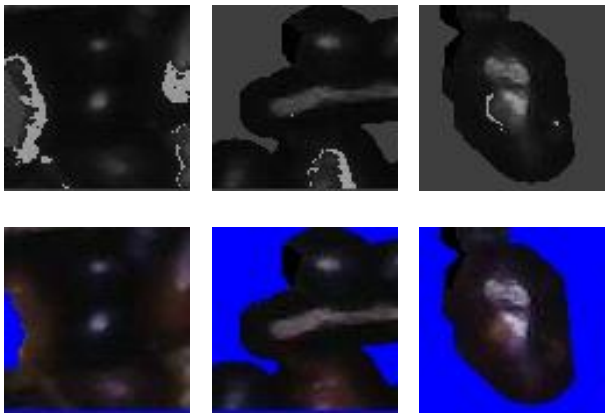

**Imagen 28. Imágenes que contienen aceitunas del suelo**

- Aceitunas rugosas:

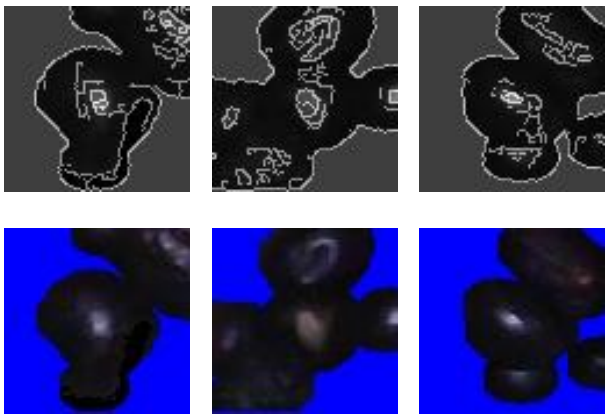

**Imagen 29. Imágenes que contienen aceitunas rugosas**

La función que se ha implementado para obtener las imágenes procesadas por el algoritmo se ha denominado **metodo_chan_vese**. A continuación se muestra un diagrama de flujo que explica su funcionamiento:

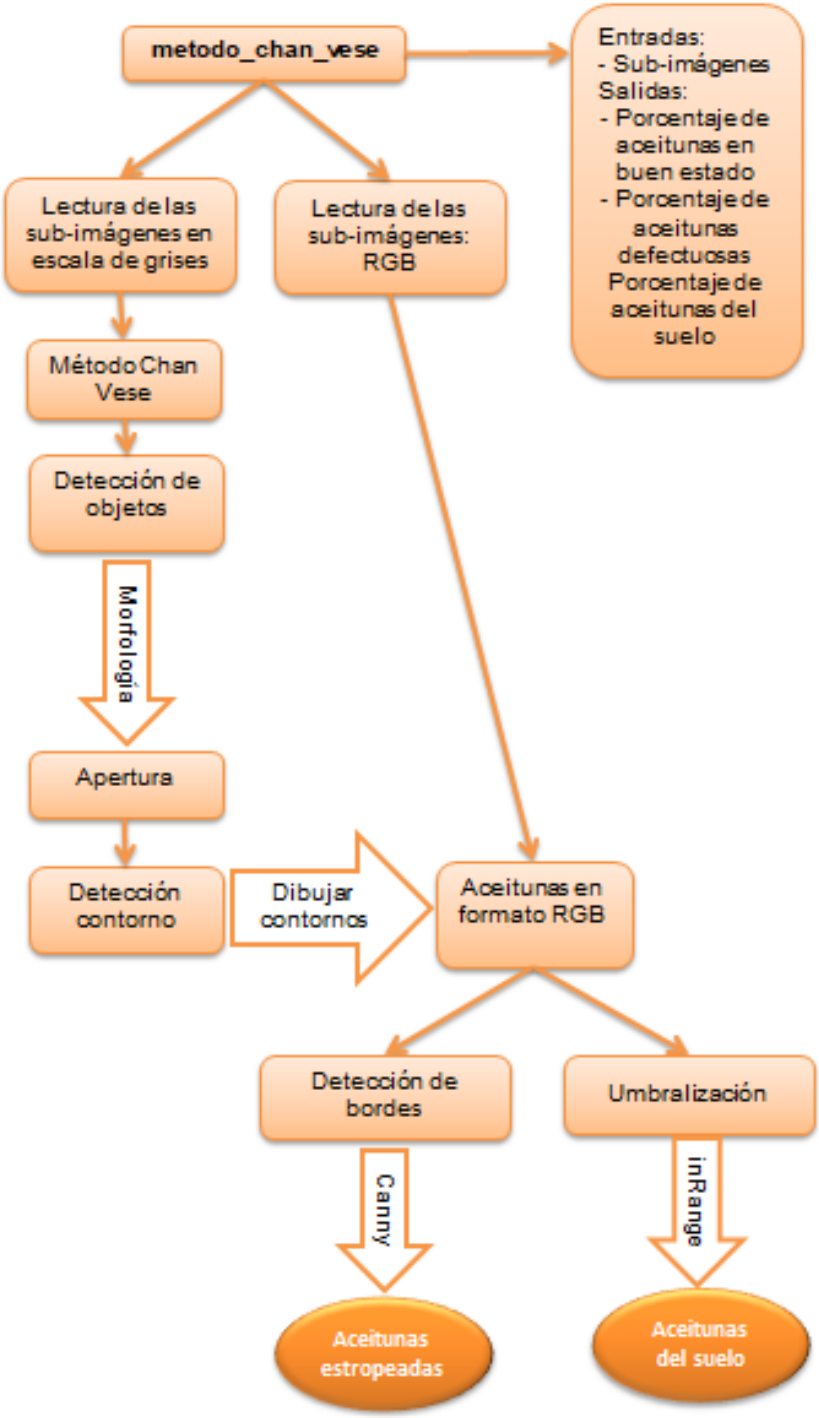


**Imagen 30. Diagrama de flujo: selección tipo aceituna**

# 5. INTERFAZ EN DASH

## 5.1. Introducción del entorno DASH

Dash es una biblioteca de Python de código abierto, lanzada bajo licencia permisiva del MIT. Es una biblioteca de interfaz usuario para crear aplicaciones web analíticas.

Dash está escrito sobre Flask, Plotly.js y React.js, por lo que es una herramienta muy útil para crear aplicaciones de visualización directamente con lenguaje Python y con una gran libertad para personalizar una interfaz usuario.

Cada elemento estético de la aplicación es personalizable: el tamaño, la posición de los elementos, los colores y las fuentes. Las aplicaciones se crean y se publican en la Web.

El código de Dash es funcional, la aplicación puede leer valores del estado global de Python pero no puede modificarlos, por lo que permite entradas y salidas de información sin que se modifique funcionalmente el código. (Dash User Guide, 2019)

Entre las características de Dash, destacan:

- Las aplicaciones Dash son fáciles de usar y mucho más ligeras que un código estándar de Python
- Dash facilita una interfaz sencilla que vincula controles (deslizantes, desplegables y gráficos) de la interfaz de usuario con el código de análisis de datos de Python
- Facilita una aplicación completamente personalizable debido a que cada elemento estético puede modificarse a nivel usuario.

## 5.2. Funciones empleadas

Las aplicaciones de Dash se componen de dos partes, siendo la primera el “layout” de la aplicación, y la segunda la que genera la interactividad de la aplicación.

```
App = dash.Dash()

 App.layout = html.Div()


if __name__ == '__main__':

  App.run_server(port=8050, host= '0.0.0.0')
```

Dash proporciona clases de Python para todos los componentes visuales que pueden emplearse en la aplicación, siendo estas:

```
import dash

import dash_core_components as dcc

import dash_html_components as html

from dash.dependencies import Input, Output
```

El siguiente fragmento de código que se muestra, es lo imprescindible para que funcione el entorno. La primera línea genera la aplicación, la segunda genera el diseño de la aplicación, y por último, se genera el servidor, que es el que activa de manera predeterminada que Dash actualiza automáticamente el navegador al realizar una modificación en el código.

El desarrollador con el que se ha realizado este Trabajo de Fin de Máster, ***Anaconda***, requiere que el servidor se genere exactamente como está especificado en el recuadro superior de la página. Esta aplicación se ejecuta en la dirección: http://127.0.0.1:8050/ en el navegador Web.

El "layout" se compone de un árbol de funciones que dan forma a la aplicación, éstas componentes son diversas y se explicarán las fundamentales, siendo el foco principal las que se han usado para generar esta aplicación.

- **html.Div( )**: este componente es un elemento de división de contenido HTML. Es el contenedor genérico para el contenido de flujo, se usa para agrupar contenido y diseñar de una manera fácil los atributos disponibles a la función.
- **dcc.Graph( id = ‘ ’, figure ={ } )**: este componente sirve para visualizar datos interactivos utilizando la biblioteca de gráficos de plotly.js, siendo el ID obligatorio. Puede emplearse para modificar el gráfico posteriormente. También existe el elemento *figure*, que contiene los datos y el diseño del gráfico.
- **dcc.Input( id = ‘ ‘, value = ‘ ‘, type = ‘ ‘ )**: esta función genera un campo de entrada, se puede acceder a ella mediante su ID, mientras que *value* será el valor por defecto de la entrada. En *type* se especifica el tipo de variable con el que se trabaja.
- **dcc.Dropdown( )**: es un elemento interactivo despegable de selección de uno o más elementos dentro de una lista.
- **dcc.Slider( )**: es un elemento que genera un control deslizante básico que se puede vincular a una devolución de llamada.
- **dcc.Upload( )**: es un componente que permite cargar archivos en la aplicación

Se puede crear una función que genera la modificación en la aplicación en función de las entradas interactivas. Esto se consigue con el siguiente código, especificando cuáles son las entradas y las salidas a modificar. (Unipython, 2018)

```
@App.callback(
  Output(component_id = ‘ ‘, component_property = ‘ ‘),
  [Input(component_id = ‘ ‘, component_property = ‘ ‘)])
```

## 5.3. Interfaz usuario generada

Para generar la interfaz, se han tenido en cuenta los objetivos de este Trabajo de Fin de Máster, que son obtener los resultados de cantidad de hoja está presente en la imagen, así como la clasificación de la aceituna. Por lo tanto, la mejor manera de representar esta información, ha sido mediante un gráfico circular, en el que se representen los porcentajes de cada clase estudiada: hoja, aceituna buena, aceituna del suelo y aceituna estropeada (es la que con anterioridad se ha mencionado como rugosa).

Al cargar la página, se presenta una interfaz como muestra la imagen 31. Tenemos dos pestañas para elegir: selección de una imagen y selección de todas las imágenes disponibles.

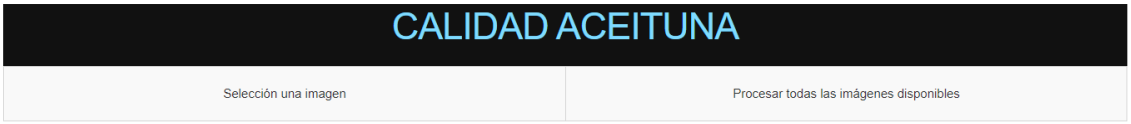


**Imagen 31. Interfaz usuario en entorno Dash**

La primera parte de la interfaz la podemos apreciar en la siguiente imagen (Imagen 32), en la que se puede seleccionar el número de imagen que se quiere procesar. En este caso, solamente se obtiene la información de una sola imagen, la que el usuario escriba en el recuadro, además de mostrar la imagen original y clasificada que se ha seleccionado.

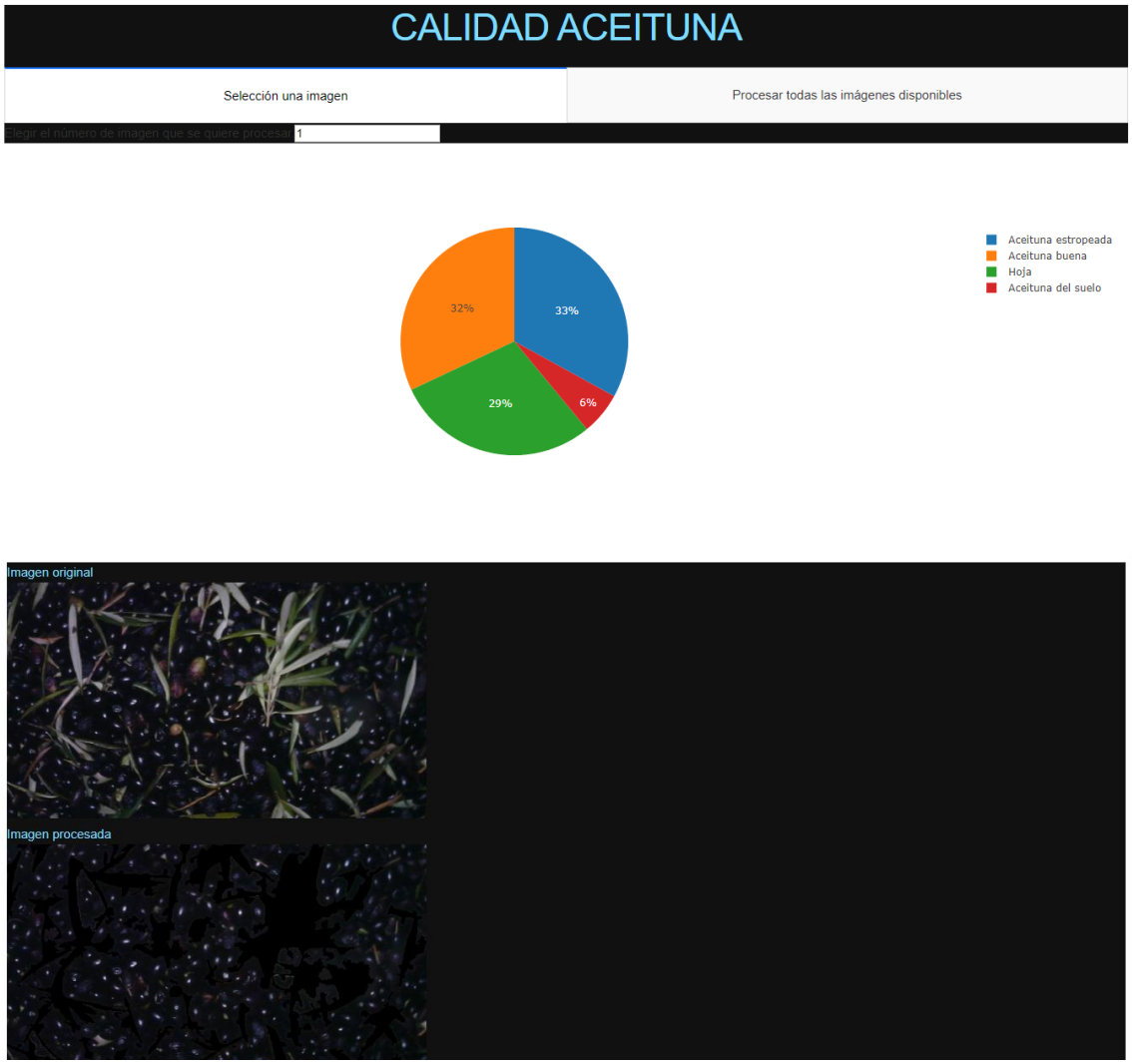


**Imagen 32. Interfaz usuario en entorno Dash**

Por otro lado, la segunda parte de la aplicación, genera información sobre todas las imágenes disponibles en el directorio. Se procesan todas las imágenes y se han extrapolado datos mediante las medias de todos los resultados individuales de cada imagen.

El resultado visualmente es el mismo que en el caso superior, pero con la información de todas las imágenes y sin la posibilidad de interactuar con la aplicación. Esto se puede apreciar en la imagen 33.

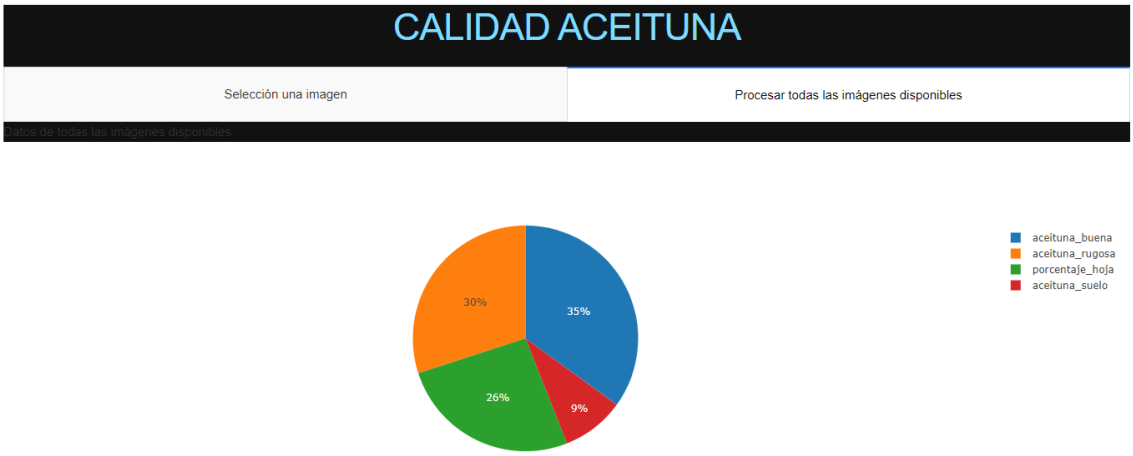


**Imagen 33. Interfaz Dash: clasificación todas imágenes**

# 6. RESULTADOS OBTENIDOS

En los dos últimos apartados del estudio, se han desarrollado los métodos, soluciones y resultados de este Trabajo de Fin de Máster.

En este apartado, se mostrarán más ejemplos de los que se han añadido con anterioridad y otras ramas de estudio que se descartaron por los resultados obtenidos, además de explicar las dificultades encontradas en la consecución de los objetivos.

## 6.1. Caminos de estudio descartados y más ejemplos

La primera opción ha sido separar las hojas de las aceitunas mediante máscaras manuales, pero no se ha conseguido detectar las diferentes tonalidades de éstas sin detectar una cantidad demasiado alta de aceituna en el proceso.

A continuación se aprecia una imagen prototipo, de la que se han obtenido diferentes muestras con varias máscaras. Se aprecia en cada una de ellas la incapacidad de obtener resultados concluyentes como los obtenidos mediante el clasificador *k-means*.

**Imagen 34. Imagen original**

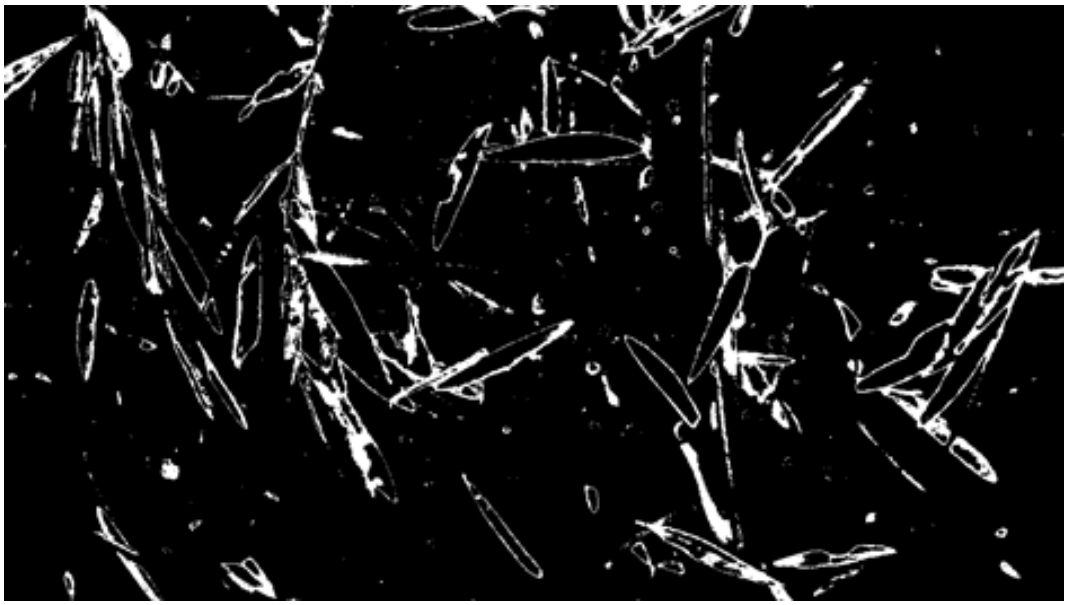

**Imagen 35. Ejemplo nº 1: imagen procesada mediante máscara**

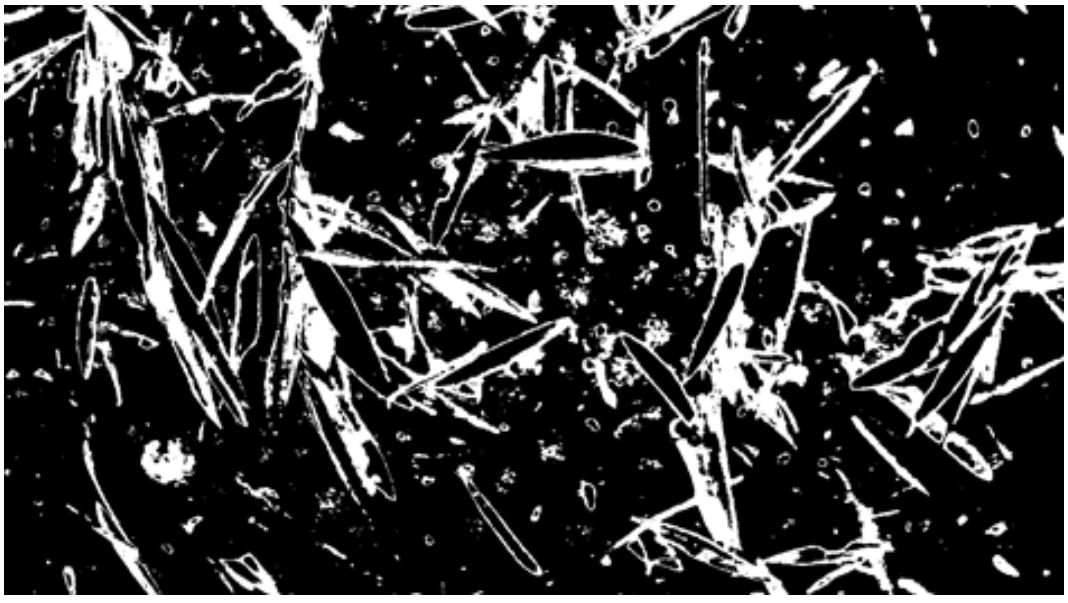

**Imagen 36. Ejemplo nº 2: imagen procesada mediante máscara**

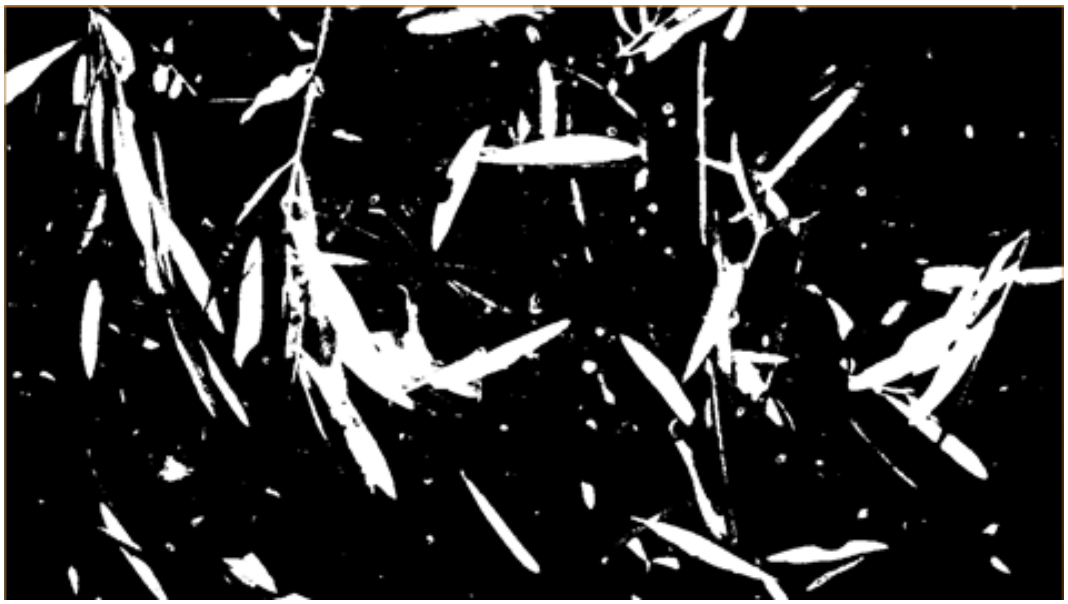

**Imagen 37. Ejemplo nº 3: imagen procesada mediante máscara**

Ahora, se muestra el resultado de esta misma imagen con el proceso *k-means*:

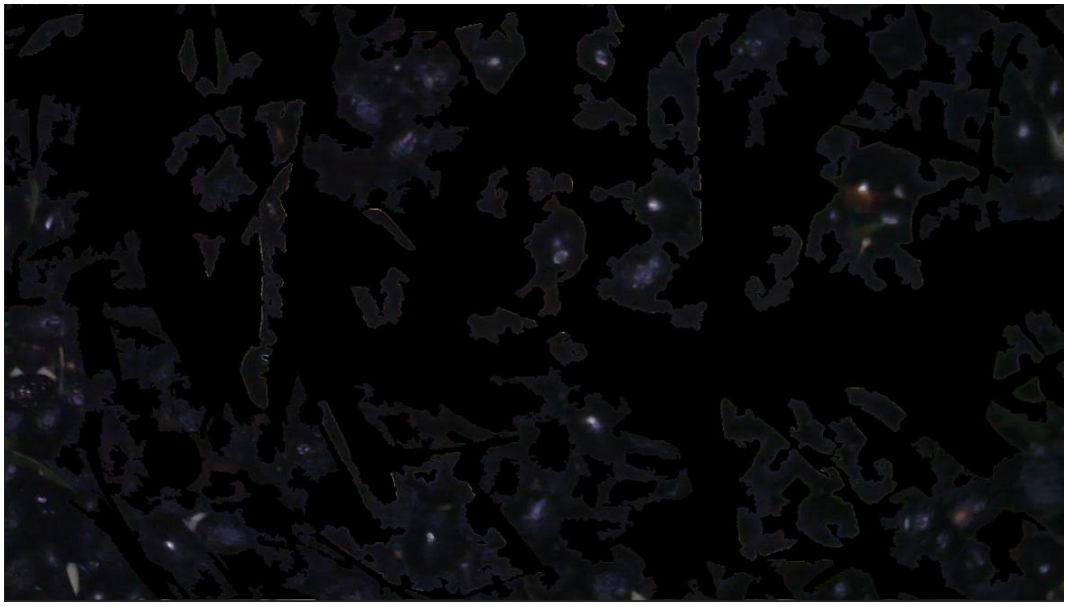

**Imagen 38. Imagen final de la selección de hojas mediante *k-means***

Algunos ejemplos más de este punto del trabajo se pueden apreciar en las siguientes:

**Imagen 39. Ejemplo nº 1: imagen después de clasificador *k-means***

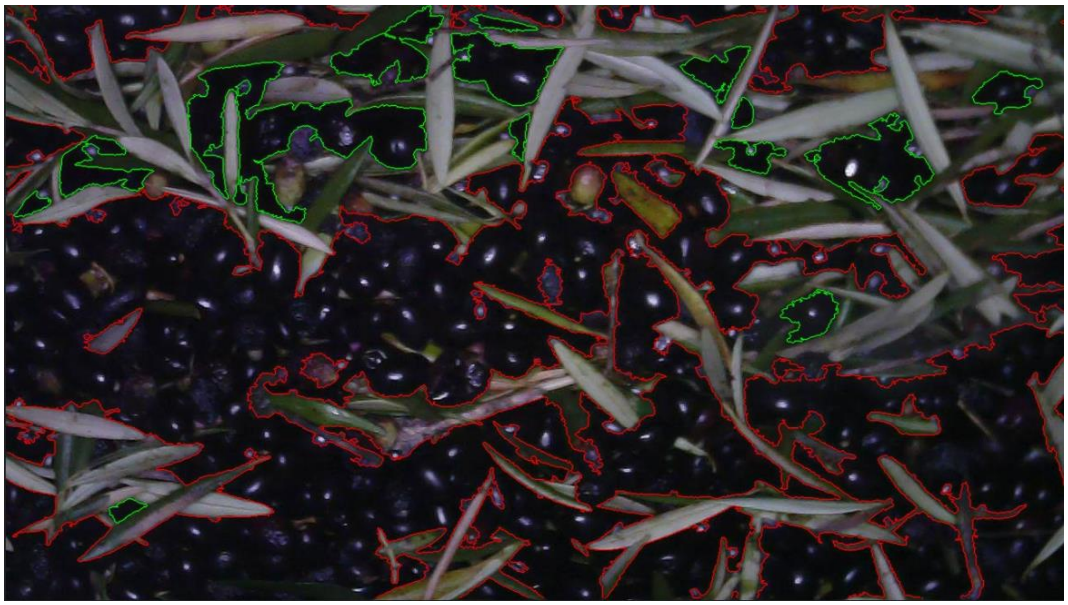

**Imagen 40. Ejemplo nº 2: imagen después de clasificador *k-means***

**Imagen 41. Ejemplo nº 3: imagen después de clasificador *k-means***

Una vez que se ha realizado la clasificación de hojas, se ha intentado realizar la detección de las aceitunas de manera individual mediante morfología de la imagen (apertura y cierre con un elemento estructurante del tamaño medio de la aceituna en la imagen).

Los resultados de estas pruebas se han descartado por la falta de utilidad que representaban las imágenes resultantes. Algunos ejemplos de estos casos se muestran a continuación:

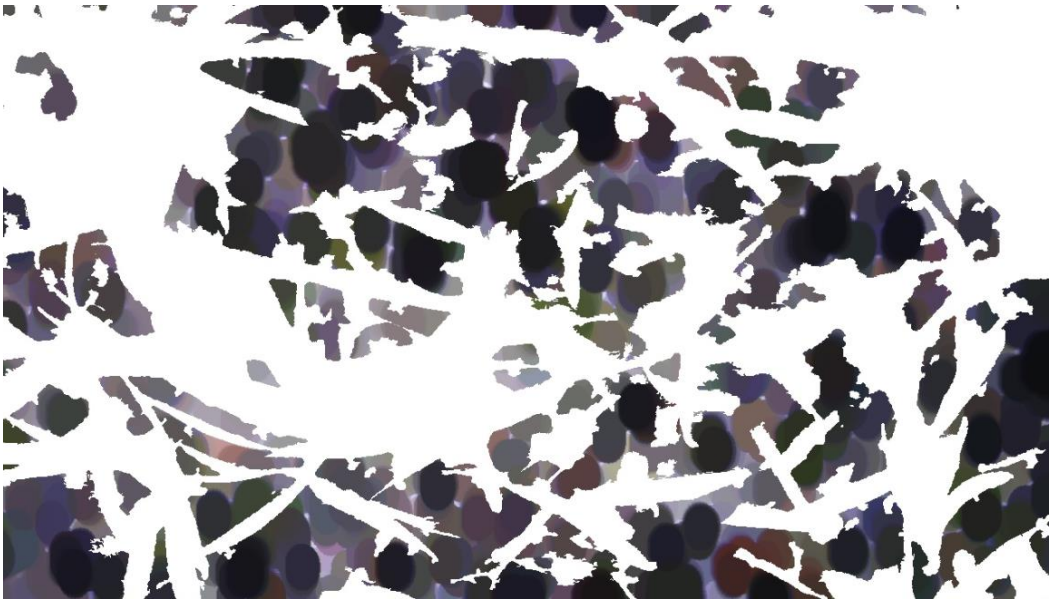

**Imagen 42. Ejemplo de imagen obtenida mediante morfología**

Con este camino no se puede obtener cada aceituna de manera individual que es el siguiente objetivo a estudiar, por este motivo, lo se ha realizado al final ha sido el método de contornos activos sin bordes. Este método ha supuesto tener que cambiar la librería de procesado de imagen objetivo de este Trabajo de Fin de Máster, pero debido a la falta de implementación del método en *OpenCV*, y la dificultad que supone el algoritmo, ese ha sido el siguiente paso a estudiar.

El paso previo a aplicar el método, ha sido obtener cada aceituna de manera individual, ya que el resultado obtenido al procesar toda la imagen ha sido inconcluyente.

Para este paso, se han probado dos caminos, detectar el centro de la aceituna con una máscara, que es el método que se ha aplicado y explicado con detalle, y por otro lado, recortar en sub-imágenes la imagen principal. Sin embargo, el problema que ha presentado esta opción, ha sido que estas sub-imágenes no contienen ninguna aceituna al completo, sino fragmentos de varias.

A pesar de que el primer método ha sido el seleccionado por obtener mejor información desde el punto de vista del procesado de imagen, es cierto que se han perdido muchas aceitunas en el proceso de generar sub-imágenes.

A continuación tenemos algunas sub-imágenes obtenidas por ambos métodos para ejemplificar el texto anterior.

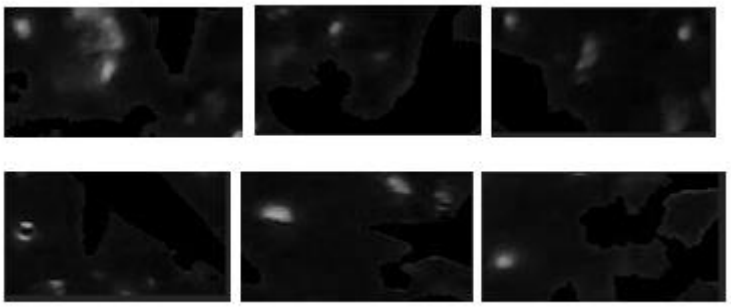

**Imagen 43. Sub-imágenes obtenidas por el segundo método**

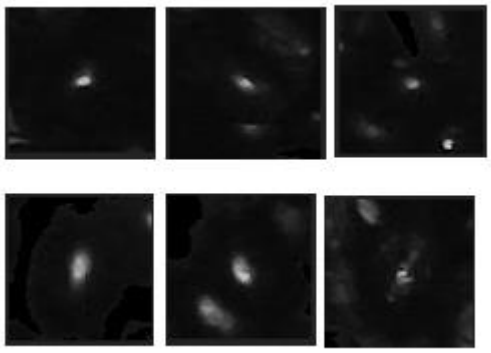

**Imagen 44. Sub-imágenes obtenidas con el brillo de las aceitunas**

Como se puede apreciar en las imágenes, se aprecian mejor las aceitunas si se obtienen las sub-imágenes a partir del brillo de las aceitunas. Sin embargo, como el método de Chan-Vese solo se encuentra implementado para imágenes en blanco y negro, se pierde mucha información en este proceso.

A continuación, se muestran algunos ejemplos de la detección de aceituna después de aplicar el algoritmo. Las siguientes imágenes han sido procesadas para facilitar la visibilidad y compresión del resultado.

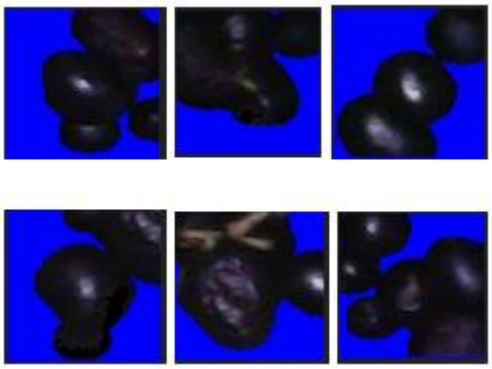

**Imagen 45. Sub-imágenes procesadas mediante el método Chan-Vese**

## 6.2. Dificultades encontradas en el desarrollo

Todos los trabajos de investigación y desarrollo requieren esfuerzo, paciencia y dedicación, sobre todo los que implican introducir herramientas que aún no han sido desarrolladas en su plenitud.

Sin embargo, una de las mayores dificultades encontradas en el desarrollo de este Trabajo de Fin de Máster ha sido la calidad de las imágenes facilitadas para generar el programa y la aplicación.

Este problema es algo fácilmente apreciable en las sub-imágenes que se han mostrado en el apartado anterior en blanco y negro. En esas sub-imágenes, la información que se obtiene a simple vista es muy poca, ya que ni visualmente se consiguen apreciar las aceitunas. Esto se ha intentado solventar sacando en paralelo sub-imágenes en blanco y negro para procesar mediante el algoritmo Chan-Vese, y por otro lado sub-imágenes con un formato representativo de color RGB para no evitar la pérdida de información.

Además de la mala calidad de las imágenes, la librería OpenCV no posee todas las funcionalidades requeridas para la implementación de los dos

métodos más importantes que se han utilizado en este trabajo: el clasificador y el algoritmo de detección de contornos.

En la siguiente imagen, se puede ver un gráfico que se ha obtenido mediante la librería Seaborn (una librería que posee funcionalidades estadísticas) el resultado obtenido mediante el módulo de selección k-means. Se aprecian las dos clases: hoja y aceituna, diferenciadas en el espacio.

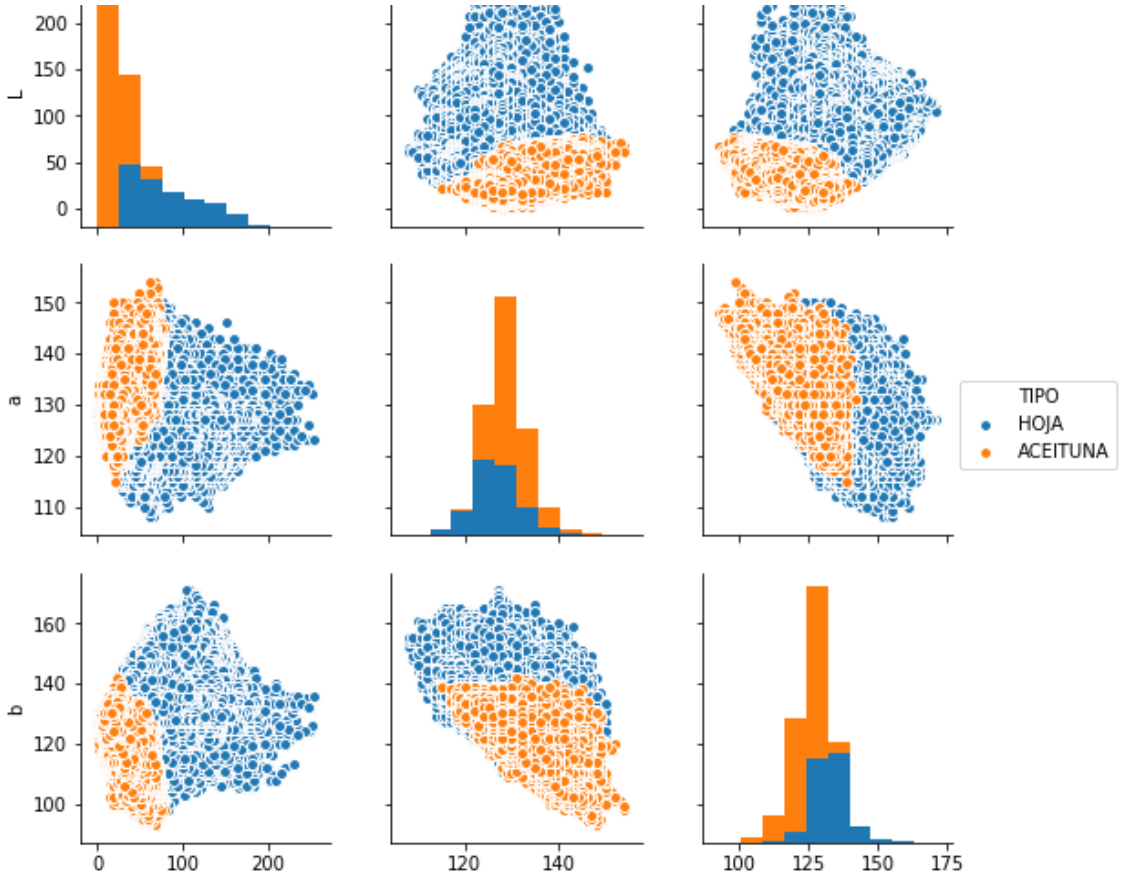


**Imagen 46. Clasificación estadística de las clases**

# 7. CONCLUSIONES Y LÍNEAS DE FUTURO

## 7.1. Cumplimiento de los objetivos principales

Los objetivos principales de este Trabajo de Fin de Máster han consistido en obtener información sobre imágenes tomadas en la cinta de una almazara mientras transportaba aceitunas y hojas. Por un lado, obtener el porcentaje de hoja presente en cada imagen, y por otro, obtener porcentajes de aceituna buena y mala.

A pesar de las dificultades encontradas, los objetivos marcados se han cumplido, aunque quizás no con la exactitud que se pretendía al iniciar el trabajo.

Uno de los mayores puntos de bloqueo que se han encontrado ha sido el sistema de obtención de imágenes. Se aprecia a simple vista la mala calidad de imagen que se obtiene, por lo que una mejora de los resultados debería ser invertir en un buen sistema de captura de imágenes.

Además, el modelo representativo del color HSV, que es uno de los más utilizados y conocidos, no se ha podido emplear en este trabajo debido al formato de imagen y a la concentración de tonalidades que poseen.

Sin embargo, con el método de Chan-Vese se han obtenido unos resultados bastantes buenos a pesar de la poca información que se obtiene de las sub-imágenes debido a que se ha tenido que trabajar en escala de grises. Esto demuestra que los algoritmos implementados son muy potentes y han facilitado la consecución de los objetivos a pesar de las dificultades.

Por último, la aplicación que se ha realizado es muy simple y sencilla, pero resume toda la información obtenida en el desarrollo del trabajo. Se ha realizado con el objetivo de que sea muy sencilla de usar, además de lo más rápida posible. Aunque el tiempo dependerá del número de imágenes que se vayan a procesar en la segunda parte de la aplicación, ya que se van a

procesar todas las que haya en la carpeta donde se encuentre el código de la aplicación.

## 7.2. Líneas de futuro

La línea de investigación que se puede seguir a partir de este trabajo se basa en el procesado de imagen. En este trabajo, se ha conseguido obtener mediante el lenguaje de programación de Python y diversas librerías un programa bastante potente de clasificación de aceituna para una almazara. Sin embargo, la librería principal que se ha empleado para esto ha sido OpenCV, que es una librería que actualmente no se encuentra en su total desarrollo, pues apenas acaba de comenzar a completar sus funcionalidades en este campo de investigación. Seguramente, de aquí a un tiempo, sea muy posible mejorar los resultados obtenidos en este trabajo aplicando diferentes funcionalidades que se añadan al código implementado en la librería OpenCV. Un ejemplo de esto, es la falta del algoritmo de Chan-Vese para encontrar los contornos activos de la aceituna en esta librería.

Por otro lado, el hecho de limitar este trabajo al uso casi exclusivo de la librería de OpenCV, ha sido un obstáculo para su crecimiento, pudiendo ser complementado en el futuro con otras librerías de procesado de imagen disponibles en Python. La principal opción que se encuentra disponible y que se ha requerido incluso para realizar una parte de este trabajado ha sido la librería: scikit-image. Una combinación de ambas librerías supondría un adelanto impresionante para una consecución más precisa de los objetivos, dada la gran variedad que ambas librerías ofrecen.

Por último, se puede conseguir un gran progreso en la aplicación que se ha realizado, obteniendo mucha más información de ella actualizando la interfaz para mejorar su interactividad con el usuario.

En conclusión, este Trabajo de Fin de Máster queda como un punto y seguido de diversas opciones y oportunidades de investigación de un campo en el que aún queda mucho por desarrollar.

# 8. BIBLIOGRAFÍA

Alberto, D. C. (Septiembre de 2013). Entorno de programación gráfico OpenCV. Universidad Carlos III Madrid.

*Dash User Guide*. (Septiembre de 2019). Recuperado el 4 de Diciembre de 2019, de https://dash.plot.ly//

Dr. Vicente Candela Pomares, D. V. (Julio de 2014). Método de detección de contornos: implementación dinámica del modelo de Chan-Vese y detección de contornos basado en multirresolución.

E. Guzmán, V. B.-M. (2013). Infrared machine vision system for the automatic detections of olive fruit quality. Talanta.

*Entendiendo el espacio de color CieLAB*. (Septiembre de 2014). Recuperado el 13 de Octubre de 2019, de http://sensing.konicaminolta.com.mx/2014/09/entendiendo-el-espacio-de-color-cie-lab/

*Esencia de olivo*. (18 de Noviembre de 2011). Recuperado el 6 de Agosto de 2019, de http://www.esenciadeolivo.es/aceite-de-oliva/aceite-de-jaen/

Guerrero, R. (2015). *Teoría del color*. Recuperado el 7 de Septiembre de 2019, de http://dirinfo.unsl.edu.ar/servicios/abm/assets/uploads/materiales/a68e3-03_color_15.pdf

Irene Gómez Moreno, C. G. (s.f.). Algoritmo de aprendizaje: KNN & KMEANS. Universidad Carlos III Madrid.

Puerto, D. A. (2019). Online system for the identification and classification of olive fruits for the olive oil production process. *Journal of Food Measurement and Characterization*, 12.

R. Diaz, L. G. (2004). Comparison of three algorithms in the classification of table olives by means of computer vision. *J. Food Eng*, 101-107.

R. Diaz, L. G. (2004). Comparison of three algorithms in the classification of table olives by means of computer vision. *J. Food Eng. 61*, 101-107.

R. Furferi, M. C. (2010). A machine vision system for real-time and automatic assessment of olives colour and surface defects. *Int. J. Comput. Res.*

*Robologs*. (24 de Junio de 2019). Recuperado el 3 de Noviembre de 2019, de https://robologs.net/2019/06/24/buscar-el-centro-de-un-contorno-con-opencv-python/

SaharZafari, T. E. (12 de Diciembre de 2015). Segmentation of Overlapping Elliptical Objects in Silhouette Images.

*Scikit-image*. (s.f.). Recuperado el 16 de Noviembre de 2019, de https://scikit-image.org/docs/dev/api/skimage.segmentation.html#skimage.segmentation.chan_vese

T. Ram, Z. W. (2010). Olive oil content prediction models based on image processing. *Biosyst Eng.*, 210-232.

Tony F. Chan, L. A. (2 de Febrero de 2001). Active Contours Without Edges.

*Unipython*. (26 de Marzo de 2018). Recuperado el 4 de Diciembre de 2019

Waskom, M. (2012). *Seaborn*. Recuperado el 29 de Septiembre de 2020, de https://seaborn.pydata.org/

# 1. ANEXOS

En este apartado se adjuntan los códigos en Python realizados en el desarrollo de este trabajo de Fin de Máster, así como una Demo de la aplicación generada en el entorno Dash by plotly.

## 2.1. Código Fuente

En este apartado se agrupan las funciones generadas que resumen el desarrollo del proyecto.

Clasificación hoja mediante K-Means de imagen muestra:

```
import numpy as np
import cv2
from sklearn.cluster import KMeans
import pandas as pd

im = cv2.imread('aceitunas0.jpg', cv2.IMREAD_COLOR)
im = cv2.cvtColor(im, cv2.COLOR_BGR2Lab)

[f,c,p] = im.shape
im_ = cv2.imread('aceitunas0.jpg', cv2.IMREAD_COLOR)
m = f*c
matriz = np.reshape(im,(m, p))
clases = 9
kmeans1 = KMeans(n_clusters = clases , init = 'k-means++' , n_init = 10 , max_iter = 300).fit(matriz)
r = kmeans1.labels_
r_ = np.reshape(r,(f,c))

vector_hoja = []
i = 0
vector_aceituna = []
ii = 0

for j in list(range(f)):
   for l in list(range(c)):
      if r_[j,l] == r_[18,0] or r_[j,l] == r_[370,100] or r_[j,l] == r_[100,20] or r_[j,l] == r_[10,10] or r_[j,l] == r_[663,153] or r_[j,l] == r_[200,145]: #hoja
         im[j,l,:] = [255, 42 , 20] #azul
         vector_hoja.append([])
         vector_hoja[i] = im[j,l,:]
         i += 1
      if r_[j,l] == r_[150,20] or r_[j,l] == r_[185,65] or r_[j,l] == r_[1,1]: #aceituna
         im[j,l,:] = [0,226,223] #amarillo
         vector_aceituna.append([])
         vector_aceituna[ii] = im[j,l,:]
         ii += 1

excel = pd.DataFrame(vector_hoja, columns = ['L','a','b'])
excel.to_excel('vectorhojaLab.xlsx', sheet_name = 'vectorhojaLab' )
excel = pd.DataFrame(vector_aceituna, columns = ['L','a','b'])
excel.to_excel('vectoraceitunaLab.xlsx', sheet_name = 'vectoraceitunaLab' )
```

Función **seleccion_hojas**:

```
def seleccion_hojas(directorio, num = 1):
  import cv2
  import numpy as np
  import pandas as pd
  df = pd.read_excel('vectorhojaLab.xlsx')
  L = np.array(df['L'])
  a = np.array(df['a'])
  b = np.array(df['b'])
  min_mascara = np.array([min(L), min(a), min(b)])
  max_mascara = np.array([max(L), max(a), max(b)])
  im = cv2.imread(directorio, cv2.IMREAD_COLOR)
  im_ = cv2.cvtColor(im, cv2.COLOR_BGR2Lab)
  [f,c,p] = im.shape
  m = f*c

  mascara = cv2.inRange(im_, min_mascara, max_mascara)

  vec = []
  area = 0
  i = 0
  s = 0
  contorno,jerarquia = cv2.findContours(mascara, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
  for co in contorno:
    ar = cv2.contourArea(co)
    if ar > 600  and jerarquia[0,s,3] != -1:
      vec.append(cv2.contourArea(co))
      area = area - vec[i]
      i += 1
      cv2.drawContours(im_,[co], 0, [255,226,223], -1, cv2.LINE_AA)
    if ar > 1250 and ar < 850000 and jerarquia[0,s,3] == -1:
      vec.append(cv2.contourArea(co))
      area = area + vec[i]
      i += 1
      cv2.drawContours(im_,[co], 0, [0,42,20], -1, cv2.LINE_AA) #sin rellenar
    s += 1


  for j in list(range(f)):
     for h in list(range(c)):
        if np.equal(im_[j, h,:], [0,42,20]).any():
           im[j,h,:] = [0,0,0]
```

```
nombre_archivo = 'mascara{a}.jpg'.format(a = num)
  cv2.imwrite(nombre_archivo, im)

  im__ = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
  mascara1 = cv2.inRange(im__, 60,180)
  nombre_archivo_ = 'mascara_van_chese{nu}.jpg'.format(nu = num)

  cv2.imwrite(nombre_archivo_, mascara1)
  return (porcentaje_hoja)
```

Función **todas_sub_imagenes**:

```
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
from skimage.segmentation import chan_vese
import os

def todas_sub_imagenes(num =200):

 for i in list(range(200)):
  if os.path.isfile('mascara{a}.jpg'.format( a = i)):
   nombre_archivo = 'mascara{a}.jpg'.format(a = num)
   im = cv2.imread(nombre_archivo, cv2.IMREAD_COLOR)
   im_ = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
   [f,c,p] = im.shape

mascara = cv2.inRange(im_, 70,160)

   contorno, _ = cv2.findContours(mascara, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
   i = 0

   Vec = []
   dista = []
   for co in contorno:
   #cv2.drawContours(im_,[co], 0, 155, -1 , cv2.LINE_AA)
    momentos = cv2.moments(co)
    if momentos['m00'] != 0:

      cx = int(momentos['m10']/momentos['m00'])
      cy = int(momentos['m01']/momentos['m00'])

      if i == 0:
       Vec. append([cx,cy])
       i += 1

      else:
       for m in range(len(Vec)):
          dista.append(math.sqrt((cx-Vec[m][0])**2+(cy-Vec[m][1])**2))

                    q = 0
```

```
    for x in range(len(dista)):
          if dista[x] < 60:
             q= 1

    dista = []

    if q == 0:
     if [cx,cy] not in Vec:
      Vec.append([cx,cy])
    i +=1

i = 0

for V in range(len(Vec)):

 if Vec[V][0] >= 40 and Vec[V][0] <= 1240 and Vec[V][1] >= 40 and Vec[V][1] <= 680:
      rec = im_[(Vec[V][1] - 40):(Vec[V][1] + 40), (Vec[V][0] - 40):(Vec[V][0] + 40)]
      rec_ = im[(Vec[V][1] - 40):(Vec[V][1] + 40), (Vec[V][0] - 40):(Vec[V][0] + 40)]

      if np.average(rec) > 14:
        nombre_ar = 'recortes_prueba{a}.jpg'.format(a = i)
        nombre_archivo = 'recortes_color{a}.jpg'.format(a = i)
        i = i+1
        cv2.imwrite(nombre_ar, rec)
        cv2.imwrite(nombre_archivo, rec_)

 if Vec[V][0] < 40 and Vec[V][1] > 40 and Vec[V][1] < 680:
       rec = im_[(Vec[V][1] - 40):(Vec[V][1] + 40), 0:(Vec[V][0] + 40)]
       rec_ = im[(Vec[V][1] - 40):(Vec[V][1] + 40), 0:(Vec[V][0] + 40)]

       if np.average(rec) > 14:
        nombre_ar = 'recortes_prueba{a}.jpg'.format(a = i)
        nombre_archivo = 'recortes_color{a}.jpg'.format(a = i)
        i = i+1

        cv2.imwrite(nombre_ar, rec)
        cv2.imwrite(nombre_archivo, rec_)

 else:
    i +=1
```

Función **metodo_chan_vese**:

```
import cv2
import numpy as np
import os
from skimage.segmentation import chan_vese

def metodo_van_chese():

 suelo = 0
 rugosa = 0
 aceitunas = 0

 for i in list(range(170)):

  if os.path.isfile('recortes_color{a}.jpg'.format( a = i)):
   nombre_archivo = 'recortes_prueba{a}.jpg'.format(a = i)
   nombre_ar = 'recortes_color{a}.jpg'.format(a = i)
   im = cv2.imread(nombre_archivo, 0)
   im_ = cv2.imread(nombre_ar, 1)

   cv = chan_vese(im, mu=0.35, lambda1=1, lambda2=1, tol=1e-3, max_iter=200,
              dt=0.8, init_level_set="checkerboard", extended_output=True)

   [f,c] = im.shape
   F = 0
   C = 0

   for C in list(range(c)):
    for F in list(range(f)):
      if cv[0][F][C] == True and im[F,C] > 10:
         im[F,C] = 255
      else:
         im[F,C] = 0

   kernel_e = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (6,6))
   im_er = cv2.erode(im, kernel_e, iterations=1)

   kernel_d = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (30,25))
   im_dil = cv2.dilate(im_er, kernel_d, iterations = 1)

   co,_ = cv2.findContours(im_dil, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

```
  for con in co:
   area = cv2.contourArea(con)
   if area > 200:
     cv2.drawContours(im,[con], 0, 100, -1, cv2.LINE_AA)

     F = 0
     C = 0
     for C in list(range(c)):
      for F in list(range(f)):
       if im[F,C] != 100:
         im_[F,C,:] = (255,0,0)

     im__ = cv2.cvtColor(im_, cv2.COLOR_RGB2GRAY)

     if np.average(im_[:,:,0]) < 200:
       aceitunas += 1

       limi = np.array([17,23,42])
       lims = np.array([32,38,56])
       imm = cv2.inRange(im_,limi,lims)

       if np.average(imm) > 0.5:
        suelo += 1

        imagen_suelo= cv2.addWeighted(imm,0.5,im__,0.8,1)
        nombre = 'aceituna_suelo{a}.jpg'.format(a=i)
        cv2.imwrite(nombre, imagen_suelo)
       else:

        immm = cv2.Canny(im__, 10, 70)
        imagen_rugosa = cv2.addWeighted(immm,0.5,im__,0.8,1)

        if np.average(imagen_rugosa) > 46.0:
         rugosa +=1

         nombre = 'aceituna_rugosa{a}.jpg'.format(a=i)
         cv2.imwrite(nombre, imagen_rugosa)

return [aceitunas, suelo, rugosa]
```

Función **Todas_imagenes**:

```
import os
import numpy as np
import pandas as pd

from seleccion_hojas import seleccion_hojas
from sub_imagenes import sub_imagenes
from metodo_van_chese import metodo_van_chese

def Todas_imagenes(num = 3):
 hoja = []
 tuna = []
 suel = []
 rugos = []
 for u in list(range(num)):
  if os.path.isfile('aceituna{a}.jpg'.format( a = u)):
        nom_archivo = 'aceituna{a}.jpg'.format(a = u)
        p = seleccion_hojas(nom_archivo, u)
        hoja.append(round(p))
        tuna.append(100 - hoja[u])
        sub_imagenes(u)
        [a,s,r]=metodo_van_chese()
        suel.append(round(tuna[u]*s/a))
        rugos.append(round(tuna[u]*r/a))
        a_suelo = round(np.mean(suel))
        a_rugosa = round(np.mean(rugos))
        p_hoja = round(np.mean(hoja))
        a_buena = round(np.mean(tuna)-a_rugosa-a_suelo)

 data = {'media': ['aceituna_suelo', 'aceituna_rugosa', 'porcentaje_hoja',
'aceituna_buena'],
        'valor': [a_suelo, a_rugosa, p_hoja, a_buena]}
 df = pd.DataFrame(data, columns = ['media', 'valor'])

 df.to_excel('Todas_imagenes.xlsx', sheet_name = 'Todas_imagenes')
```

## 2.2. INTERFAZ

En este apartado se agrupa la función generada para realizar la interfaz mediante el entorno Dash by Plotly.

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import os
import numpy as np
import pandas as pd
import base64


from seleccion_hojas import seleccion_hojas
from sub_imagenes import sub_imagenes
from metodo_van_chese import metodo_van_chese
from Todas_imagenes import Todas_imagenes

Todas_imagenes()
df = pd.read_excel('Todas_imagenes.xlsx')
datos = np.array(df['valor'])
etiquetas = np.array(df['media'])

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
colors = {'background': '#111111','text': '#7FDBFF', 'subtext': '#EF963B'}

 app.layout = html.Div(
        style={'backgroundColor': colors['background']},
        children= [
         html.H1(
          'CALIDAD ACEITUNA',
           style={'textAlign': 'center', 'color': colors['text']}
         ),
         dcc.Tabs(
                id= 'tabs',
                value = 'tab-2',
                children=[
                  dcc.Tab(
                        label = 'Selección una imagen',
                        value = 'tab_1',
                        children=["Elegir el número de imagen que se quiere procesar.",
                                dcc.Input(
```

```
                        id = 'input',
                        type= 'int',
                        value = 1 ,
                        placeholder = "Introducir número imagen"
                        ),
                    dcc.Graph(id = 'Pie'),
                    html.Div(
                        'Imagen original',
                        style={'textAlign': 'left', 'color': colors['text']}
                        ),
                    html.Div(
                        html.Img(
                            id = 'im1',
                            style = {'width' : '1000px'}
                            )
                        ),
                    html.Div(
                        'Imagen procesada',
                        style={'textAlign': 'left', 'color': colors['text']}
                        ),
                    html.Div(
                        html.Img(
                            id = 'im2',
                            style={'width': '1000px'}
                            )
                        )
                    ]
                ),
            dcc.Tab(
                label = 'Procesar todas las imágenes disponibles',
                value = 'tab_2',
                children = ["Datos de todas las imágenes disponibles",
                    dcc.Graph(
                        id = 'Pie2',
                        figure = {
                            'data': [go.Pie(
                                labels = etiquetas,
                                values = datos,
                                automargin = True
                                    )
                                ]
                            }
                        )
                    ])])

])
```

```
@app.callback(
        dash.dependencies.Output('Pie', 'figure'),
        [dash.dependencies.Input('input', 'value')]
)

def update_figure(input_data):
    i = 'aceituna{}.jpg'.format(input_data)
    Por_h = seleccion_hojas(i, input_data)
    por_h = round(Por_h)
    por_a = 100 - por_h
    sub_imagenes(input_data)
    [aceituna, suelo, rugosa] = metodo_van_chese()
    por_a_suelo = round(por_a*suelo/aceituna)
    por_a_rugosa = round(por_a*rugosa/aceituna)
    return {'data': [go.Pie(labels = ['Aceituna buena','Aceituna del suelo','Aceituna estropeada' , 'Hoja'],
                        values = [(por_a - por_a_suelo - por_a_rugosa),por_a_suelo, por_a_rugosa, por_h], automargin = True)]}

@app.callback(
 dash.dependencies.Output('im1', 'src'),
 [dash.dependencies.Input('input', 'value')]
)

def update_image_1(input_value):
   image_path_1 = 'aceituna{}.jpg'.format(input_value)
   encoded_image_1 = base64.b64encode(open(image_path_1, 'rb').read())
   return '{}'.format(encoded_image_1.decode())

@app.callback(
 dash.dependencies.Output('im2', 'src'),
 [dash.dependencies.Input('input', 'value')]
)

def update_image_2(input_value):
   image_path_2 = 'mascara{}.jpg'.format(input_value)
   encoded_image_2 = base64.b64encode(open(image_path_2, 'rb').read())
   return '{}'.format(encoded_image_2.decode())


if __name__ == '__main__':
    app.run_server(port=8050, host= '0.0.0.0')
```