

ÍNDICE GENERAL

1	Introducción	1
1.1.	Procesamiento digital de imágenes	1
1.2.	Procesamiento digital de imágenes: fundamentos y aplicaciones con GNU Octave y OpenCV	1
2	Planteamiento del problema	3
2.1.	Clasificación de hojas de café	3
2.2.	Objetivo general	3
2.3.	Objetivo específico	3
2.4.	Alcance	3
2.5.	Justificación	4
2.6.	Especificaciones técnicas	4
2.7.	Metodología	4
2.7.1.	Modelo de colores HSV	4
3	Procedimiento	5
3.1.	Lectura del dataset	5
3.1.1.	La clase CoffeeLeaf	6
3.2.	Procesado de la imagen	7
3.2.1.	Conversión de BGR a RGB	8
3.2.2.	Creación de las regiones de interés	8
3.2.3.	Creación de la máscara	9
3.2.4.	Enmascaramiento de las regiones de interés	9
3.2.5.	Histograma de la región de interés	12

3.2.6.	Segmentación de la imagen	13
3.2.7.	Clasificación de la hoja	13
3.3.	Presentación de los datos	14
3.3.1.	Resumen	14
3.3.2.	Imagen original	15
3.3.3.	Máscara	15
3.3.4.	Regiones de interés	16
3.3.5.	Imagen segmentada	16
3.3.6.	Histograma	16
4	Resultados	19
4.1.	Resultado principal	19
4.1.1.	Segmentación Hue en 30 y 120	19
4.2.	Resultados secundarios	21
4.3.	Casos especiales	21
4.3.1.	Iluminación	21
4.3.2.	Envés de la hoja	21
5	Conclusiones	23
5.1.	Conclusiones específicas	23
5.1.1.	Mejoras y trabajo a futuro	23
5.2.	Conclusiones generales	23
	Referencias	23

ÍNDICE DE CÓDIGOS

1.	Cargar las anotaciones del dataset	5
2.	La clase CoffeeLeaf	6
3.	Lista de objetos CoffeeLeaf	7
4.	Función process de la clase CoffeeLeaf	7
5.	Iniciar procesamiento de las imágenes	7
6.	Convertir imagen BGR a RGB	8
7.	Puntos (x, y) del contorno de la hoja de café	9
8.	Crear regiones de interés	9
9.	Crear máscara	11
10.	Enmascarar las regiones de interés	12
11.	Cálculo de histograma de la región de interés	13
12.	Segmentar la región de interés	14
13.	Clasificar hoja de café	15
14.	Describir objeto CoffeeLeaf	15
15.	Mostrar resumen de la clasificación	16
16.	Mostrar imagen original	16
17.	Mostrar máscara	16
18.	Mostrar regiones de interés	17
19.	Mostrar segmentación de la imagen	17
20.	Mostrar histograma de la región de interés	17

ÍNDICE DE FIGURAS

3.1. Imagen original en RGB	8
3.2. Polígono que delimita el contorno de la hoja	9
3.3. Cuadro que aísla a la hoja de café	10
3.4. Región de interés en RGB	10
3.5. Región de interés en HSV	11
3.6. Máscara	11
3.7. Región de interés RGB enmascarada	12
3.8. Región de interés Hue enmascarada	12
3.9. Histograma de la región de interés	13
3.10. Imagen segmentada	14
4.1. Eficiencia detectando el estado con Hue 30-120	20
4.2. Eficiencia detectando la categoría con Hue 30-120	20
4.3. Eficiencia por categoría con Hue 30-120	21

ÍNDICE DE TABLAS

3.1. Anotaciones del dataset	6
4.1. Eficiencias generales con Hue 30-120	19
4.2. Eficiencia por categoría con Hue 30-120	20



1

INTRODUCCIÓN

1.1. Procesamiento digital de imágenes

1.2. Procesamiento digital de imágenes: fundamentos y aplicaciones con GNU Octave y OpenCV

Durante el seminario *Procesamiento Digital de Imágenes: Fundamentos y Aplicaciones con GNU Octave y OpenCV* llevado a cabo en los meses de junio a agosto de 2025 e impartido por el PhD. Luis Escalante Zárate se revisaron varios temas principales acerca del procesamiento digital de imágenes.



PLANTEAMIENTO DEL PROBLEMA

2.1. Clasificación de hojas de café

2.2. Objetivo general

Demostrar las habilidades adquiridas durante el seminario *Procesamiento Digital de Imágenes: Fundamentos y Aplicaciones con GNU Octave y Open CV* aplicando los principios y técnicas básicas de manera práctica a un proyecto en particular.

2.3. Objetivo específico

Crear un algoritmo que clasifique hojas de café como sanas o infectadas y su nivel de afectación, y evaluar su eficiencia comparando los resultados obtenidos con los proporcionados en el conjunto de datos.

2.4. Alcance

A pesar de que el conjunto de datos de prueba contiene seis clasificaciones para las hojas, el algoritmo desarrollado sólo incluirá la clasificación sana y los cuatro niveles de



afectación, excluyendo la clasificación *araña roja* debido a las retriicciones en el tiempo del proyecto.

2.5. Justificación

El algoritmo y las técnicas utilizadas pueden aplicarse de manera directa en el mundo real dentro del área de la agricultura y/o agronomía, e idealmente puede servir como base para desarrollar procesos automatizados para el control de calidad en el campo del café.

2.6. Especificaciones técnicas

Se utilizará Python como lenguaje de programación para la implementación del algoritmo debido a su facilidad de uso y al amplio número de bibliotecas disponibles para el procesamiento de imágenes tales como OpenCV.

2.7. Metodología

2.7.1. Modelo de colores HSV



3

PROCEDIMIENTO

A continuación se describen los procesos del algoritmo que permiten solucionar el problema especificado. El código fuente está disponible de manera digital en la plataforma de GitHub [1].

3.1. Lectura del dataset

Se comienza leyendo el dataset que contiene información que ha sido etiquetada de manera manual por los autores del mismo. Este proceso se describe en Código 1.

```
import json

annotations_file = "RoCoLe.json"

with open(annotations_file, "r") as f:
    annotations = json.load(f)
```

Código 1: Cargar las anotaciones del dataset

Utilizando la biblioteca `json` de Python leemos el archivo `RoCoLe.json` (originalmente `Annotations/RoCoLe-json.json`). La variable `annotations` contiene la información necesaria para contruir nuestro conjunto de datos de prueba (véase Tabla 3.1).



Anotación	Descripción
ID	Identificador de la hoja
Label.Leaf.0.state	Estado de la hoja como saludable o infectada
Label.classification	Clasificación de la hoja o nivel de afectación
Label.Leaf.0.geometry	Puntos (x,y) que determinan el contorno de la hoja

Tabla 3.1: Anotaciones del dataset

3.1.1. La clase CoffeeLeaf

A continuación creamos una clase llamada `CoffeeLeaf` la cual se encarga de contener los datos proporcionados en las anotaciones y que representa a una hoja de café. Los atributos de esta clase pueden observarse en Código 2.

```
class CoffeeLeaf:
    def __init__(self, leaf_id, state, classification, image_bgr,
        ↪ geometry):
        self.id = leaf_id
        self.state_manual = state
        self.state_computed = None
        self.classification_manual = classification
        self.classification_computed = None
        self.image_bgr = image_bgr
        self.image_rgb = None
        self.roi_rgb = None
        self.roi_hsv = None
        self.masked_roi_rgb = None
        self.masked_roi_hue = None
        self.mask = None
        self.area = None
        self.affected_percentage = None
        self.histogram_hue = None
        self.limit_below = None
        self.limit_above = None
        self.binary = None
        self.contours = None
        self.contours_canvas = None
        self.polygon = None
        self.geometry = geometry
        self._processed = False
```

Código 2: La clase CoffeeLeaf

Una vez creada la clase `CoffeeLeaf` y leído los datos del dataset, procedemos a crear la lista `coffee_leaves` utilizando los datos de la Tabla 3.1, tal como se muestra en Código 3. El directorio `../rocole.photos/` contiene los archivos `.jpeg` del dataset (`Annotations/RoCoLe-voc.tar.gz/export`).



```
IMAGES_PATH = "../rocole_photos/"

coffee_leaves = []

for annotation in annotations:
    classification = annotation["Label"]["classification"]
    if classification != "red_spider_mite":
        leaf_id = annotation["ID"]
        state = annotation["Label"]["Leaf"][0]["state"]
        geometry = annotation["Label"]["Leaf"][0]["geometry"]
        leaf_image = f"{IMAGES_PATH}/{leaf_id}.jpeg"
        image_bgr = cv.imread(leaf_image)
        coffee_leaf = CoffeeLeaf(leaf_id, state, classification,
                                ↪ image_bgr, geometry)
        coffee_leaves.append(coffee_leaf)
```

Código 3: Lista de objetos CoffeLeaf

3.2. Procesado de la imagen

Una vez creada la lista `coffee_leaves` iniciamos el procesamiento de las imágenes a través de la función `process` (Código 4) de la clase `CoffeeLeaf`. Véase Código 5.

```
def process(self):
    self._generate_image_rgb()
    self._create_polygon()
    self._create_roi()
    self._create_mask()
    self._create_masked_roi()
    self._compute_histogram()
    self._binarize()
    self._categorize()
    self._processed = True
```

Código 4: Función process del la clase CoffeeLeaf

```
for coffee_leaf in coffee_leaves:
    coffee_leaf.process()
```

Código 5: Iniciar procesamiento de las imágenes



3.2.1. Conversión de BGR a RGB

Debido a que cuando creamos los objetos `CoffeeLeaf` el argumento `image_bgr` pasa datos de una imagen en color BGR (Blue, Green, Red), que es la representación de color por defecto de *OpenCV*, y ya que *matplotlib*, la herramienta para la visualización de las imágenes, utiliza una representación RGB (Red, Green, Blue), una conversión de color es necesaria. Véase Código 6.

```
def _generate_image_rgb(self):  
    self.image_rgb = cv.cvtColor(self.image_bgr, cv.COLOR_BGR2RGB)
```

Código 6: Convertir imagen BGR a RGB

El resultado de esta conversión permite visualizar la imagen original (Figura 3.1).



Figura 3.1: Imagen original en RGB

3.2.2. Creación de las regiones de interés

La anotación `geometry` del dataset nos provee de una serie de puntos (x,y) que representan el contorno de la hoja de café y que nos sirven para crear un polígono (Código 7).

El contorno de la hoja de café se puede apreciar en la Figura 3.2.

Posteriormente creamos el rectángulo mínimo que encierra a esta región, como se aprecia en la Figura 3.3 y lo usamos para recortar nuestra región de interés (véase Código 8).

Creamos dos regiones de interés: la primera en RGB (Figura 3.4) para la presentación al usuario y la segunda en HSV (Hue, Saturation, Value) (Figura 3.5) que nos servirá en la segmentación de la imagen.



```
def _create_polygon(self):  
    polygon_points = [list(point.values()) for point in self.geometry]  
    self.polygon = np.array(polygon_points)
```

Código 7: Puntos (x, y) del contorno de la hoja de café



Figura 3.2: Polígono que delimita el contorno de la hoja

```
def _create_roi(self):  
    x,y,w,h = cv.boundingRect(self.polygon)  
    self.roi_rgb = self.image_rgb[y:y+h, x:x+w].copy()  
    self.roi_hsv = cv.cvtColor(self.roi_rgb, cv.COLOR_RGB2HSV)
```

Código 8: Crear regiones de interés

3.2.3. Creación de la máscara

Utilizando las regiones de interés previamente creadas y el polígono que delimita el contorno de la hoja, procedemos a crear una máscara (Código 9) que nos será útil en las operaciones matriciales del procesamiento de la imagen. Véase Figura 3.6.

Nótese que en Código 9 calculamos el área que ocupa la máscara (píxeles en color blanco), o lo que es mismo, el área de la hoja de café.

3.2.4. Enmascaramiento de las regiones de interés

Una vez teniendo la máscara y las regiones de interés, procedemos a *enmascarar* dichas regiones (Código 10). Para el caso de la región de interés en RGB es por mera conveniencia



Figura 3.3: Cuadro que aísla a la hoja de café



Figura 3.4: Región de interés en RGB

al presentar esta región al usuario. Sin embargo, para el enmascaramiento de la region de interés en **HSV** es de suma importancia usar únicamente el canal Hue (Matiz) puesto que nuestra segmentación se basará en el color.

El resultado de este enmascaramiento se puede apreciar en la Figura 3.7 prara RGB y en la Figura 3.8 para Hue.

Nótese que el valor más bajo para el canal Hue no es un color negro sino un matiz rojo. Esto se debe a la representación circular que emplea el modelo **HSV**.

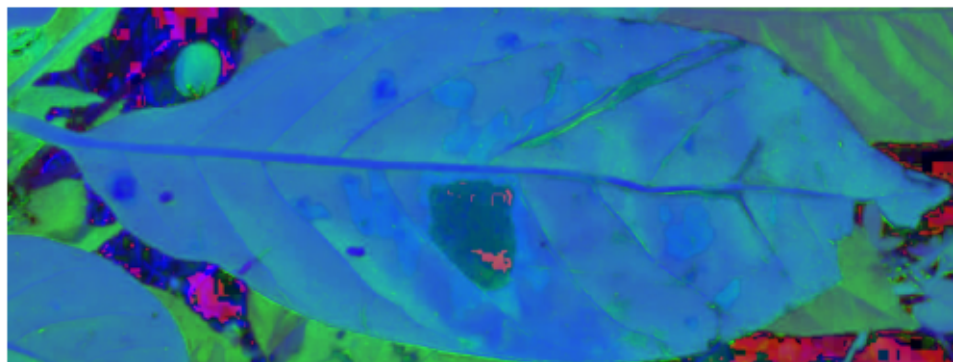


Figura 3.5: Región de interés en HSV

```
def _create_mask(self):  
    self.mask = np.zeros(self.roi_hsv.shape[:2], np.uint8)  
    polygon_start = self.polygon.min(axis=0)  
    polygon_at_zero = self.polygon - polygon_start  
    CONTOURS = -1 # All contours  
    COLOR = (255, 255, 255) # White  
    THICKNESS = -1 # Fill  
    self.mask = cv.drawContours(self.mask, [polygon_at_zero], CONTOURS,  
                                ↪ COLOR, THICKNESS)  
    self.area = cv.countNonZero(self.mask)
```

Código 9: Crear máscara

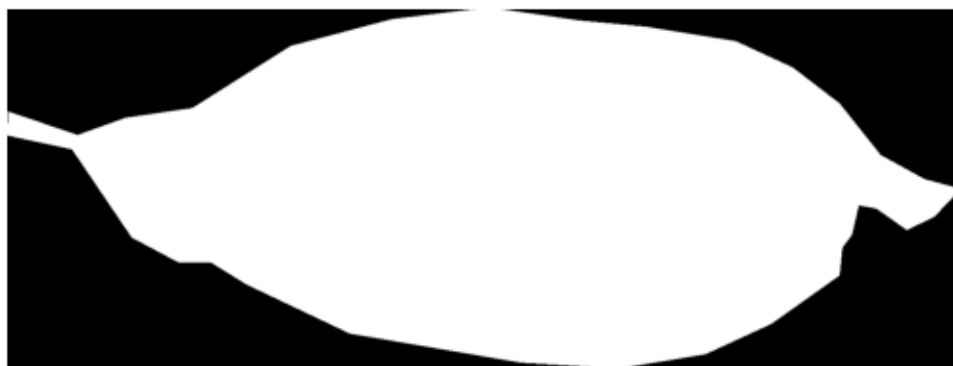


Figura 3.6: Máscara



```
def _create_masked_roi(self):  
    self.masked_roi_rgb = cv.bitwise_and(self.roi_rgb, self.roi_rgb,  
        ↪ mask=self.mask)  
    self.masked_roi_hue = cv.bitwise_and(self.roi_hsv[:, :, 0], self.mask)
```

Código 10: Enmascarar las regiones de interés



Figura 3.7: Región de interés RGB enmascarada

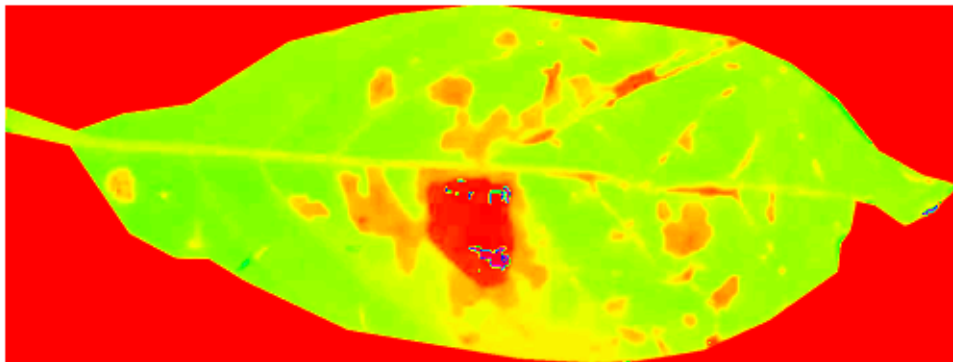


Figura 3.8: Región de interés Hue enmascarada

3.2.5. Histograma de la región de interés

Una vez que tenemos la región de interés en el canal Hue, procedemos a calcular su histograma (Código 11) utilizando la máscara (Figura 3.6) de tal manera que no representemos datos fuera del contorno de la hoja. El resultado de la distribución de color puede verse en la Figura 3.9.



```
def _compute_histogram(self):
    hsv_normalizer = Normalize(vmin=0, vmax=179)
    self.hsv_mappable = ScalarMappable(norm=hsv_normalizer, cmap='hsv')
    self.histogram_hue = cv.calcHist([self.masked_roi_hue], [0],
    ↪ self.mask, [180], [0,180])
```

Código 11: Calcular histograma de la región de interés

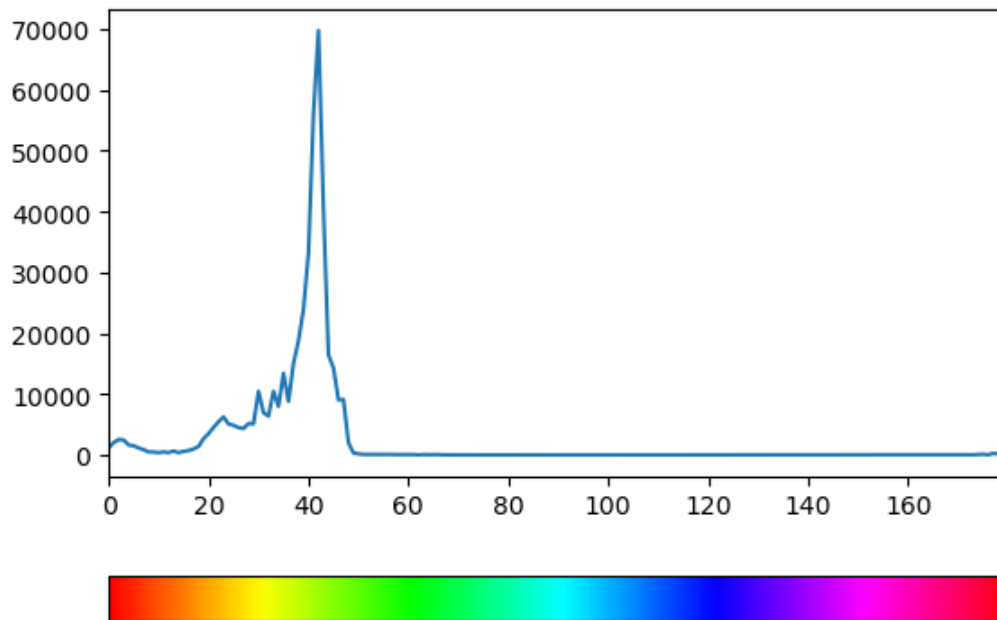


Figura 3.9: Histograma de la región de interés

3.2.6. Segmentación de la imagen

Del histograma nos interesa la región de la hoja que se considera saludable, es decir, la región del espectro Hue en matices verdes, un poco del amarillo-naranja (umbral inferior) y hasta la región azul (umbral superior) para cuando las hojas tengan un verde intenso acercándose a tonos azules.

En el Código 12 hemos asignado al umbral inferior el valor de 30 y al umbral superior el valor de 120. Segmentamos cada parte de los umbrales y el resultado los unimos usando una operación AND elemento por elemento, resultando en la imagen segmentada apropiadamente (Figura 3.10).

3.2.7. Clasificación de la hoja

Una vez segmentada la hoja de café en su parte saludable (píxeles blancos) e infectada (píxeles negros dentro de la máscara), procedemos a clasificarla utilizando la misma



```
def _binarize(self):  
    GREEN_LIMIT_BELOW = 30  
    GREEN_LIMIT_ABOVE = 120  
    _, segments_below_green = cv.threshold(self.masked_roi_hue,  
    ↪ GREEN_LIMIT_BELOW, 179, cv.THRESH_BINARY)  
    _, segments_above_green = cv.threshold(self.masked_roi_hue,  
    ↪ GREEN_LIMIT_ABOVE, 179, cv.THRESH_BINARY_INV)  
    self.binary = cv.bitwise_and(segments_below_green,  
    ↪ segments_above_green)
```

Código 12: Segmentar la región de interés



Figura 3.10: Imagen segmentada

ponderación usada por los autores (REF Tabla).

Para el cálculo del porcentaje afectado, simplemente calculamos el área saludable de la hoja a partir de la imagen segmentada (Figura 3.10) y restamos este valor al área de la máscara (Código 9), como se describe en Código 13.

Esta clasificación marca el fin del algoritmo para el procesamiento de la imagen, obteniéndose dos categorías: 1) el estado de la hoja y 2) el nivel de afectación en caso de ser una hoja infectada.

3.3. Presentación de los datos

A continuación se muestran las funciones que permiten describir el objeto *Coffee Leaf* de una manera intuitiva al usuario. La función `describe` (Código 14) orquesta la presentación pero cada función puede accederse de manera individual.

3.3.1. Resumen

La función `show_summary` (Código 15) muestra la clasificación original de la hoja, la nueva clasificación calculada y el porcentaje de afectación calculado.



```
def _categorize(self):
    healthy_area = cv.countNonZero(self.binary)
    affected_area = self.area - healthy_area
    self.affected_percentage = int((affected_area / self.area) * 100)
    if self.affected_percentage < 1:
        self.state_computed = "healthy"
        self.classification_computed = "healthy"
    elif self.affected_percentage < 6:
        self.state_computed = "unhealthy"
        self.classification_computed = "rust_level_1"
    elif self.affected_percentage < 21:
        self.state_computed = "unhealthy"
        self.classification_computed = "rust_level_2"
    elif self.affected_percentage < 51:
        self.state_computed = "unhealthy"
        self.classification_computed = "rust_level_3"
    else:
        self.state_computed = "unhealthy"
        self.classification_computed = "rust_level_4"
```

Código 13: Clasificar hoja de café

```
def describe(self):
    if not self._processed:
        print("Imagen aún no procesada")
        return
    self.show_summary()
    self.show_original_image()
    self.show_mask()
    self.show_roi()
    self.show_roi(hue=True)
    self.show_binary()
    self.show_histogram()
```

Código 14: Describir objeto CoffeeLeaf

3.3.2. Imagen original

La función `show_original_image` (Código 16) muestra la imagen original en RGB (Figura 3.1) y agrega un título.

3.3.3. Máscara

La función `show_mask` (Código 17) muestra la máscara calculada (Figura 3.6) y agrega un título. Nótese que es necesario especificar un mapa de colores, en este caso *gray*, para poder visualizar la máscara en escala de grises, ya que *matplotlib* utiliza el mapa de colores



```
def show_summary(self):
    original_class = self.classification_manual
    new_class = self.classification_computed
    affected_area = self.affected_percentage
    title_md =
    ↪ f"### {original_class} --> {new_class} ({affected_area} %)"
    display(Markdown(title_md))
```

Código 15: Mostrar resumen de la clasificación

```
def show_original_image(self):
    plt.imshow(self.image_rgb)
    plt.title("Imagen Original")
    plt.axis("off")
    plt.show()
```

Código 16: Mostrar imagen original

viridis por defecto.

```
def show_mask(self):
    plt.imshow(self.mask, cmap="gray")
    plt.title("Máscara")
    plt.axis("off")
    plt.show()
```

Código 17: Mostrar máscara

3.3.4. Regiones de interés

La función `show_roi` (Código 18) se encarga de mostrar las regiones de interés enmascaradas y agrega un título. Esta función recibe un argumento *booleano* `hue` para indicar si se debe mostrar la región de interés en RGB (Figura 3.7) o bien la del canal Hue (Figura 3.8). Nótese que si se quiere mostrar la región de interés del canal Hue, es necesario especificar el mapa de colores *hsv* para una visualización apropiada.

3.3.5. Imagen segmentada

La función `show_binary` (Código 19) se encarga de mostrar la imagen segmentada (Figura 3.10) y agrega un título.

3.3.6. Histograma

Finalmente, la función `show_histogram` (Código 20) se encarga de mostrar el histograma (Figura 3.9) de la región de interés del canal Hue (Figura 3.8) y agrega un título,



```
def show_roi(self, hue=False):
    if hue:
        colorspace = "Hue"
        plt.imshow(self.masked_roi_hue, cmap="hsv")
    else:
        colorspace = "RGB"
        plt.imshow(self.masked_roi_rgb)
    plt.title(f"Región de Interés ({colorspace})")
    plt.axis("off")
    plt.show()
```

Código 18: Mostrar regiones de interés

```
def show_binary(self):
    plt.imshow(self.binary, cmap="gray")
    plt.title(f"Segmentación")
    plt.axis("off")
    plt.show()
```

Código 19: Mostrar segmentación de la imagen

el cual contiene el valor Hue (o Matiz) más abundante.

```
def show_histogram(self):
    fig, ax = plt.subplots()
    ax.plot(self.histogram_hue)
    colorbar = plt.colorbar(self.hsv_mappable, ax=ax, location="bottom")
    colorbar.set_ticks([])
    idx_max = np.argmax(self.histogram_hue)
    plt.title(f"Histograma (Hue) Máx={idx_max}")
    plt.margins(x=0)
    plt.show()
```

Código 20: Mostrar histograma de la región de interés



4

RESULTADOS

En el capítulo anterior describimos el proceso para segmentar las imágenes a través del canal Hue. Empleando diferentes valores para los parámetros de umbral inferior y umbral superior (Sección 3.2.6) se obtienen los siguientes resultados.

4.1. Resultado principal

A continuación se presenta el resultado principal, definido en Código 12.

4.1.1. Segmentación Hue en 30 y 120

La Tabla 4.1 muestra el porcentaje de eficiencia alcanzado para clasificar las hojas de café por su estado saludable o infectado (Figura 4.1), o bien, el porcentaje de eficiencia global para las categorías (Figura 4.2).

Clasificación	Total	Aciertos	Errores	Eficiencia
Estado	1393	962	431	69.059
Categoría	1393	715	678	51.328

Tabla 4.1: Eficiencias generales con Hue 30-120

A su vez la Tabla 4.2 muestra la eficiencia alcanzada por cada una de las categorías: saludable, roya nivel 1, roya nivel 2, roya nivel 3 y roya nivel 4 (Figura 4.3).

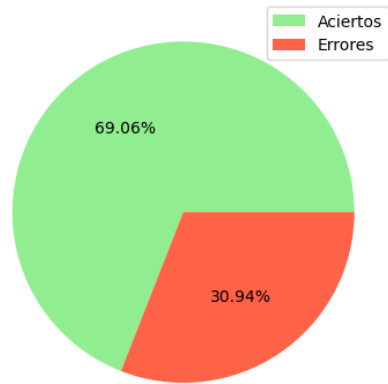


Figura 4.1: Eficiencia detectando el estado con Hue 30-120

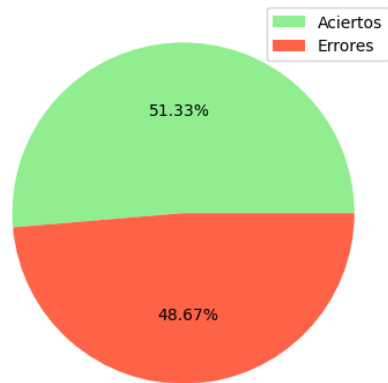


Figura 4.2: Eficiencia detectando la categoría con Hue 30-120

Categoría	Original	Calculado	Eficiencia
healthy	791	445	56.257
rust_level_1	344	196	56.976
rust_level_2	166	67	40.361
rust_level_3	62	6	9.677
rust_level_4	30	1	3.333

Tabla 4.2: Eficiencia por categoría con Hue 30-120

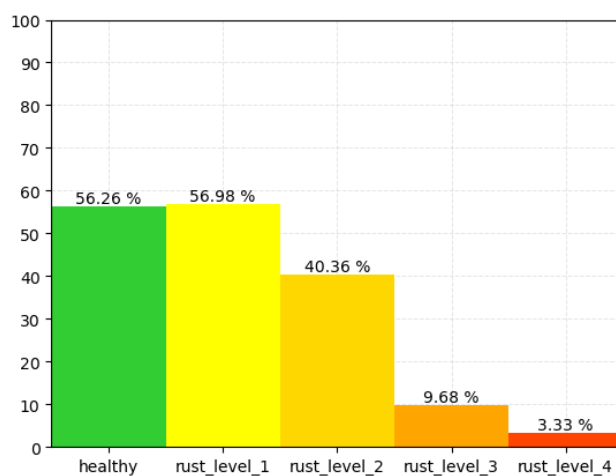


Figura 4.3: Eficiencia por categoría con Hue 30-120

4.2. Resultados secundarios

4.3. Casos especiales

4.3.1. Iluminación

4.3.2. Envés de la hoja



5

CONCLUSIONES

5.1. Conclusiones específicas

5.1.1. Mejoras y trabajo a futuro

5.2. Conclusiones generales

REFERENCIAS

- [1] L. Dominguez, “Coffee Leaves Classification,” 07 2025. [Online]. Available: <https://github.com/LindermanDgz/coffee-leaves-classification>

