

Práctica 1. Manejo básico de imágenes y visualización Iris Setosa

Universidad Nacional Autónoma de México

Facultad de Ingeniería

Reconocimiento de Patrones (757-1)

Basile Álvarez Andrés José

Keller Ascencio Rodolfo Andrés

Shen Shuai

*

Resumen—A lo largo de este documento se presentarán los resultados y análisis de nuestra primera práctica de la materia de Reconocimiento de Patrones, donde se estará trabajando con el manejo básico de imágenes y la visualización Iris Setosa. El objetivo principal de esta práctica busca que el alumno conozca las diversas formas de desplegar imágenes en distintos formatos, así como algunos métodos de manipulación de imágenes siendo esta una práctica introductoria para el resto de trabajos que se realizarán en la materia a lo largo del semestre.

I. INTRODUCCIÓN

A lo largo de la realización de esta práctica se busca aprender acerca del manejo de imágenes, haciendo uso de diversas herramientas que nos permitan realizar un correcto análisis de una imagen.

Como se menciona en el documento de la práctica, una imagen es una figura o representación visual en dos dimensiones que contiene información como lo es el color o la intensidad de la luz, cuya unidad mínima de composición resulta ser el pixel. La imagen dispone de un número m de columnas por un número n de filas, formando un arreglo ordenado donde se encuentran dispuestos los píxeles. Las imágenes pueden contar con uno o múltiples canales dependiendo del formato y contenido de la imagen, así como el tipo de color con el que cuente.

Una imagen contiene información importante con respecto a su contenido, donde dependiendo de la naturaleza de la imagen podemos hacer uso de esta información o composición para realizar un análisis más completo. Dentro de las características que se evaluarán y analizarán a lo largo de la práctica, se trabajará con la resolución de la imagen, el formato de la imagen y el tipo de imagen.

Con respecto a la resolución de la imagen se reconoció la existencia de tres tipos de resolución; la resolución espacial, que se encarga de indicar la cantidad de píxeles dispuestos en columnas y renglones con las que cuenta una imagen, expresando estas relaciones como 600x400 o 1024x600; la resolución temporal, que indica la cantidad de cuadros por unidad de tiempo de una imagen continua donde a manera de ejemplo contamos con los cuadros por segundo o *frames per second* (FPS); y la resolución por bit, la cual define el número posible de valores de intensidad o color que un pixel puede tomar, por ejemplo, una imagen binaria cuenta con dos colores y una imagen en escala de grises cuenta con 256 colores.

Como segunda característica a destacar a lo largo de esta práctica, el formato de imagen se refiere a la manera mediante la cual se va a almacenar la información de una imagen. Existe un gran número de formatos de imagen donde algunos de los más utilizados son el formato JPEG, TIFF, DICOM, RAW, PNG, entre otros.

- El formato JPEG es un formato reconocido por su capacidad de compresión de datos, el cuál permite reducir de manera significativa el tamaño y peso de la imagen con una pérdida mínima de información visual haciendo uso de la Transformada de Coseno Discreta.[1]
- El formato RAW, a diferencia de JPEG, es un formato que almacena datos de imagen sin comprimir ni procesar, lo cual permite captar casi todos los detalles que se aprecian desde un visor. Debido a sus características, es uno de los tipos de formato de imagen más pesados pues almacena la mayor cantidad de detalles posibles para que, posteriormente, la información pueda ser utilizada, convertida o comprimida a otros formatos dependiendo del uso que se le quiera dar a la imagen.[2]
- El formato PNG es empleado generalmente para mostrar imágenes digitales de alta calidad ofreciendo una compresión sin pérdida de datos así como una paleta de colores más variada y brillante.[3]
- El formato TIFF permite almacenar y codificar documentos escaneados, pues se utilizaba para datos de un solo bit, sin embargo, actualmente pueden almacenar 16 bits. [4]
- El formato DICOM es un formato utilizado para codificar y transmitir Imágenes de Resonancia Magnética (MRI) al igual que Imágenes de Tomografía Computarizada debido a que permite almacenar información del paciente o estudio realizado a la par que la imagen.

Otra característica de las imágenes se refiere a su tipo, donde la elección del tipo de imagen se da dependiendo de la información que se quiera obtener de la imagen y el uso que se le dará a la misma, así como su resolución de bit, pues existen imágenes de un solo canal o de múltiples canales. Las imágenes de canal único representan a cada pixel por un solo valor, por otro lado, las imágenes multicanal representan a cada pixel por una tupla de valores.

Imágenes de canal único:

- Las imágenes binarias representan a cada pixel por medio de un valor binario, ya sea 0 ó 1, donde el cero representa el color negro y el 1 el blanco.[5]



Figura 1. Tipo de Imagen. Binaria

- Las imágenes en escala de grises permiten que cada pixel sea representado por 8 bits, es decir, que puede ser representado por un valor entero que se encuentre en un rango de 0 a 255.[6]

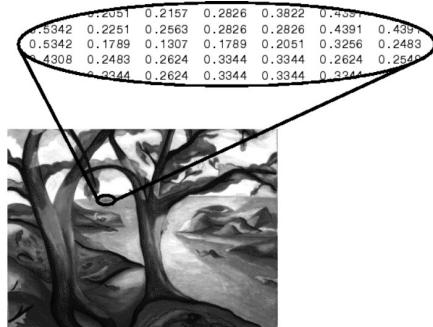


Figura 2. Tipo de Imagen. Escala de Grises

Imágenes de canal múltiple:

- Las imágenes RGB son imágenes cuyos pixeles cuentan con arreglos de tres valores, donde cada uno indica la cantidad de color rojo, verde o azul con el que cuenta un pixel de la imagen.[6]

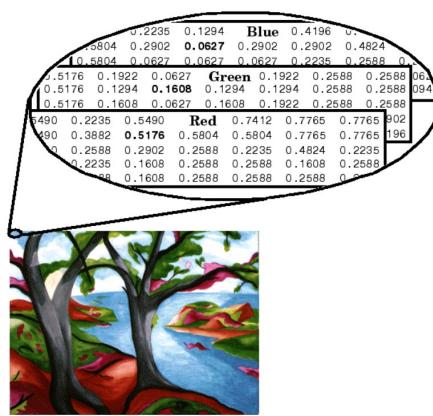


Figura 3. Tipo de Imagen. RGB

- Las imágenes HSV son imágenes que al igual que el caso de las RGB, cada pixel cuenta con un arreglo de tres valores, sin embargo, este arreglo indica la cantidad de color (H), saturación (S) y brillo (V) del pixel.
- Las imágenes RGBA, por su lado, son similares a las RGB, sin embargo, su diferencia radica en que este tipo de imagen cuenta con un cuarto valor en el arreglo del pixel, el cual indica la transparencia.

II. OBJETIVO

El alumno conocerá las distintas formas de desplegar una imagen en distintos formatos, así como algunas manipulaciones básicas para el procesamiento de imágenes.

III. PARTE A: IMÁGENES

III-A. 4.1.Desarrolla un script para leer y desplegar cada imagen con los paquetes de Matplotlib, OpenCV, Scikit-Image y PIL.

Para este primer ejercicio de la práctica, se creó un *script* a modo de función, el cual recibe como parámetros la *url* de la imagen que se quiere leer y desplegar, así como el *paquete* con el cual se desplegará (Matplotlib, OpenCV, Scikit-Image y PIL). Para el caso de *paquete = 1* (Matplotlib), la imagen se puede leer utilizando la función *imread(url)* de *matplotlib.image*. Una vez que la imagen se leyó con Matplotlib, se puede desplegar utilizando *imshow(img)* de *matplotlib.pyplot*.

Para el caso de *paquete = 2* (OpenCV), la imagen primero debe de ser leída ocupando la función *imread* de la biblioteca *cv2* (OpenCV). Posteriormente, se ocupa *imshow* para crear una nueva ventana que despliegue la imagen leída.

Cuando *paquete = 3* (Scikit-Image), la imagen se lee ocupando la función *imread* del módulo *io* de Scikit. Una vez leída, se utilizan los métodos *imshow* y *show* para desplegar la imagen.

Finalmente, cuando *paquete = 4* (PIL), la imagen se lee ocupando *Image.open(url)* y se despliega con el método *display(img)*. Los métodos de despliegue de las imágenes pueden ser vistos a continuación:

```
def leerImagen(url, paquete):
    if paquete == 1:
        print("matplotlib: \n")
        img = matplotlib.image.imread(url)
        plt.imshow(img)
        plt.show()

    elif paquete == 2:
        print("opencv: \n")
        img = cv2.imread(url)
        cv2.imshow('Imagen', img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    elif paquete == 3:
        print("scikit-image: \n")
        img = io.imread(url)
        io.imshow(img)
        io.show()

    elif paquete == 4:
        print("PIL: ")
        img = PIL.Image.open(url)
        display(img)

    else:
        print("Número de paquete de visualización incorrecto. 1) Matplotlib, 2) OpenCV, 3) Scikit-Image, 4) PIL")
```

Figura 4. Script para leer y desplegar imágenes con Matplotlib, OpenCV, Scikit-Image y PIL.

El script anterior se puede utilizar dentro de un ciclo *for* para leer y desplegar todas las imágenes dentro de una carpeta en específico (en este caso la carpeta de Drive *Baseline_Keller_Shen_Practical*). Lo anterior se muestra en la figura:

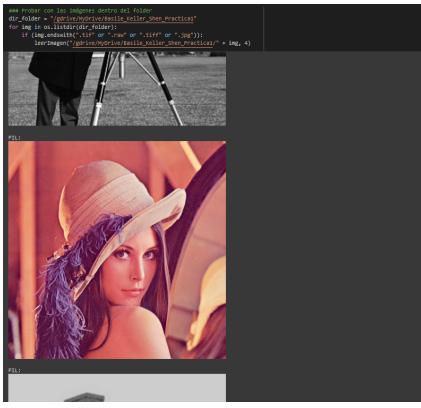


Figura 5. Ejecución del script para leer y desplegar todas las imágenes de un directorio.

III-B. 4.2.Imprimir el tipo de imagen, el tamaño y el tipo de dato

Para este ejercicio de la práctica, se ocupó el folder de Drive *Basile_Keller_Shen_Practical* con varias imágenes de la práctica y, dentro de un ciclo *for*, se busca todos los elementos del folder que sean imágenes (en este caso, que los nombres de archivo terminen con *.tif*, *.jpg*, *.raw*, etc.) y se cargan ocupando *Image.open* de la biblioteca *PIL*. Una vez cargada cada imagen, se ocupa *filename* para imprimir el nombre del archivo, *.size* para desplegar el tamaño de la imagen (ancho, largo) en pixeles, *.format* para imprimir el tipo de imagen (*.tif*, *.jpg*, etc.) y finalmente *.format_description* para obtener una descripción del formato utilizado. La ejecución de este inciso se muestra a continuación:

```
# Para todos los imágenes de un folder
dir_folder = "/gdrive/MyDrive/Basile_Keller_Shen_Practical"
i = 0
for img in os.listdir(dir_folder):
    if (img.endswith(".tif") or ".raw" or ".tiff" or ".jpg"):
        i += 1
        img = PIL.Image.open("/gdrive/MyDrive/Basile_Keller_Shen_Practical/" + img)
        print("Imagen: " + str(i))
        print("Nombre: " + img.filename)
        print("Tamaño: ", img.size)
        print("Tipo de dato: ", img.format)
        print("Descripción del formato: ", img.format_description)
        print("\n")

Imagen 1
Nombre: /gdrive/MyDrive/Basile_Keller_Shen_Practical/rxpie-rodilla.tif
Tamaño: (447, 447)
Tipo de dato: TIFF
Descripción del formato: Adobe TIFF

Imagen 2
Nombre: /gdrive/MyDrive/Basile_Keller_Shen_Practical/lake.tif
Tamaño: (512, 512)
Tipo de dato: TIFF
Descripción del formato: Adobe TIFF

Imagen 3
Nombre: /gdrive/MyDrive/Basile_Keller_Shen_Practical/peppers_color.tif
Tamaño: (512, 512)
Tipo de dato: TIFF
Descripción del formato: Adobe TIFF

Imagen 4
Nombre: /gdrive/MyDrive/Basile_Keller_Shen_Practical/cameraman.tif
Tamaño: (512, 512)
Tipo de dato: TIFF
Descripción del formato: Adobe TIFF

Imagen 5
Nombre: /gdrive/MyDrive/Basile_Keller_Shen_Practical/lena_color_512.tif
Tamaño: (512, 512)
Tipo de dato: TIFF
Descripción del formato: Adobe TIFF

Imagen 6
Nombre: /gdrive/MyDrive/Basile_Keller_Shen_Practical/house.tif
Tamaño: (512, 512)
Tipo de dato: TIFF
```

Figura 6. Ejecución del código que permite conocer el tipo de imagen, el tamaño y tipo de dato.

III-C. 4.3.De las imágenes *lena_color_512.tif*, *peppers_color.tif*. Desarrolla un script con OpenCV y Scikit-Image para cambiar el espacio de color

Para los ejercicios siguientes, se ocuparon funciones que reciben como parámetros el *url* de la imagen, el *paquete* con el cual se quiere cambiar el espacio de color (1=OpenCV, 2=Scikit) y el formato al cual se quiere convertir (ya sea RGB a escala de grises, RGB a YUV o RGB a HSV). Un espacio de color es un sistema de interpretación del color, donde *RGB* es un espacio definido en términos de la intensidad de los colores primarios de la luz (un valor entre 0-255 de R (rojo), un valor entre 0-255 de G (verde) y un valor entre 0-255 de B (azul)), ocupando tres canales distintos (tres matrices) para poder definir los colores en la imagen. La escala de grises ocupa un único canal, donde a cada pixel de la imagen se le asigna un valor entre 0-255 (negro-blanco). El sistema YUV permite separar una imagen en los componentes: luminancia (Y) y crominancia (UV), ocultando de mejor manera los posibles errores de transmisión o imperfecciones de comprensión de imágenes. El modelo HSV contiene tres valores para describir cada imagen (Hue, Saturation y Value / Matiz, Saturación y Valor).

Por separado, las funciones que permiten convertir de un espacio de color a otro, se muestran en las figuras 7, 8 y 9:

III-C1. 4.3.1.RGB a Escala de grises:

Se muestra código e imagen.



Figura 7. Ejecución del código que permite cambiar el espacio de color de RGB a escala de grises.

III-C2. 4.3.2.RGB a YUV: Se muestra código e imagen.

```
#!/usr/bin/python
# This code shows how to change the color space from RGB to YUV. Y is brightness (luminance) and U and V are chrominance (color). It also shows how to save the arrays to be implemented in OpenCV.

def RGBtoYUV(img):
    # Convert image to grayscale
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Convert image to YUV
    img_YUV = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
    # Save image
    cv2.imwrite("lena_color_512.tif", img_YUV)

if __name__ == "__main__":
    # Load image
    img = cv2.imread('lena_color_512.tif')
    # Convert image to YUV
    img_YUV = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
    # Save image
    cv2.imwrite("lena_color_512.tif", img_YUV)
```

Figura 8. Ejecución del código que permite cambiar el espacio de color de RGB a YUV.

III-C3. 4.3.3.RGB a HSV: En una misma función, se muestra cómo convertir de un espacio de color a otro (ver figura 10).

```
#!/usr/bin/python
# This code shows how to change the color space from RGB to HSV. In the same function, it shows how to convert from one color space to another.
# It also shows how to save the arrays to be implemented in OpenCV.

def RGBtoYUV_HSV(url, paquete, formato):
    # Paquete 1: 0 = cv2, 1 = skimage
    # Formato 1 = 0 = gray, 2 = 0 YUV, 3 = 0 HSV
    if paquete == 0:
        if formato == 0:
            img = cv2.imread(url)
            img_GRAY = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            cv2.imshow("Imagen con paleta gray", img_GRAY)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
        elif formato == 1:
            img = skimage.io.imread(url)
            img_YUV = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
            cv2.imshow("Imagen con paleta YUV", img_YUV)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
        elif formato == 2:
            img = cv2.imread(url)
            img_HSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
            cv2.imshow("Imagen con paleta HSV", img_HSV)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
    elif paquete == 1:
        if formato == 0:
            img = skimage.io.imread(url)
            img_GRAY = rgb2gray(img)
            io.imshow(img_GRAY) # visualizador por default
            io.show()
        elif formato == 1:
            img = skimage.io.imread(url)
            img_YUV = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
            io.imshow(img_YUV) # visualizador por default
            io.show()
        elif formato == 2:
            img = skimage.io.imread(url)
            img_HSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
            io.imshow(img_HSV) # visualizador por default
            io.show()

    RGBtoYUV_HSV("gdrive/MyDrive/Basile_Keller_Shen_Practical/lena_color_512.tif", 2, 2)
```

Figura 9. Ejecución del código que permite cambiar el espacio de color de RGB a HSV.

III-D. 4.4.Despliega la paleta de colores de RGB por separado, donde la barra de la derecha con valores sea la paleta de colores.

Para esta sección, se ocupó la biblioteca *matplotlib.pyplot*, en donde primeramente se cargó la imagen y el *cmap* o espacio de color al que pertenece. Posteriormente, se utiliza la función *colorbar* para añadir la paleta de colores de la imagen a un costado del despliegue (ver figura 11), el cual se realiza con *show*. En este caso, se desplegó la imagen con distintas paletas (HSV, Gray, BRG) para poder observar distintos espacios de color (ver figura 12).

```
print("Imagen: ")
Imagen = plt.imread('gdrive/MyDrive/Basile_Keller_Shen_Practical/lena_color_512.tif')
plt.imshow(Imagen)
plt.show()

# gray
print("Imagen con paleta gray")
plt.imshow(Imagen, cmap="gray")
plt.colorbar()
plt.show()

# paleta HSV
print("Imagen con paleta HSV")
plt.imshow(Imagen, cmap="hsv")
plt.colorbar()
plt.show()

# paleta BRG
print("Imagen con paleta BRG")
plt.imshow(Imagen, cmap="brg")
plt.colorbar()
plt.show()
```

Figura 11. Código para desplegar la imagen con su paleta de color.

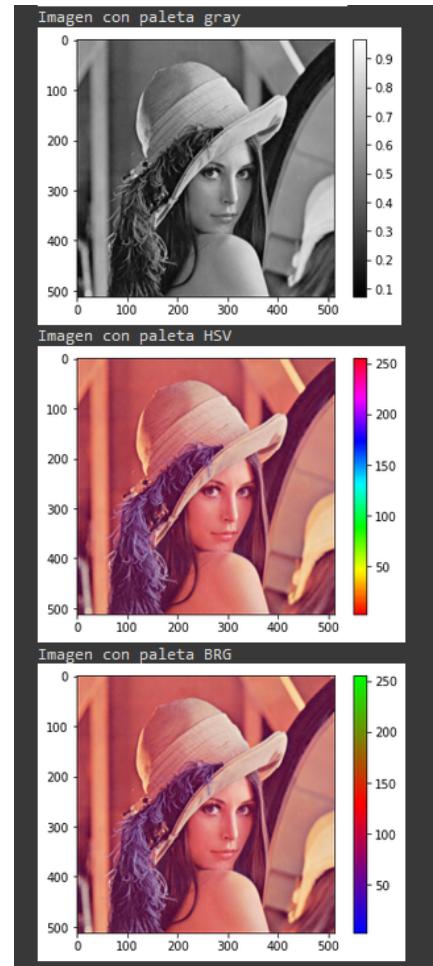


Figura 10. Ejecución del código que permite cambiar el espacio de color de RGB a Gray, HSV y YUV en una misma función.

Figura 12. Ejemplos para distintas paletas de colores.

III-E. 4.5. De una imagen que usted escoja, dejarla en escala de grises y procure que sea igual en renglones y en columnas. Programe una función que realice decimación de una imagen, reduciéndola a la mitad de su tamaño original. Y promediando en grupos de 4 píxeles. Pruebe con su imagen.

La decimación de una imagen es el proceso de reducir su tamaño descartando algunos de sus píxeles. Esto se hace típicamente para reducir la memoria requerida para almacenar o procesar la imagen. El proceso de decimación implica dividir la imagen en una cuadrícula de regiones cuadradas y luego eliminar cada otra fila y columna de píxeles. Esto resulta en una imagen que tiene la mitad del ancho y alto de la original, con una cuarta parte del número de píxeles.

Para este ejercicio, primeramente se realizó un script para convertir la imagen cargada a escala de grises y revisar que el ancho de la imagen sea igual a su alto. En caso de que ésto no se cumpla, se modifica el tamaño de la imagen para cumplir con los requisitos mencionados (ver figura 13).

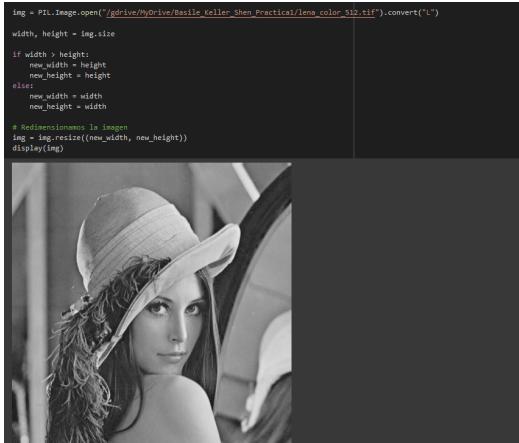


Figura 13. Conversión a escala de grises y redimensionamiento de la imagen para que su alto sea igual al ancho.

Hecho lo anterior, se programaron dos métodos distintos para realizar la decimación. El primero de ellos obtiene como parámetro la imagen en escala de grises obtenida anteriormente y la convierte a un arreglo numérico de *numpy* (arreglo con los valores de escala de grises para cada pixel de la imagen). Luego, se construye un arreglo numérico inicializado con ceros y de la mitad del tamaño del anterior (donde se guardarán los valores para cada pixel luego de la decimación). En un ciclo *for* anidado, se recorre la matriz y, para cada pixel, se promedian los valores de escala de grises de ese pixel con sus vecinos (grupos de 4 píxeles), y el valor promediado se añade al arreglo con la mitad del tamaño creado anteriormente. Finalmente, se construye la imagen a partir del arreglo utilizando *fromarray*. Este método, que en principio debería de ser correcto, nos arrojó resultados peores de los esperados, en cuanto que la resolución de la imagen se perdía demasiado o se desplegaba con errores. Por motivos de tiempo, se decidió ocupar otro método para realizar la decimación, y el primer método quedará pendiente de mejoras en un futuro.

El segundo método utilizado para la decimación ocupa la biblioteca *OpenCV*, específicamente el método *resize*, el cual toma como parámetro la imagen (en formato *array*), la dimensión (en este caso queremos que la dimensión final sea la mitad de la imagen original, y el método de redimensionamiento que utilizará (en este caso *interpolationcv2.INTER_AREA*, que justamente realiza la interpolación de los valores de grupos de píxeles en la imagen (ver figura 14).

Los resultados de las decimación con los dos métodos pueden verse en las figuras 15 y 16.

```
# Forma 1
def decimate_image(img): # acepta imágenes con width > height
    width, height = img.shape[1], img.shape[0]
    # Usar porque queremos enteros de 8 bits que representen un valor en la escala de grises del 0 al 255.
    arr = np.zeros((height // 2, width // 2), dtype=np.uint8)
    arr_4px_mean = np.zeros((width // 2, height // 2), dtype=np.uint8)
    # Recorremos la imagen original y promediamos los valores de cada grupo de 4 píxeles
    for i in range(0, height - 1, 2):
        for j in range(0, width - 1, 4):
            sum_px = arr[i][j] + arr[i][j+1] + arr[i][j+2] + arr[i][j+3]
            arr_4px_mean[j // 2][i // 2] = sum_px // 4
            arr_4px_mean[j // 2][i // 2] = max(0, min(255, arr_4px_mean[j // 2][i // 2]))
    # Convertimos el arreglo en una imagen PIL
    new_img = Image.fromarray(arr_4px_mean)
    return new_img

# Forma 2
def decimate_image(img):
    width, height = img.shape[1], img.shape[0]
    dim = (width // 2, height // 2)
    # img es una imagen en escala de grises que tiene el mismo tamaño que la original y que ocupa interpolación para obtener el valor de cada pixel.
    dec_img = cv2.resize(np.array(img), dim, interpolation=cv2.INTER_AREA)
    cv2.imwrite('Imagen con decimación', dec_img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


```

Figura 14. Métodos utilizados para la decimación de imágenes.



Figura 15. Resultado del primer método de decimación.

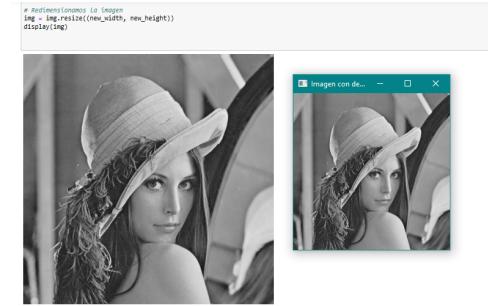


Figura 16. Resultado del segundo método de decimación, donde se aprecia la imagen original convertida a escala de grises (izquierda) y la imagen luego del proceso de decimación (derecha).

III-F. 4.6. Convierte la imagen peppers_color.tif a escala de grises

Para este ejercicio, se ocupó la biblioteca *PIL*, convirtiendo la imagen a escala de grises con el método *convert("L")*.

III-F1. 4.6.1. Recortela de manera que solo quede uno de los pimientos verdes en ese recorte: El recorte se realizó definiendo las coordenadas para la imagen recortada y utilizando el método *crop*, pasando como parámetros dichas coordenadas.

III-F2. 4.6.2. GUÁRDALA EN FORMATO jpg.: Para almacenar una imagen con la que se está trabajando, basta con ocupar el método *save*, pasando como parámetro el nombre y extensión de la imagen como cadena de texto. (ver figura 17).

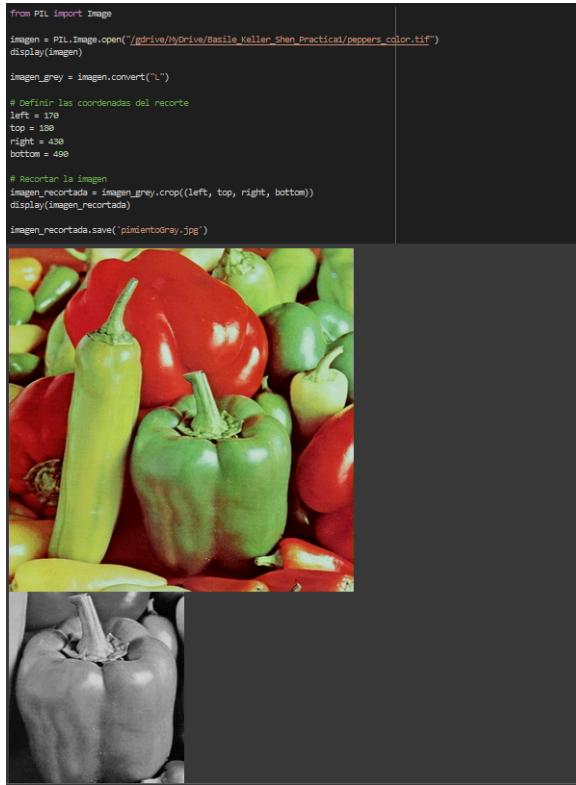


Figura 17. Resultado del recorte de la imagen convertida a escala de grises.

III-G. 4.7. Un formato de imágenes sin ningún tipo de codificación se conoce como formato crudo (RAW). De la imagen "rosa800x600.raw" lea y despliegue la imagen. Tome en cuenta que esta imagen maneja la precisión de integer8 y el tamaño es de 600x800 pixels.

En esta sección, primero se definió la dirección en donde almacenamos la imagen *raw* de la rosa, así como sus medidas (en caso de no tener las medidas correctas, el despliegue de la imagen será incorrecto). Una vez que contamos con la ruta del archivo, lo abrimos con el parámetro *rb*, que indica que leeremos un archivo binario, ya que *raw* generalmente implica lectura de este tipo de archivos, donde cada pixel se representa por un byte o una serie de bytes. Utilizando el método *read* y *Image.frombytes* (al cual pasamos como parámetro el espacio de color, ancho y alto de la imagen y el archivo leído) de *PIL*,

Una vez leída la imagen *raw*, la desplegamos utilizando *show* (figura 18).

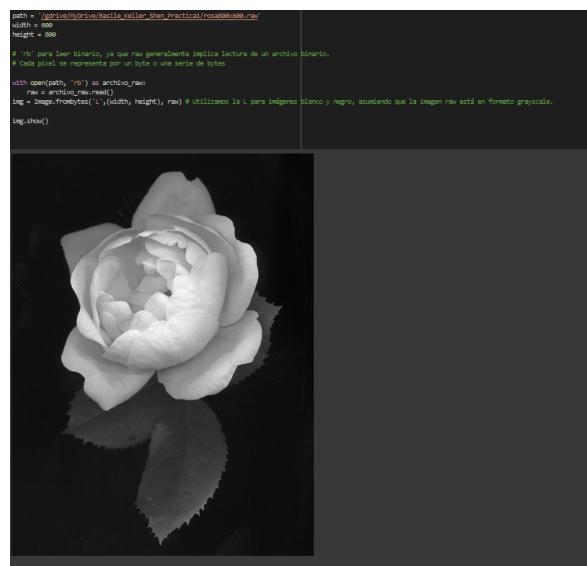


Figura 18. Despliegue de la imagen en formato *raw*.

IV. PARTE B: IRIS SETOSA

En la última actividad de la práctica, estuvimos trabajando con una tabla de datos procesados y ordenados sobre la flor iris. En ella, contábamos con variables como la longitud y ancho de los sépalos y pétalos, y también con la clasificación de cada iris en un tipo específico, es decir, teníamos datos etiquetados.

En esta actividad, nuestro objetivo era llevar a cabo un análisis exploratorio de los datos, que incluyó identificar su tipo, detectar la presencia de datos nulos y calcular estadísticas básicas. Además, utilizamos algunas librerías de Python para visualizar las relaciones entre las variables, así como la dispersión y distribución de los datos.

IV-A. 5.1. Cargue los datos iris en un data frame (pandas) e imprima la descripción de los datos (columnas y renglones), tipo y las 10 primeras filas de los datos. Fuente de datos: <https://archive.ics.uci.edu/ml/datasets/Iris>.

Para cargar un archivo con la extensión *.data* en un data frame, podemos utilizar la función *read_csv* y especificar correctamente la ruta del archivo. En caso de que sea necesario, también podemos indicar otros parámetros, como la falta de encabezados en este caso.

Una vez cargados los datos, podemos obtener su descripción mediante los comandos *describe()* o *info()*. Si queremos conocer los tipos de variables presentes, podemos utilizar el comando *dtypes*. Si deseamos visualizar únicamente los primeros 10 registros, podemos usar el comando *head(10)*, donde el valor 10 indica el número de filas a mostrar.

```

Datos iris en un data frame:           Descripción de los datos (columnas y renglones)

   0   1   2   3   4
0  5.1  3.5  1.4  0.2  Iris-setosa
1  4.9  3.0  1.4  0.2  Iris-setosa
2  4.7  3.2  1.3  0.2  Iris-setosa
3  4.6  3.1  1.5  0.2  Iris-setosa
4  5.0  3.6  1.4  0.2  Iris-setosa
... ... ... ... ...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 # Column Non-Null Count Dtype 
---  -- 
0   147 non-null   float64
1   150 non-null   float64
2   150 non-null   float64
3   149 non-null   float64
4   150 non-null   object  
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
Tipos:
   0      float64
   1      float64
   2      float64
   3      float64
   4      object
dtype: object
las 10 primeras filas de los datos:
   0   1   2   3   4
0  5.1  3.5  1.4  0.2  Iris-setosa
1  4.9  3.0  1.4  0.2  Iris-setosa
2  4.7  3.2  1.3  0.2  Iris-setosa
3  4.6  3.1  1.5  0.2  Iris-setosa
4  5.0  3.6  1.4  0.2  Iris-setosa
5  5.4  3.9  1.7  0.4  Iris-setosa
6  4.6  3.4  1.4  0.3  Iris-setosa
7  5.0  3.4  1.6  0.2  Iris-setosa
8  4.4  2.9  1.4  0.2  Iris-setosa
9  4.9  3.1  1.5  0.1  Iris-setosa

```

Figura 19. DataFrame, describe(), info(), dtypes y head(10)

IV-B. 5.2.Imprima las llaves y el número de filas y de columnas.

Las llaves corresponden a los nombres de las columnas del data frame y podemos acceder a ellas a través de la función `columns`. En cuanto a las dimensiones del data frame, podemos obtener tanto el número de columnas como el número de filas a través de los elementos 1 y 2 de la lista que retorna la función `shape`.

```

Llaves: Int64Index([0, 1, 2, 3, 4], dtype='int64')
Filas: 150
Columnas: 5

```

Figura 20. Llaves, columnas y filas

IV-C. 5.3.Obtenga el número de muestras faltantes o Nan.

Podemos obtener los valores faltantes de las muestras utilizando la función `isnull()`. Para saber la cantidad de valores faltantes, basta con utilizar la función `.sum()` después de aplicar la función anterior.

```

Número de muestras faltantes:
   0    3
   1    0
   2    0
   3    1
   4    0
dtype: int64

```

Figura 21. Muestras faltantes

IV-D. 5.4.Cree un arreglo 2-D de tamaño 5x5 con unos en la diagonal y ceros en el resto. Convierta el arreglo NumPy a una matriz dispersa de ScyPy en formato CRS. Nota: una matriz se considera dispersa cuando el porcentaje de ceros es mayor a 0.5.

En Python, podemos crear una matriz de 5x5 que tenga unos en su diagonal y ceros en el resto utilizando la función `eye(5)` de la biblioteca `numpy`. Para ahorrar espacio de memoria, podemos convertir esta matriz en una matriz dispersa en formato CRS mediante la función `csr_matrix` de la biblioteca `scipy.sparse`. Esto resulta especialmente útil cuando la matriz contiene muchos ceros y deseamos representarla de manera compacta.

```

Arreglo de 2D, 5x5, unos en la diagonal y ceros en el resto
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
Matriz dispersa de ScyPy
(0, 0)      1.0
(1, 1)      1.0
(2, 2)      1.0
(3, 3)      1.0
(4, 4)      1.0

```

Figura 22. Conversión de matriz dispersa

IV-E. 5.5.Muestre estadísticas básicas como percentil, media, mínimo, máximo y desviación estándar de los datos. Use describe para ello. Imprima sólo la media y la desviación estándar.

Podemos obtener fácilmente estadísticas básicas de los datos en un dataframe utilizando la función `describe()`. Si es necesario, podemos indicar específicamente cuáles estadísticas deseamos visualizar mediante la función `loc()`.

```

Estadística básicas: sólo la media y la desviación estándar
   0   1   2   3
mean  5.857823  3.054000  3.758667  1.196644
std   0.828013  0.433594  1.764420  0.765331

```

Figura 23. Estadísticas básicas

IV-F. 5.6.Obtenga el número de muestras para cada clase.

Podemos obtener el número de muestras por clase utilizando la función `value_counts()`.

Clases:	
Iris-setosa	50
Iris-versicolor	50
Iris-virginica	50
Name: 4, dtype: int64	

Figura 24. Número de muestras por clase

IV-G. 5.7.Añada un encabezado a los datos usando los nombres en `iris.names` y repita el ejercicio anterior.

La información de los encabezados se encuentra en el archivo con extensión `.name`, y podemos acceder a ella mediante el bloque de instrucciones `with open()` de Python. Una vez analizado el contenido, podemos agregar estos encabezados a nuestro dataframe utilizando la función `columns`.

```
datos.columns=['sepal length in cm', 'sepal width in cm', 'petal length in cm', 'petal width in cm', 'class']
print("\nClases:\n", datos['class'].value_counts())
Clases:
Iris-setosa      50
Iris-versicolor  50
Iris-virginica   50
Name: class, dtype: int64
```

Figura 25. Añadir encabezados

IV-H. 5.8.Imprima las diez primeras filas y las dos primeras columnas del data frame usando los índices de las columnas.

Podemos acceder a las primeras filas o columnas de un dataframe mediante la función `iloc`. En este caso, podemos utilizar los parámetros `iloc[:10, :2]` para acceder a las primeras 10 filas y las primeras 2 columnas.

```
Datos:
    sepal length in cm  sepal width in cm
0                  5.1              3.5
1                  4.9              3.0
2                  4.7              3.2
3                  4.6              3.1
4                  5.0              3.6
5                  5.4              3.9
6                  4.6              3.4
7                  5.0              3.4
8                  4.4              2.9
9                  4.9              3.1
```

Figura 26. Primeras 10 filas y 2 columnas

IV-I. 5.9.Cree una gráfica de barras que muestre la media, mínimo y máximo de todos los datos.

Podemos obtener la media, el mínimo y el máximo de los datos utilizando la función `describe()`. Una vez que hayamos obtenido estos datos en forma de lista, podemos manipular la gráfica de barras con las funciones de Matplotlib de Python.

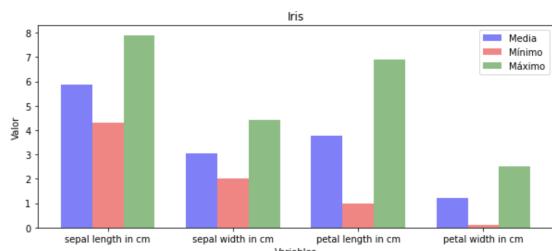


Figura 27. Media, mínimo y máximo de los datos

IV-J. 5.10.Muestre la frecuencia de las tres especies como una gráfica de pastel.

Ya hemos obtenido la frecuencia de las tres especies utilizando la función `value_counts()`. Para representar esta información en una gráfica de pastel, solo necesitamos aplicar la función `plot.pie`, indicando los parámetros adecuados.

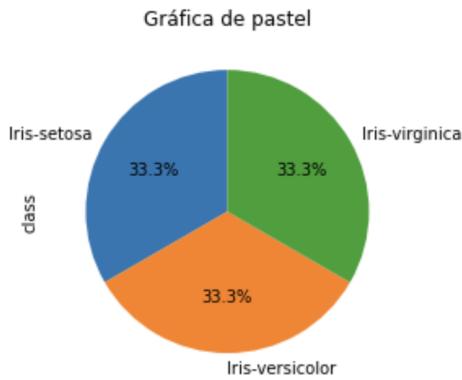


Figura 28. Frecuencia de las tres especies

IV-K. 5.11.Cree una gráfica que muestre la relación entre la longitud y ancho del sépalo de las tres especies conjuntamente.

Calculamos la relación entre la longitud y el ancho del sépalo de los iris sumando la longitud de todos los registros de una especie y dividiéndola por la suma del ancho de todos los registros de la misma especie.

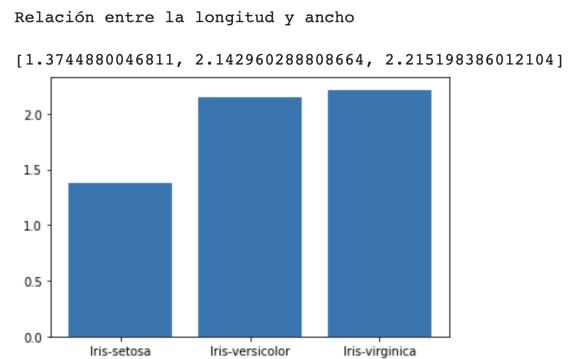


Figura 29. Relación entre longitud y ancho

IV-L. 5.12.Obtenga los histogramas de las variables SepalLength, SepalWidth, PetalLength y PetalWidth.

En Pandas, existe una función llamada `hist` que nos permite obtener el histograma de todas las variables numéricas de un dataframe. (ver figura 30).

IV-M. 5.13.Cree gráficas de dispersión usando `pairplot` de `seaborn` y muestre con distintos colores las tres especies en las gráficas de dispersión.

La función `pairplot` de Seaborn permite crear automáticamente una gráfica de dispersión con diferentes colores para cada clase. (ver figura 31).

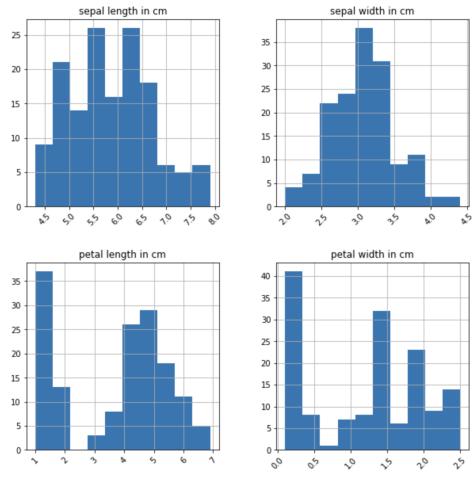


Figura 30. Histograma

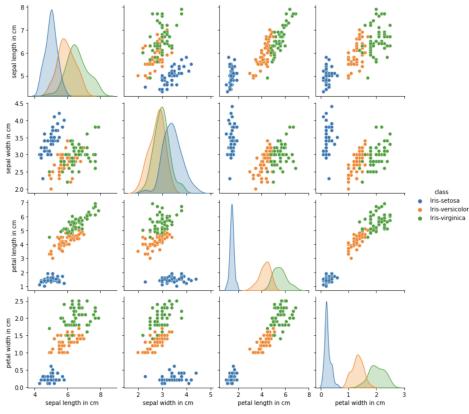


Figura 31. Graficas de dispersión con pairplot

IV-N. 5.14. Cree una gráfica usando joinplot de seaborn para mostrar la dispersión entre la longitud y ancho del sépalo y las distribuciones de estas dos variables.

Una alternativa para visualizar la dispersión de datos es la función *joinplot* de Seaborn.

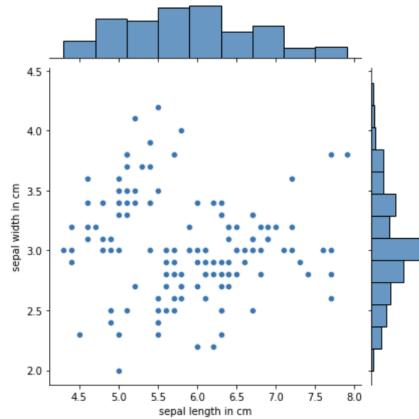


Figura 32. Graficas de dispersión con jointplot

IV-N. 5.15. Repita el ejercicio anterior, pero esta vez usando joinplot con kind="hex".

Al agregar el parámetro *kind='hex'* en el ejercicio anterior, se produce un cambio en la gráfica, reemplazando la gráfica de dispersión de puntos por una gráfica de hexágonos. Esta modificación permite tener una representación visual más clara de la distribución de los datos cuando hay una gran cantidad de puntos, ya que los valores se agrupan en hexágonos y se colorean en función de la densidad de puntos que caen en esa área.

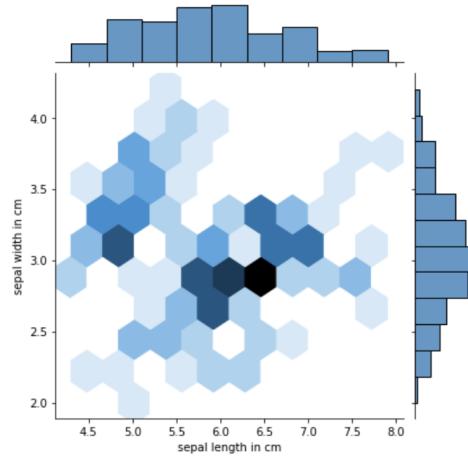


Figura 33. Graficas de hexágonos

V. CONCLUSIONES

A través de esta práctica se buscó que los alumnos aprendieran acerca del manejo de imágenes al hacer uso de diversas herramientas permitieran realizar un correcto análisis de una imagen, siendo esta una práctica introductoria para la materia donde los alumnos se familiarizaran con diversas bibliotecas de Python, así como funciones que les permitieran mejorar sus conocimientos y habilidades con el uso de este tipo de archivos.

Como objetivo principal de la práctica se buscaba que el alumno conociera las diversas formas de desplegar una imagen en distintos formatos, así como algunas manipulaciones básicas para el procesamiento de imágenes. De esta forma consideramos que el objetivo de la práctica se cumplió debido a que logramos realizar con éxito los ejercicios indicados dentro de la práctica, donde a pesar de haber contado con algunas complicaciones para el uso de algunas funciones, se logró llegar a soluciones que nos permitieran obtener los resultados esperados.

Esta práctica se dividió en dos secciones, la parte A donde se realizó el manejo de imágenes y la parte B donde se trabajó Iris Setosa.

Dentro de la primera parte, la parte A, aprendimos a realizar el despliegue de imágenes a partir de diversos paquetes, ya sea *Matplotlib*, *OpenCV*, *Scikit-Image* o *PIL*, se reconocieron los métodos para la impresión del tipo de imagen, su tamaño, su tipo de dato, así como la manipulación del color a partir del

cambio de espacio de color; se logró desplegar una imagen que contuviera distintas paletas de colores y el mayor reto se apreció en el ejercicio que nos pedía trabajar con la decimación de la imagen. Para este ejercicio, la dificultad se encontró en el hecho en que no sabíamos con exactitud lo que el proceso de decimación significaba, por lo tanto tuvimos que investigar y a partir de la información obtenida trabajamos con dos métodos de decimación, esto debido a que no encontramos una función que nos permitiera realizar este proceso directamente. El primer método utilizado fue con el que obtuvimos resultados diferentes a los esperados, ya que en lugar de únicamente bajar la resolución de la imagen, nuestro método la distorsionó demasiado, incluso llegando a causar que las imágenes decimadas tuvieran errores en el despliegue (por ejemplo, objetos que se borraron o que se movieron de posición inesperadamente). Por motivos de tiempo, se decidió ocupar otro método para realizar la decimación (el cual arrojó los resultados esperados, como fue presentado en apartados previos de este documento), y el primer método quedará pendiente de mejoras en un futuro.

Para la segunda parte de la práctica, la parte B, trabajamos con Iris Setosa, donde al tener una tabla de datos procesados y ordenados sobre esta flor pudimos realizar la graficación de diversos valores o relaciones entre el ancho y largo de los pétalos y los sépalos. Lo primero que se realizó fue cargar la tabla de datos en *Python*, para posteriormente realizar un análisis rápido de la conformación de la tabla, por filas y columnas. Se obtuvieron datos estadísticos acerca de los valores contenidos dentro de la tabla y posteriormente se procedió a realizar un análisis por medio de elementos visuales, donde trabajamos con una gráfica de barras para apreciar los valores máximos, mínimos y promedio de los datos contenidos en nuestra tabla, una gráfica de pastel para mostrar la distribución de los datos con respecto a las especies de la planta, un histograma y otra gráfica de barras para apreciar la relación entre el valor del ancho y alto de los sépalos de las especies, así como gráficas de dispersión para reconocer estas mismas variaciones y de esta manera reconocer con qué combinación de variables apreciamos una mayor dispersión de los datos que permitieran realizar la diferenciación de los tres tipos de especies. Finalmente se trabajó con una gráfica de dispersión entre la longitud y el largo del sépalo así como la distribución de estas variables.

En conclusión, esta fue una primera práctica introductoria que nos permitió reconocer las bibliotecas, métodos y funciones con los cuales podemos realizar la manipulación de imágenes, así como la forma mediante la cual podemos realizar un análisis tras obtener un conjunto de datos a partir de imágenes que nos encontramos analizando, de esta manera, podemos apreciar patrones que nos permitan realizar un proceso de clasificación, en este caso, el proceso de clasificación se podría realizar entre las especies de plantas a partir de las características de sus pétalos y sépalos. Consideramos que fue una práctica muy completa que nos brindó herramientas útiles que utilizaremos con regularidad a lo largo de futuras prácticas.

REFERENCIAS

- [1] Adobe, “Archivos jpeg. <https://www.adobe.com/mx/creativecloud/file-types/image/raster/jpeg-file.html>,” 2023.
- [2] ——, “Archivos raw. <https://www.adobe.com/mx/creativecloud/file-types/image/raw.html>,” 2023.
- [3] ——, “Archivos png. <https://www.adobe.com/mx/creativecloud/file-types/image/raster/png-file.html>,” 2023.
- [4] ——, “Archivos tiff. <https://www.adobe.com/mx/creativecloud/file-types/image/raster/tiff-file.html>,” 2023.
- [5] H. A. C. Historia, “Imágenes binarias. <https://www.hisour.com/es/binary-image-27221/>,” 2023.
- [6] MathWorks, “Tipos de imagen. https://es.mathworks.com/help/matlab/creating_plots/imagetypes.html,” 2023.