

**UNIVERSIDAD AUTONOMA DE NUEVO LEON**

**FACULTAD DE INGENIERIA MECANICA Y ELECTRICA**

**Nombre: Roberto Erick Aguilar Morales**

**Matricula: 1871004**

**Carrera: Ingeniero en Tecnologías del Software**

**4.– Umbrales**

**Materia: VISION COMPUTACIONAL LABORATORIO**

**Docente:** **RAYMUNDO SAID ZAMORA PEQUEÑO**

**Hora: N1-N2 Días: Miércoles**

**Fecha: 21/08/24**

**Objetivo**

Utilizar umbrales para discretizar una imagen.

* Varianza
* Entropía
* Valle global

Utilice los umbrales para generar una imagen con 3, 4, 8 y 16 tonos

**Marco teórico**

**Procesamiento de Imágenes**

El procesamiento de imágenes es una disciplina que se ocupa de la manipulación y análisis de imágenes digitales a través de computadoras. Se emplea en diversas áreas, desde la medicina y la astronomía, hasta la inteligencia artificial y el control de calidad industrial. Dentro de esta área, el propósito principal es extraer información útil de las imágenes o transformarlas en representaciones más adecuadas para su análisis.

El primer paso en el procesamiento de imágenes digitales suele ser la conversión de la imagen a escala de grises, lo que simplifica el análisis y permite un tratamiento más eficiente de los píxeles. Las imágenes en escala de grises son representaciones bidimensionales donde cada píxel tiene un valor de intensidad que oscila entre 0 (negro) y 255 (blanco).

**Umbralización de Imágenes**

La umbralización es una técnica fundamental en el procesamiento de imágenes que consiste en transformar una imagen de múltiples tonos a una imagen binaria (blanco y negro) o con un número reducido de tonos. Se utiliza principalmente para resaltar o segmentar regiones de interés en una imagen, separando los objetos del fondo. Esto se logra mediante la selección de uno o más **umbrales** que determinan los puntos de corte de los niveles de intensidad de los píxeles.

**Umbrales Globales y Locales**

Un umbral global utiliza un único valor de umbral para toda la imagen, mientras que un umbral local se ajusta de manera dinámica según diferentes regiones de la imagen. En este trabajo nos enfocamos en métodos globales que establecen un único umbral para segmentar la imagen.

**Métodos de Umbralización Automática**

Para determinar de manera automática un umbral adecuado para la segmentación de imágenes, se han desarrollado diversos métodos que consideran aspectos estadísticos y de entropía de la distribución de los niveles de gris en la imagen.

**Umbralización por Entropía**

El método de Umbralización por Entropía utiliza la teoría de la información para maximizar la entropía de la imagen segmentada en dos clases: los píxeles de intensidad baja y los de intensidad alta. El objetivo es encontrar el umbral que mejor separa la imagen en términos de la cantidad de información que contienen ambas clases. Este método es particularmente útil cuando las intensidades de los píxeles en las regiones de interés y fondo son comparables, y se busca maximizar la cantidad de información en la segmentación.

La entropía se define como: donde es la probabilidad de ocurrencia del nivel de intensidad ​. El umbral óptimo es el que maximiza la entropía de la imagen segmentada.

**Umbralización por Grupo de Varianza**

La Umbralización por Grupo de Varianza se basa en la minimización de la varianza intragrupo y la maximización de la varianza intergrupo, lo que significa que busca el umbral que mejor separa los píxeles en dos clases: los de baja intensidad y los de alta intensidad. Este método es una implementación del criterio de Otsu, que es un algoritmo ampliamente utilizado en la umbralización automática.

La fórmula para calcular la varianza entre clases es: donde y son las proporciones de los píxeles en las dos clases, y son las medias de las intensidades en las dos clases, y ​es la media total de la imagen.

**Aproximación de Valle Global**

La Aproximación de Valle Global se basa en la forma del histograma de la imagen. Este método busca los puntos de inflexión en el histograma para determinar un umbral adecuado. Los valles en el histograma representan las transiciones entre regiones claras y oscuras en la imagen, y el umbral óptimo se encuentra en el valle más profundo, lo que indica una buena separación entre el fondo y los objetos.

Este enfoque es particularmente útil cuando el histograma tiene una distribución bimodal o multimodal, lo que facilita la identificación de un umbral que se ajuste adecuadamente a la transición entre las regiones de la imagen.

**Reducción de Tonos en Imágenes**

La reducción de tonos es el proceso de simplificar la representación de una imagen mediante la disminución de los niveles de gris a un número específico de tonos. Este proceso es útil para simplificar la segmentación y el análisis de imágenes, especialmente cuando la imagen original tiene mucho ruido o variabilidad en los tonos.

En este trabajo, hemos aplicado la reducción de tonos a **3, 4, 8 y 16** tonos, para cada uno de los métodos de umbralización presentados (Entropía, Grupo de Varianza y Valle Global). Este enfoque permite comparar el rendimiento de los métodos y la calidad de la segmentación en diferentes niveles de tonalidades.

**Introducción**

Los 3 programas de umbrales semi-adaptativos llamados grupo varianza, umbral por entropía y aproximación por valle global están diseñados para encontrar el umbral optimo en una misma imagen, a su vez esta se dividirá en 3, 4, 8 y 16 tonos, por lo que su mayor cambio será su método principal para dividir los colores dependiendo su método diferente, aclarando esto podemos explicar de manera mas general la explicación del código de 3 programas en un solo documento

**Umbral por Entropía**

El método de Umbral por Entropía utiliza la teoría de la información para encontrar el umbral óptimo que maximiza la entropía entre las dos clases de píxeles (claros y oscuros). A continuación, se muestra el fragmento de código responsable del cálculo del umbral óptimo basado en la entropía:

def calcular\_entropia(histograma, total\_pixeles):

    # Cálculo de la entropía

    probabilidad = histograma / total\_pixeles

    probabilidad = probabilidad[probabilidad > 0]  # Filtrar ceros

    entropia = -np.sum(probabilidad \* np.log2(probabilidad))

    return entropia

def encontrar\_mejor\_umbral\_entropia(imagen):

    # Calcular el histograma de la imagen

    histograma = calcular\_histograma(imagen)

    total\_pixeles = imagen.size

    # Variables para encontrar el mejor umbral

    mejor\_entropia = 0

    mejor\_umbral = 0

Este fragmento implementa la búsqueda del umbral óptimo al calcular la entropía de las dos clases de la imagen. Este método es eficiente cuando se busca maximizar la información contenida en la segmentación de la imagen.

**Grupo de Varianza**

El método de Grupo de Varianza busca minimizar la varianza dentro de las clases y maximizar la varianza entre clases. Este enfoque es especialmente útil en imágenes bimodales, donde los niveles de gris se agrupan en dos clases claramente definidas.

def calcular\_grupo\_varianza(histograma, total\_pixeles):

    # Cálculo de varianza entre los grupos

    suma\_total = np.sum([i \* histograma[i] for i in range(256)])

    suma\_b = 0

    w\_b = 0

    varianza\_max = 0

    mejor\_umbral = 0

    for umbral in range(256):

        w\_b += histograma[umbral]

        w\_f = total\_pixeles - w\_b

        if w\_b == 0 or w\_f == 0:

            continue

        suma\_b += umbral \* histograma[umbral]

        m\_b = suma\_b / w\_b if w\_b != 0 else 0

        m\_f = (suma\_total - suma\_b) / w\_f if w\_f != 0 else 0

        varianza\_entre\_clases = w\_b \* w\_f \* (m\_b - m\_f) \*\* 2

        if varianza\_entre\_clases > varianza\_max:

            varianza\_max = varianza\_entre\_clases

            mejor\_umbral = umbral

    return mejor\_umbral

Este código identifica el umbral óptimo basado en la varianza entre clases. La función calcular\_grupo\_varianza es clave para evaluar el umbral que maximiza la separación entre las clases de la imagen.

**Aproximación de Valle Global**

La Aproximación de Valle Global identifica los puntos de transición entre objetos y fondo en una imagen basada en los valles del histograma. El umbral óptimo se encuentra en el valle más profundo, lo que sugiere una buena separación entre las dos clases de píxeles.

def calcular\_desviacion\_estandar(imagen):

    # Cálculo de la desviación estándar

    return np.std(imagen)

def encontrar\_picos(histograma):

    # Encontrar picos en el histograma (máximos locales)

    picos = []

    for i in range(1, len(histograma) - 1):

        if histograma[i] > histograma[i - 1] and histograma[i] > histograma[i + 1]:

            picos.append(i)

    return picos

def calcular\_valle\_global(imagen):

    # Obtener el histograma

    histograma = calcular\_histograma(imagen)

    total\_pixeles = imagen.size

    # Calcular el umbral utilizando grupo varianza

    mejor\_umbral = calcular\_grupo\_varianza(histograma, total\_pixeles)

    # Calcular la desviación estándar

    desviacion\_estandar = calcular\_desviacion\_estandar(imagen)

    # Encontrar los picos del histograma

    picos = encontrar\_picos(histograma)

    return mejor\_umbral, desviacion\_estandar, picos

Este fragmento de código busca los valles en el histograma de la imagen y selecciona el valle más profundo como el umbral óptimo para la segmentación. Este método es útil cuando el histograma de la imagen presenta una clara distribución multimodal.

**Generación de Imágenes con Diferentes Tonos**

Para los tres métodos, se ha implementado una funcionalidad que permite aplicar los umbrales obtenidos para generar imágenes con **3, 4, 8 y 16 tonos**. A continuación, se muestra el fragmento común que se utiliza en los tres programas:

ef aplicar\_tonos(imagen, num\_tonos):

    # Definir los umbrales

    max\_val = 255

    umbrales = np.linspace(0, max\_val, num\_tonos + 1)

    imagen\_tonos = np.zeros\_like(imagen)

    # Asignar tonos basados en los umbrales definidos

    for i in range(num\_tonos):

        lower\_bound = umbrales[i]

        upper\_bound = umbrales[i + 1]

        imagen\_tonos[(imagen >= lower\_bound) & (imagen < upper\_bound)] = int((lower\_bound + upper\_bound) / 2)

    return imagen\_tonos

Este código es responsable de reducir los tonos de la imagen a 3, 4, 8 o 16 niveles de gris, lo que simplifica la representación de la imagen y facilita su análisis en diferentes escalas de tonos.

**Visualización del Histograma y Guardado de Imágenes**

Finalmente, en cada uno de los métodos, se ha implementado una sección para visualizar los histogramas y guardar las matrices resultantes en archivos CSV, lo cual permite un análisis cuantitativo de los resultados:

# Mostrar el histograma y las imágenes umbralizadas

    plt.figure(figsize=(15, 10))

    # Mostrar el histograma

    plt.subplot(2, len(tonos), 1)

    plt.bar(range(256), histograma, color='black')

    plt.title('Histograma Original')

    plt.xlim([0, 255])

    # Mostrar las imágenes umbralizadas

    for idx, num\_tonos in enumerate(tonos):

        plt.subplot(2, len(tonos), idx + 2)

        plt.imshow(imagenes\_tonos[idx], cmap='gray')

        # Crear nombre de archivo dinámicamente usando f-string

        nombre\_archivo = f"imagen\_{num\_tonos}\_tonos.csv"

        np.savetxt(nombre\_archivo, imagenes\_tonos[idx], delimiter=',', fmt='%d')

        plt.title(f'{num\_tonos} tonos')

    for num\_tonos in tonos:

        imagen\_tonos = aplicar\_tonos(imagen, num\_tonos)

        # Guardar la imagen umbralizada con diferentes tonos

        cv2.imwrite(f'imagen\_{num\_tonos}\_tonos.png', imagen\_tonos)

    # Mostrar las imágenes con el histograma

    plt.tight\_layout()

    plt.show()

Este fragmento de código es común en los tres métodos y permite visualizar las imágenes resultantes y guardar las matrices en formato CSV para su posterior análisis.

**Cálculos y Resultados**

**P04UmbralAproximacionDeValleGlobal.py**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage import io

def calcular\_histograma(imagen):

    # Calcula el histograma de la imagen en escala de grises

    histograma, \_ = np.histogram(imagen.ravel(), bins=256, range=(0, 256))

    return histograma

def calcular\_grupo\_varianza(histograma, total\_pixeles):

    # Cálculo de varianza entre los grupos

    suma\_total = np.sum([i \* histograma[i] for i in range(256)])

    suma\_b = 0

    w\_b = 0

    varianza\_max = 0

    mejor\_umbral = 0

    for umbral in range(256):

        w\_b += histograma[umbral]

        w\_f = total\_pixeles - w\_b

        if w\_b == 0 or w\_f == 0:

            continue

        suma\_b += umbral \* histograma[umbral]

        m\_b = suma\_b / w\_b if w\_b != 0 else 0

        m\_f = (suma\_total - suma\_b) / w\_f if w\_f != 0 else 0

        varianza\_entre\_clases = w\_b \* w\_f \* (m\_b - m\_f) \*\* 2

        if varianza\_entre\_clases > varianza\_max:

            varianza\_max = varianza\_entre\_clases

            mejor\_umbral = umbral

    return mejor\_umbral

def calcular\_desviacion\_estandar(imagen):

    # Cálculo de la desviación estándar

    return np.std(imagen)

def encontrar\_picos(histograma):

    # Encontrar picos en el histograma (máximos locales)

    picos = []

    for i in range(1, len(histograma) - 1):

        if histograma[i] > histograma[i - 1] and histograma[i] > histograma[i + 1]:

            picos.append(i)

    return picos

def calcular\_valle\_global(imagen):

    # Obtener el histograma

    histograma = calcular\_histograma(imagen)

    total\_pixeles = imagen.size

    # Calcular el umbral utilizando grupo varianza

    mejor\_umbral = calcular\_grupo\_varianza(histograma, total\_pixeles)

    # Calcular la desviación estándar

    desviacion\_estandar = calcular\_desviacion\_estandar(imagen)

    # Encontrar los picos del histograma

    picos = encontrar\_picos(histograma)

    return mejor\_umbral, desviacion\_estandar, picos

def aplicar\_tonos(imagen, num\_tonos):

    # Definir los umbrales

    max\_val = 255

    umbrales = np.linspace(0, max\_val, num\_tonos + 1)

    imagen\_tonos = np.zeros\_like(imagen)

    # Asignar tonos basados en los umbrales definidos

    for i in range(num\_tonos):

        lower\_bound = umbrales[i]

        upper\_bound = umbrales[i + 1]

        imagen\_tonos[(imagen >= lower\_bound) & (imagen < upper\_bound)] = int((lower\_bound + upper\_bound) / 2)

    return imagen\_tonos

# Cargar la imagen

imagen\_a\_color = cv2.imread("imagen a color.png")

if imagen\_a\_color is None:

    print("La imagen no se cargó correctamente, favor de verificar la ruta")

else:

    # Convertir la imagen a escala de Grises

    imagen = cv2.cvtColor(imagen\_a\_color, cv2.COLOR\_BGR2GRAY)

    # Calcular umbrales

    mejor\_umbral, desviacion\_estandar, picos = calcular\_valle\_global(imagen)

    # Obtener el histograma de la imagen original

    histograma = calcular\_histograma(imagen)

    # Aplicar diferentes tonos

    tonos = [3, 4, 8, 16]

    imagenes\_tonos = [aplicar\_tonos(imagen, num\_tonos) for num\_tonos in tonos]

    # Mostrar el histograma y las imágenes umbralizadas

    plt.figure(figsize=(15, 10))

    # Mostrar el histograma

    plt.subplot(2, len(tonos), 1)

    plt.bar(range(256), histograma, color='black')

    plt.title('Histograma Original')

    plt.xlim([0, 255])

    # Mostrar las imágenes umbralizadas

    for idx, num\_tonos in enumerate(tonos):

        plt.subplot(2, len(tonos), idx + 2)

        plt.imshow(imagenes\_tonos[idx], cmap='gray')

        # Crear nombre de archivo dinámicamente usando f-string

        nombre\_archivo = f"imagen\_{num\_tonos}\_tonos.csv"

        np.savetxt(nombre\_archivo, imagenes\_tonos[idx], delimiter=',', fmt='%d')

        plt.title(f'{num\_tonos} tonos')

    for num\_tonos in tonos:

        imagen\_tonos = aplicar\_tonos(imagen, num\_tonos)

        # Guardar la imagen umbralizada con diferentes tonos

        cv2.imwrite(f'imagen\_{num\_tonos}\_tonos.png', imagen\_tonos)

    # Mostrar las imágenes con el histograma

    plt.tight\_layout()

    plt.show()

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

Pantalla de computadora

Descripción generada automáticamente

**P04UmbralGrupoVarianza.py**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage import io

def calcular\_histograma(imagen):

    # Calcula el histograma de la imagen en escala de grises

    histograma, \_ = np.histogram(imagen.ravel(), bins=256, range=(0, 256))

    return histograma

def calcular\_grupo\_varianza(histograma, total\_pixeles):

    # Cálculo de varianza entre los grupos

    suma\_total = np.sum([i \* histograma[i] for i in range(256)])

    suma\_b = 0

    w\_b = 0

    varianza\_max = 0

    mejor\_umbral = 0

    for umbral in range(256):

        w\_b += histograma[umbral]

        w\_f = total\_pixeles - w\_b

        if w\_b == 0 or w\_f == 0:

            continue

        suma\_b += umbral \* histograma[umbral]

        m\_b = suma\_b / w\_b if w\_b != 0 else 0

        m\_f = (suma\_total - suma\_b) / w\_f if w\_f != 0 else 0

        varianza\_entre\_clases = w\_b \* w\_f \* (m\_b - m\_f) \*\* 2

        if varianza\_entre\_clases > varianza\_max:

            varianza\_max = varianza\_entre\_clases

            mejor\_umbral = umbral

    return mejor\_umbral

def aplicar\_tonos(imagen, num\_tonos):

    # Definir los umbrales

    max\_val = 255

    umbrales = np.linspace(0, max\_val, num\_tonos + 1)

    imagen\_tonos = np.zeros\_like(imagen)

    # Asignar tonos basados en los umbrales definidos

    for i in range(num\_tonos):

        lower\_bound = umbrales[i]

        upper\_bound = umbrales[i + 1]

        imagen\_tonos[(imagen >= lower\_bound) & (imagen < upper\_bound)] = int((lower\_bound + upper\_bound) / 2)

    return imagen\_tonos

# Cargar la imagen

imagen\_a\_color = cv2.imread("imagen a color.png")

if imagen\_a\_color is None:

    print("La ruta de la imagen no es correcta, porfavor ingrese la ruta adecuada")

else:

    # Convertrir la imagen a escala de grises

    imagen = cv2.cvtColor(imagen\_a\_color,cv2.COLOR\_BGR2GRAY)

    # Calcular el histograma de la imagen original

    histograma = calcular\_histograma(imagen)

    total\_pixeles = imagen.size

    # Calcular el mejor umbral usando el método de grupo varianza

    mejor\_umbral = calcular\_grupo\_varianza(histograma, total\_pixeles)

    # Aplicar diferentes tonos (3, 4, 8, 16)

    tonos = [3, 4, 8, 16]

    imagenes\_tonos = [aplicar\_tonos(imagen, num\_tonos) for num\_tonos in tonos]

    # Mostrar el histograma y las imágenes umbralizadas

    plt.figure(figsize=(15, 10))

    # Mostrar el histograma original

    plt.subplot(2, len(tonos), 1)

    plt.bar(range(256), histograma, color='black')

    plt.title('Histograma Original')

    plt.xlim([0, 255])

    # Mostrar las imágenes umbralizadas

    for idx, num\_tonos in enumerate(tonos):

        plt.subplot(2, len(tonos), idx + 2)

        plt.imshow(imagenes\_tonos[idx], cmap='gray')

        # Crear nombre de archivo dinámicamente usando f-string

        nombre\_archivo = f"imagen\_{num\_tonos}\_tonos.csv"

        np.savetxt(nombre\_archivo, imagenes\_tonos[idx], delimiter=',', fmt='%d')

        plt.title(f'{num\_tonos} tonos')

    for num\_tonos in tonos:

        imagen\_tonos = aplicar\_tonos(imagen, num\_tonos)

        # Guardar la imagen umbralizada con diferentes tonos

        cv2.imwrite(f'imagen\_{num\_tonos}\_tonos.png', imagen\_tonos)

    # Mostrar las imágenes y el histograma

    plt.tight\_layout()

    plt.show()

Diagrama

Descripción generada automáticamente

Pantalla de computadora

Descripción generada automáticamente

**P04UmbralPorEntropia.py**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage import io

def calcular\_histograma(imagen):

    # Calcula el histograma de la imagen en escala de grises

    histograma, \_ = np.histogram(imagen.ravel(), bins=256, range=(0, 256))

    return histograma

def calcular\_entropia(histograma, total\_pixeles):

    # Cálculo de la entropía

    probabilidad = histograma / total\_pixeles

    probabilidad = probabilidad[probabilidad > 0]  # Filtrar ceros

    entropia = -np.sum(probabilidad \* np.log2(probabilidad))

    return entropia

def encontrar\_mejor\_umbral\_entropia(imagen):

    # Calcular el histograma de la imagen

    histograma = calcular\_histograma(imagen)

    total\_pixeles = imagen.size

    # Variables para encontrar el mejor umbral

    mejor\_entropia = 0

    mejor\_umbral = 0

    # Calcular la entropía para cada posible umbral

    for umbral in range(256):

        hist\_bajo = histograma[:umbral]

        hist\_alto = histograma[umbral:]

        # Cálculo de la entropía para las dos clases

        entropia\_bajo = calcular\_entropia(hist\_bajo, total\_pixeles)

        entropia\_alto = calcular\_entropia(hist\_alto, total\_pixeles)

        # Entropía total

        entropia\_total = entropia\_bajo + entropia\_alto

        if entropia\_total > mejor\_entropia:

            mejor\_entropia = entropia\_total

            mejor\_umbral = umbral

    return mejor\_umbral

def aplicar\_tonos(imagen, num\_tonos):

    # Definir los umbrales

    max\_val = 255

    umbrales = np.linspace(0, max\_val, num\_tonos + 1)

    imagen\_tonos = np.zeros\_like(imagen)

    # Asignar tonos basados en los umbrales definidos

    for i in range(num\_tonos):

        lower\_bound = umbrales[i]

        upper\_bound = umbrales[i + 1]

        imagen\_tonos[(imagen >= lower\_bound) & (imagen < upper\_bound)] = int((lower\_bound + upper\_bound) / 2)

    return imagen\_tonos

# Cargar la imagen

imagen\_a\_color = cv2.imread("imagen a color.png")

if imagen\_a\_color is None:

    print("La imagen no se cargo correctamente, favor de verificar su ruta")

else:

    imagen = cv2.cvtColor(imagen\_a\_color,cv2.COLOR\_BGR2GRAY)

    # Calcular el histograma de la imagen original

    histograma = calcular\_histograma(imagen)

    # Calcular el mejor umbral usando el método de entropía

    mejor\_umbral = encontrar\_mejor\_umbral\_entropia(imagen)

    # Aplicar diferentes tonos (3, 4, 8, 16)

    tonos = [3, 4, 8, 16]

    imagenes\_tonos = [aplicar\_tonos(imagen, num\_tonos) for num\_tonos in tonos]

    # Mostrar el histograma y las imágenes umbralizadas

    plt.figure(figsize=(15, 10))

    # Mostrar el histograma original

    plt.subplot(2, len(tonos), 1)

    plt.bar(range(256), histograma, color='black')

    plt.title('Histograma Original')

    plt.xlim([0, 255])

    # Mostrar las imágenes umbralizadas y guardar las matrices

    for idx, num\_tonos in enumerate(tonos):

        plt.subplot(2, len(tonos), idx + 2)

        plt.imshow(imagenes\_tonos[idx], cmap='gray')

        # Guardar la matriz de cada imagen en un archivo CSV

        nombre\_archivo = f"imagen\_{num\_tonos}\_tonos\_entropia.csv"

        np.savetxt(nombre\_archivo, imagenes\_tonos[idx], delimiter=',', fmt='%d')

        plt.title(f'{num\_tonos} tonos')

    for num\_tonos in tonos:

        imagen\_tonos = aplicar\_tonos(imagen, num\_tonos)

        # Guardar la imagen umbralizada con diferentes tonos

        cv2.imwrite(f'imagen\_{num\_tonos}\_tonos.png', imagen\_tonos)

    # Mostrar las imágenes y el histograma

    plt.tight\_layout()

    plt.show()

Diagrama

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Conclusiones**

En este trabajo se han implementado y comparado tres métodos automáticos de umbralización de imágenes: Umbral por Entropía, Grupo de Varianza y Aproximación de Valle Global. Cada uno de estos métodos aborda la segmentación de imágenes desde diferentes enfoques matemáticos y estadísticos, lo que nos ha permitido observar cómo cada técnica tiene ventajas y desventajas dependiendo de la distribución de los tonos en la imagen.

En conjunto, estos tres métodos de umbralización ofrecen una gama de opciones para la segmentación automática de imágenes, permitiendo seleccionar el enfoque más adecuado según las características particulares de cada imagen. Los resultados obtenidos, tanto en la visualización de las imágenes umbralizadas como en los histogramas y matrices exportadas, permiten una evaluación cuantitativa y cualitativa que puede ser extendida a otros contextos de procesamiento de imágenes.

**Bibliografía**

*OpenCV modules*. OpenCV. (n.d.). https://docs.opencv.org/4.x/index.html

NumPy Org. (2024, June 17). NumPy. https://numpy.org/

*Matplotlib 3.5.3 documentation#*. Matplotlib documentation - Matplotlib 3.5.3 documentation. (2012). https://matplotlib.org/3.5.3/index.html