

## 3\_\_B\_\_Otsu

November 14, 2024

### 0.0.1 OTSU Thresholding

```
[33]: import cv2
import matplotlib.pyplot as plt
import numpy as np
```

### 0.0.2 Cargar la Imagen y Convertirla a Escala de Grises

Se carga una imagen y se convierte a escala de grises para reducir la complejidad y facilitar el procesamiento. La conversión permite que los métodos de umbralización y los ruidos sean aplicados sin los matices de color, lo que simplifica la segmentación y el análisis de la estructura de la imagen.

```
[34]: # Cargar la imagen en escala de grises
image = cv2.imread('lanus.png')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
gray_image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY)
```

### 0.0.3 Aplicar Umbralización de Otsu

El proceso de la umbralización de Otsu sigue estos pasos:

- 1- Calcular el histograma: Se analiza el histograma de la imagen para conocer la frecuencia de cada nivel de gris.
- 2- Probar diferentes umbrales: El algoritmo evalúa todos los posibles valores de umbral desde el nivel mínimo hasta el máximo de intensidad.
- 3- Calcular la varianza entre clases: Para cada posible umbral, se calcula la varianza entre clases, que mide cómo de separados están los píxeles de fondo y objeto. Este cálculo se hace en función de la probabilidad de los píxeles que están en cada grupo y sus medias.
- 4- Seleccionar el umbral óptimo: El umbral final es aquel que maximiza la varianza entre clases, es decir, el que mejor separa los dos grupos de píxeles.

La ventaja principal de la umbralización de Otsu es que no requiere especificar el valor del umbral manualmente. En cambio, el algoritmo determina automáticamente el umbral óptimo en función del histograma, lo cual hace que sea robusto y eficiente en imágenes con contraste definido entre objeto y fondo. Esto resulta en una segmentación precisa, especialmente útil en condiciones de iluminación uniforme.

```
[35]: # Aplicar el método de umbralización de Otsu
umbral_valor, umbral_otsu = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY_
↪+ cv2.THRESH_OTSU)

# Imprimir el valor de umbral calculado por Otsu
print(f"Umbral calculado por Otsu: {umbral_valor}")

# Mostrar la imagen original y la imagen umbralizada
plt.figure(figsize=(15, 5))

# Imagen en color original
plt.subplot(1, 3, 1)
plt.imshow(image_rgb)
plt.title('Imagen Original')
plt.axis('off')

# Imagen en escala de grises
plt.subplot(1, 3, 2)
plt.imshow(gray_image, cmap='gray')
plt.title("Imagen en Escala de Grises")
plt.axis("off")

# Imagen umbralizada con Otsu
plt.subplot(1, 3, 3)
plt.imshow(umbral_otsu, cmap='gray')
plt.title(f"Umbralización de Otsu\nUmbral final: {umbral_valor}")
plt.axis("off")

plt.show()
```

Umbral calculado por Otsu: 112.0



#### 0.0.4 Agregar Ruido Gaussiano

`num_salt = int(np.ceil(prob * imagen.size * 0.5))`: Calcula el número de píxeles que serán alterados a “sal” (blancos). `imagen.size` es el número total de píxeles en la imagen, y el 0.5 indica que se aplicará la mitad de los cambios para “sal”.

`num_pepper = int(np.ceil(prob * imagen.size * 0.5))`: Similar a `num_salt`, calcula el número de píxeles que serán alterados a “pimienta” (negros), aplicando la otra mitad de los cambios.

`salt_coords = (np.random.randint(0, imagen.shape[0], num_salt), np.random.randint(0, imagen.shape[1], num_salt))`: Genera coordenadas aleatorias para los puntos de “sal” en la imagen. `np.random.randint` selecciona valores de fila y columna aleatorios dentro del tamaño de la imagen para colocar los puntos blancos.

`imagen_ruido[salt_coords] = 255`: Asigna el valor 255 (blanco) a los píxeles en las coordenadas seleccionadas para “sal”, agregando puntos blancos a la imagen.

`pepper_coords = (np.random.randint(0, imagen.shape[0], num_pepper), np.random.randint(0, imagen.shape[1], num_pepper))`: Genera coordenadas aleatorias para los puntos de “pimienta” en la imagen de manera similar a los puntos de “sal”.

`imagen_ruido[pepper_coords] = 0`: Asigna el valor 0 (negro) a los píxeles en las coordenadas seleccionadas para “pimienta”, agregando puntos negros a la imagen.

```
[36]: # Función para agregar ruido gaussiano
def add_gaussian_noise(image, mean, stddev, contamination_percentage):
    # Genera ruido gaussiano
    gaussian_noise = np.random.normal(mean, stddev, image.shape)

    # Calcula la máscara de contaminación
    mask = np.random.rand(*image.shape) < (contamination_percentage / 100)

    # Agrega el ruido a la imagen solo en los puntos especificados por la
    ↪máscara
    noisy_image = np.copy(image)
    noisy_image[mask] = image[mask] + gaussian_noise[mask]

    # Asegura que los valores se mantengan en el rango [0, 255]
    noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)

    return noisy_image

# Aplicar el ruido gaussiano a la imagen en escala de grises
imagen_ruido_gaussiano = add_gaussian_noise(gray_image, 0, 25, 70)

# Aplicar el método de umbralización de Otsu
umbral_valor_gaussiano, imagen_otsu_gaussiana = cv2.
    ↪threshold(imagen_ruido_gaussiano, 0, 255, cv2.THRESH_BINARY + cv2.
    ↪THRESH_OTSU)

# Imprimir el valor de umbral calculado por Otsu
print(f"Umbral calculado por Otsu: {umbral_valor_gaussiano}")

# Mostrar la imagen original, la imagen con ruido exponencial y la imagen
    ↪umbralizada
```

```
plt.figure(figsize=(18, 5))

# Imagen original
plt.subplot(1, 3, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Imagen Original en Escala de Grises')
plt.axis('off')

# Imagen con ruido Gaussiano
plt.subplot(1, 3, 2)
plt.imshow(imagen_ruido_gaussiano, cmap='gray')
plt.title('Imagen con Ruido Gaussiano')
plt.axis('off')

# Imagen umbralizada con Otsu
plt.subplot(1, 3, 3)
plt.imshow(imagen_otsu_gaussiana, cmap='gray')
plt.title(f"Umbralización de Otsu Gauss\nUmbral final: {umbral_valor_gaussiano}")
plt.axis("off")

plt.show()
```

Umbral calculado por Otsu: 119.0



### 0.0.5 Agregar Ruido Exponencial Multiplicativo

**expo = np.random.exponential(scale, imagen.shape):** Genera una matriz de ruido exponencial con la misma forma que la imagen, usando `np.random.exponential`. Los valores en la matriz siguen una distribución exponencial con una escala específica (`scale`).

**imagen\_ruido = imagen \* expo:** Multiplica la imagen original por el ruido exponencial, afectando los valores de los píxeles y generando una imagen con ruido aplicado.

**imagen\_ruido = imagen\_ruido / np.max(imagen\_ruido) \* 255:** Normaliza la imagen ruidosa dividiendo por su valor máximo y multiplicando por 255, para que los valores resultantes estén en el rango de intensidad de 0 a 255.

**imagen\_ruido = np.clip(imagen\_ruido, 0, 255).astype(np.uint8):** Asegura que todos los

valores de píxeles estén dentro del rango válido (0 a 255) usando `np.clip`, y convierte la imagen resultante al tipo `uint8` para que pueda ser guardada o visualizada correctamente.

```
[37]: # Función para agregar ruido exponencial multiplicativo
def add_exponential_noise(image, scale, contamination_percentage):
    # Genera ruido exponencial
    exponential_noise = np.random.exponential(scale, image.shape)

    # Calcula la máscara de contaminación
    mask = np.random.rand(*image.shape) < (contamination_percentage / 100)

    # Agrega el ruido a la imagen solo en los puntos especificados por la
    ↪máscara
    noisy_image = np.copy(image)
    noisy_image[mask] = image[mask] * exponential_noise[mask]

    # Asegura que los valores se mantengan en el rango [0, 255]
    noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)

    return noisy_image

# Aplicar el ruido exponencial a la imagen en escala de grises
imagen_ruido_exponencial = add_exponential_noise(gray_image, 1.0, 50)

# Aplicar el método de umbralización de Otsu
umbral_valor_exponencial, imagen_otsu_exponencial = cv2.
    ↪threshold(imagen_ruido_exponencial, 0, 255, cv2.THRESH_BINARY + cv2.
    ↪THRESH_OTSU)

# Imprimir el valor de umbral calculado por Otsu
print(f"Umbral calculado por Otsu: {umbral_valor_exponencial}")

# Mostrar la imagen original, la imagen con ruido exponencial y la imagen
    ↪umbralizada
plt.figure(figsize=(18, 5))

# Imagen original
plt.subplot(1, 3, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Imagen Original en Escala de Grises')
plt.axis('off')

# Imagen con ruido exponencial
plt.subplot(1, 3, 2)
plt.imshow(imagen_ruido_exponencial, cmap='gray')
plt.title('Imagen con Ruido Exponencial')
plt.axis('off')
```

```
# Imagen umbralizada con Otsu
plt.subplot(1, 3, 3)
plt.imshow(imagen_otsu_exponencial, cmap='gray')
plt.title(f"Umbralización de Otsu Exponencial\nUmbral final:␣
↪{umbral_valor_exponencial}")
plt.axis("off")

plt.show()
```

Umbral calculado por Otsu: 99.0



### 0.0.6 Agregar Ruido de Sal y Pimienta

Finalmente, se añade ruido de sal y pimienta a la imagen. Este tipo de ruido reemplaza aleatoriamente algunos píxeles con valores máximos (blanco) y mínimos (negro), imitando el efecto de partículas blancas y negras en la imagen.

Este tipo de ruido es útil para probar la robustez de métodos de procesamiento de imagen ante datos corruptos. El bloque también aplica la umbralización de Otsu para analizar cómo esta técnica responde a la imagen con ruido de sal y pimienta.

La visualización incluye la imagen original en escala de grises, la imagen con ruido y la imagen umbralizada.

```
[38]: # Función para agregar ruido sal y pimienta
def agregar_ruido_sal_pimienta(imagen, prob=0.1):
    imagen_ruido = np.copy(imagen)
    num_salt = int(np.ceil(prob * imagen.size * 0.5))
    num_pepper = int(np.ceil(prob * imagen.size * 0.5))

    # Coordenadas de sal
    salt_coords = (np.random.randint(0, imagen.shape[0], num_salt),
                   np.random.randint(0, imagen.shape[1], num_salt))
    imagen_ruido[salt_coords] = 255 # Blanco para 'sal'

    # Coordenadas de pimienta
    pepper_coords = (np.random.randint(0, imagen.shape[0], num_pepper),
```



```

        np.random.randint(0, imagen.shape[1], num_pepper))
    imagen_ruido[pepper_coords] = 0 # Negro para 'pimienta'

    return imagen_ruido

# Aplicar el ruido sal y pimienta a la imagen en escala de grises
imagen_ruido_sal_pimienta = agregar_ruido_sal_pimienta(gray_image)

# Aplicar el método de umbralización de Otsu
umbral_valor_sp, imagen_otsu_sal_pimienta = cv2.
    ↳threshold(imagen_ruido_sal_pimienta, 0, 255, cv2.THRESH_BINARY + cv2.
    ↳THRESH_OTSU)

# Imprimir el valor de umbral calculado por Otsu
print(f"Umbral calculado por Otsu: {umbral_valor_sp}")

# Mostrar la imagen original, la imagen con ruido de sal y pimienta y la imagen
    ↳umbralizada
plt.figure(figsize=(18, 5))

# Imagen original
plt.subplot(1, 3, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Imagen Original en Escala de Grises')
plt.axis('off')

# Imagen con ruido sal y pimienta
plt.subplot(1, 3, 2)
plt.imshow(imagen_ruido_sal_pimienta, cmap='gray')
plt.title('Imagen con Ruido Sal y Pimienta')
plt.axis('off')

# Imagen umbralizada con Otsu
plt.subplot(1, 3, 3)
plt.imshow(imagen_otsu_sal_pimienta, cmap='gray')
plt.title(f"Umbralización de Otsu Sal y Pimienta\nUmbral final:
    ↳{umbral_valor_sp}")
plt.axis("off")

plt.show()

```

Umbral calculado por Otsu: 123.0



### 0.0.7 Comparación de todos los resultados

```
[39]: # Mostrar la imagen original, las bandas binarias y la imagen clasificada final
plt.figure(figsize=(20, 10))

# Mostrar la imagen original en la primera posición (arriba)
plt.subplot(2, 3, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Imagen Original en Escala de Grises')
plt.axis('off')

# Mostrar la imagen clasificada final en la segunda posición (arriba)
plt.subplot(2, 3, 2)
plt.imshow(umbral_otsu, cmap='gray')
plt.title(f'Imagen umbralizada con Otsu\nUmbral final: {umbral_valor}')
plt.axis('off')

# Mostrar las bandas binarias en la fila inferior
plt.subplot(2, 3, 4)
plt.imshow(imagen_otsu_gaussiana, cmap='gray')
plt.title(f'Imagen Otsu con Ruido Gaussiano\nUmbral final: {umbral_valor_gaussiano}')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.imshow(imagen_otsu_exponencial, cmap='gray')
plt.title(f'Imagen Otsu con Ruido Exponencial\nUmbral final: {umbral_valor_exponencial}')
plt.axis('off')

plt.subplot(2, 3, 6)
plt.imshow(imagen_otsu_sal_pimienta, cmap='gray')
plt.title(f'Imagen Otsu con Ruido Sal y Pimienta\nUmbral final: {umbral_valor_sp}')
plt.axis('off')
```



```
plt.show()
```



#### 0.0.8 Conclusiones Umbralización Otsu

1- Umbralización de Otsu sin ruido: En la imagen sin ruido, el umbral calculado por Otsu fue de 112. Este valor es óptimo para maximizar la separación entre el fondo y los objetos de interés. El método de Otsu proporciona una segmentación adecuada en este caso, permitiendo distinguir claramente las áreas de vegetación, estructura urbana y otras características.

2- Umbralización de Otsu con ruido gaussiano: Con la adición de ruido gaussiano, el umbral subió a 119. Esto indica que el método de Otsu tiende a ajustar el umbral para compensar la dispersión en las intensidades causada por el ruido. El resultado sigue siendo efectivo, aunque la segmentación puede ser un poco menos precisa en áreas donde el ruido afecta más.

3- Umbralización de Otsu con ruido exponencial: En este caso, el umbral bajó a 99. Este descenso se debe a la naturaleza del ruido exponencial, que introduce más píxeles de baja intensidad, forzando al método de Otsu a reducir el umbral para poder capturar áreas significativas de la imagen.

4- Umbralización de Otsu con ruido sal y pimienta: El umbral subió a 123 en presencia de ruido sal y pimienta. Este tipo de ruido introduce píxeles extremadamente brillantes y oscuros, lo cual afecta la distribución de intensidad y obliga al método de Otsu a elevar el umbral para minimizar el impacto de los valores extremos. La segmentación es adecuada, pero el ruido sal y pimienta genera puntos dispersos que pueden dificultar la interpretación visual.

[ ]: