Họ tên : Nguyễn Quang Anh

Mã sinh viên : B21DCDT036

CHƯƠNG 8 : PHÂN ĐOẠN ẢNH ( Image Segmentation )

THUẬT TOÁN 1 : THUẬT TOÁN OTSU

### **1**,**Giới thiệu về thuật toán Otsu**

Thuật toán Otsu là một kỹ thuật phân đoạn ảnh được sử dụng để xác định ngưỡng tối ưu trong việc chuyển đổi ảnh xám sang ảnh nhị phân. Giả định cơ bản là histogram của ảnh có tính chất **hai đỉnh (bimodal)**, tương ứng với hai lớp chính:

* Một lớp chứa các pixel thuộc **foreground (đối tượng chính)**.
* Một lớp chứa các pixel thuộc **background (nền ảnh)**.

Thuật toán Otsu hoạt động dựa trên việc:

1. **Tính toán phân phối xác suất histogram:** Xác định phân phối các giá trị pixel.
2. **Tính toán các thông số thống kê:** Độ biến thiên trong mỗi lớp (intra-class variance) và giữa các lớp (inter-class variance) tại mỗi giá trị ngưỡng.
3. **Tìm giá trị ngưỡng tối ưu:** Lựa chọn giá trị ngưỡng làm cực đại hóa sự tách biệt giữa các lớp.

### **2 Các bước thực hiện thuật toán**

1. **Chuyển đổi ảnh gốc sang ảnh xám (grayscale):** Ảnh RGB cần được chuyển thành ảnh có một kênh cường độ xám.
2. **Tính toán histogram của ảnh xám:** Phân phối tần suất của các giá trị pixel được tạo ra.
3. **Tính toán giá trị ngưỡng Otsu:** Sử dụng hàm threshold\_otsu() để xác định ngưỡng tối ưu.
4. **Tạo ảnh nhị phân:** Các pixel lớn hơn ngưỡng được đặt thành giá trị 1 (trắng), các pixel nhỏ hơn hoặc bằng ngưỡng được đặt thành 0 (đen).
5. **Hiển thị kết quả:** Bao gồm ảnh gốc, histogram, và ảnh sau khi phân đoạn.

**3,Giải thích từng dòng lệnh trong đoạn mã**

import pylab  
from skimage.io import imread  
from skimage.color import rgb2gray  
from skimage.filters import threshold\_otsu  
  
# Đọc ảnh và chuyển sang ảnh xám  
image = rgb2gray(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\me16.jpg"))  
  
# Áp dụng phương pháp Otsu để tính ngưỡng  
thresh = threshold\_otsu(image)  
  
# Tạo ảnh nhị phân  
binary = image > thresh  
  
# Tạo các subplot để hiển thị  
fig, axes = pylab.subplots(nrows=2, ncols=2, figsize=(20, 15))  
axes = axes.ravel()  
  
# Đặt các subplot  
axes[0], axes[1] = pylab.subplot(2, 2, 1), pylab.subplot(2, 2, 2)  
axes[2] = pylab.subplot(2, 2, 3, sharex=axes[0], sharey=axes[0])  
axes[3] = pylab.subplot(2, 2, 4, sharex=axes[0], sharey=axes[0])  
  
# Hiển thị ảnh gốc  
axes[0].imshow(image, cmap=pylab.cm.gray)  
axes[0].set\_title('Original', size=20)  
axes[0].axis('off')  
  
# Hiển thị histogram  
axes[1].hist(image.ravel(), bins=256, density=True)  
axes[1].set\_title('Histogram', size=20)  
axes[1].axvline(thresh, color='r')  
  
# Hiển thị ảnh nhị phân sau khi ngưỡng Otsu  
axes[2].imshow(binary, cmap=pylab.cm.gray)  
axes[2].set\_title('Thresholded (Otsu)', size=20)  
axes[2].axis('off')  
  
# Ẩn subplot thứ 4  
axes[3].axis('off')  
  
# Hiển thị tất cả các subplot  
pylab.tight\_layout()  
pylab.show()

from skimage.io import imread

from skimage.color import rgb2gray

from skimage.filters import threshold\_otsu

import pylab

* **imread:** Đọc ảnh từ đường dẫn file.
* **rgb2gray:** Chuyển đổi ảnh RGB (màu) sang ảnh xám (một kênh).
* **threshold\_otsu:** Hàm tính toán ngưỡng Otsu tối ưu.
* **pylab:** Thư viện hỗ trợ hiển thị đồ thị và hình ảnh.

image = rgb2gray(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\horse.jpg"))

* Đọc ảnh từ đường dẫn file và chuyển đổi ảnh sang ảnh xám.

thresh = threshold\_otsu(image)

* Tính toán ngưỡng Otsu dựa trên histogram của ảnh xám.

binary = image > thresh

* Tạo ảnh nhị phân bằng cách so sánh các giá trị pixel với ngưỡng đã tính.

fig, axes = pylab.subplots(nrows=2, ncols=2, figsize=(20, 15))

axes = axes.ravel()

* Tạo một lưới các subplot để hiển thị nhiều hình ảnh (ảnh gốc, histogram, ảnh nhị phân).

axes[0].imshow(image, cmap=pylab.cm.gray)

axes[0].set\_title('Original', size=20)

axes[0].axis('off')

* Hiển thị ảnh gốc ở ô đầu tiên.

axes[1].hist(image.ravel(), bins=256, density=True)

axes[1].set\_title('Histogram', size=20)

axes[1].axvline(thresh, color='r')

* Hiển thị histogram của ảnh gốc kèm đường ngưỡng Otsu (màu đỏ).

axes[2].imshow(binary, cmap=pylab.cm.gray)

axes[2].set\_title('Thresholded (Otsu)', size=20)

axes[2].axis('off')

* Hiển thị ảnh nhị phân (sau khi áp dụng ngưỡng Otsu).

axes[3].axis('off')

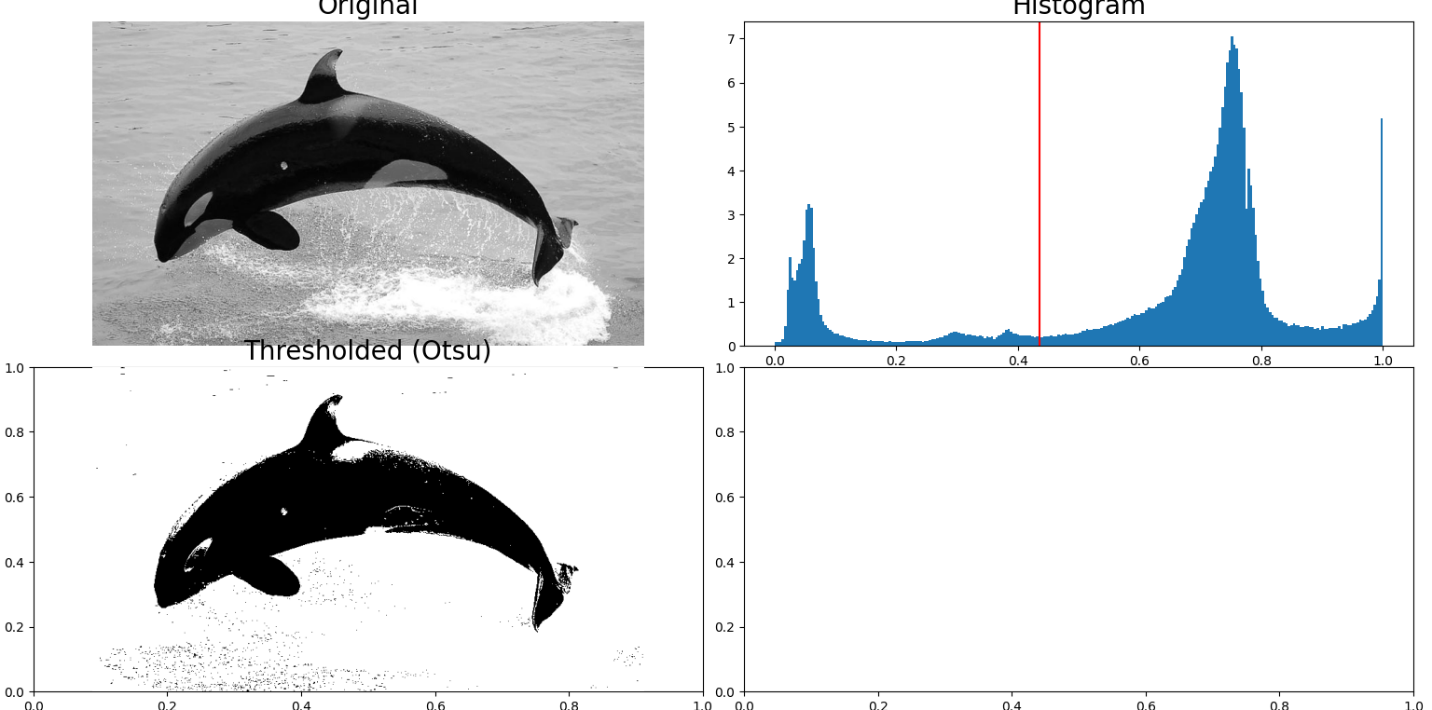
* Không hiển thị ô thứ tư.

pylab.tight\_layout()

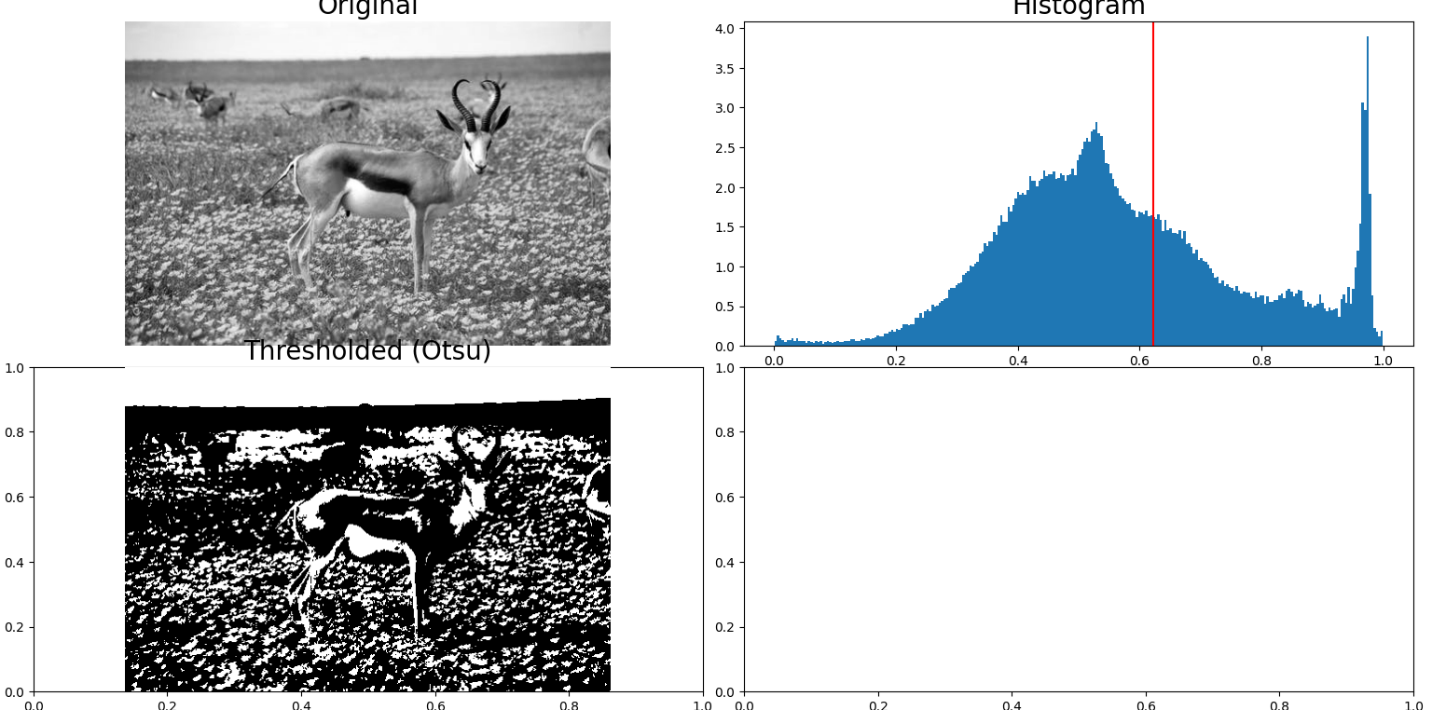
pylab.show()

**4, Kết quả thực hiện thuật toán**

Ví dụ 1 : Ảnh gốc : 

Ảnh sau khi xử lý :

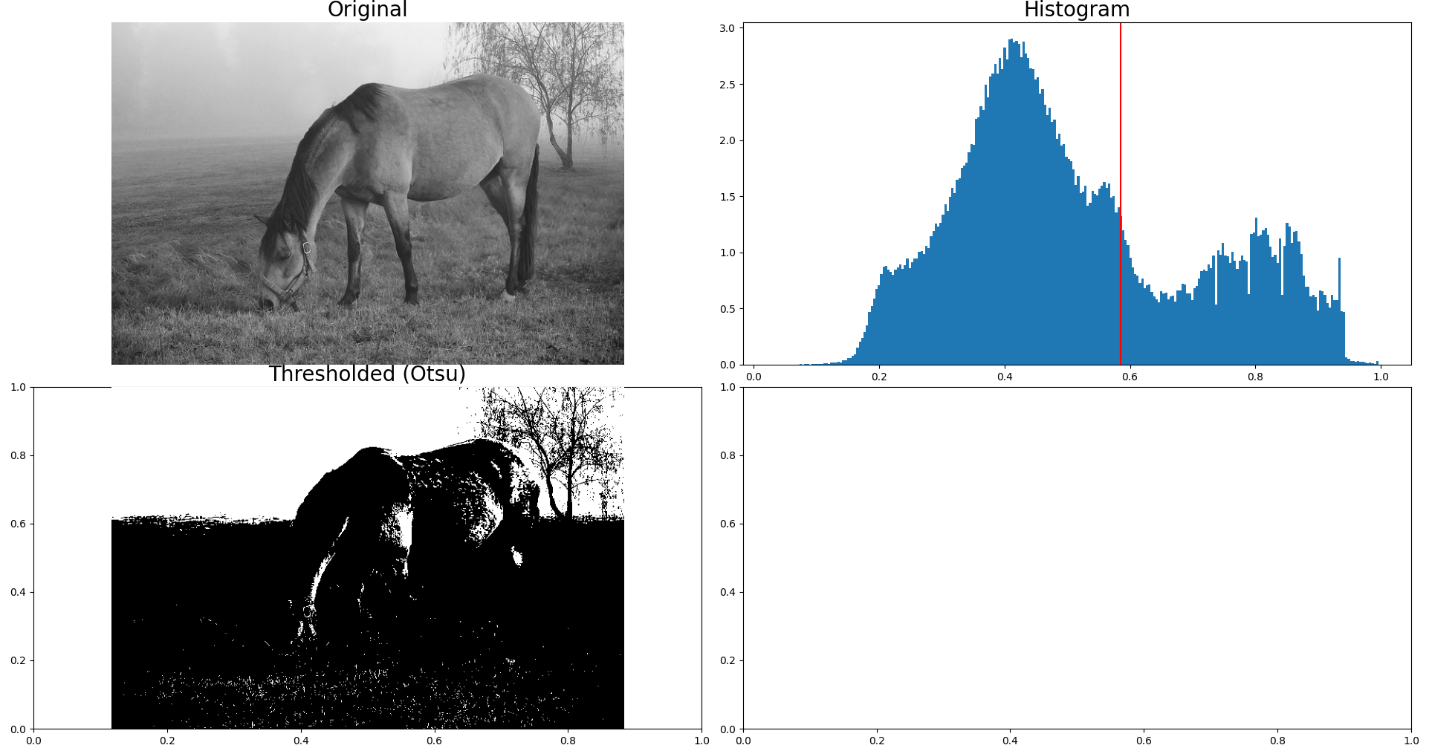
Ví dụ 2 : Ảnh gốc : 

Ảnh sau khi xử lý : 

Ví dụ 3 : Ảnh gốc :



Ảnh sau khi xử lý :



**Nhận xét ảnh đầu ra**

1. **Ảnh gốc:**
   * Đây là ảnh xám được chuyển đổi từ ảnh màu ban đầu.
   * Các pixel có cường độ thay đổi từ đen (0) đến trắng (1).
2. **Histogram:**
   * Histogram cho thấy sự phân bố cường độ pixel.
   * Có thể quan sát thấy tính chất **hai đỉnh** của histogram, lý tưởng cho phương pháp Otsu.
3. **Ảnh nhị phân (Thresholded Otsu):**
   * Đối tượng (con ngựa) được phân biệt rõ ràng với nền.
   * Phương pháp Otsu hiệu quả khi đối tượng và nền có độ tương phản cao.

**Tóm lại**

* **Ưu điểm:** Phương pháp Otsu đơn giản, không yêu cầu thông số thủ công, và hoạt động hiệu quả trên ảnh có tính chất bimodal.
* **Nhược điểm:** Kém hiệu quả với histogram phức tạp hoặc ảnh có nhiễu cao.
* **Ứng dụng:** Phân đoạn ảnh trong các bài toán nhận dạng đối tượng, xử lý văn bản (như nhận dạng chữ viết), và nhiều lĩnh vực khác trong xử lý ảnh.

THUẬT TOÁN 2 : Thuật toán phân đoạn ảnh hiệu quả dựa trên đồ thị của Felzenszwalb

**1,Giới thiệu thuật toán của Felzenszwalb**

**Ý tưởng chính:**

* Thuật toán xây dựng một đồ thị từ ảnh, trong đó:
  + Mỗi pixel là một **đỉnh** (vertex).
  + Cạnh nối giữa hai đỉnh có trọng số là độ chênh lệch (ví dụ: về cường độ sáng, màu sắc) giữa các pixel.
* Sau đó, thuật toán thực hiện phân đoạn bằng cách **tìm các thành phần liên thông**:
  + Trọng số thấp giữa hai đỉnh → pixel tương tự → cùng thành phần.
  + Trọng số cao → khác biệt → phân chia thành phần.

**Các đặc điểm quan trọng:**

1. **Tỷ lệ (Scale):** Một tham số duy nhất điều chỉnh kích thước các đoạn ảnh.
   * Giá trị nhỏ → nhiều đoạn ảnh nhỏ.
   * Giá trị lớn → ít đoạn ảnh lớn hơn.
2. **Độ tương phản cục bộ (Local contrast):** Ảnh hưởng đến số lượng và kích thước các đoạn.

**Ưu điểm:**

* Nhanh và hiệu quả ngay cả trên ảnh lớn.
* Bảo toàn các chi tiết ở vùng có biến đổi thấp.

**Nhược điểm:**

* Phụ thuộc vào tham số tỷ lệ; cần thử nghiệm để đạt kết quả tốt.

**2,Các bước thực hiện thuật toán**

1. **Xây dựng đồ thị từ ảnh:** Tạo đồ thị với các pixel làm đỉnh và trọng số cạnh là độ chênh lệch giữa các pixel.
2. **Xác định các thành phần liên thông:** Phân chia đồ thị thành các thành phần liên thông dựa trên trọng số cạnh.
3. **Kiểm soát kích thước thành phần:** Sử dụng tham số tỷ lệ để hợp nhất hoặc chia nhỏ các thành phần liên thông.
4. **Tạo ảnh phân đoạn:** Áp dụng kết quả phân đoạn lên ảnh đầu vào.

**3, Giải thích từng dòng lệnh mã nguồn**

import pylab  
from skimage.io import imread  
from skimage.segmentation import felzenszwalb  
from skimage.color import rgb2gray  
from skimage.segmentation import mark\_boundaries  
  
# Hàm vẽ ảnh  
def plot\_image(img, title):  
 pylab.imshow(img)  
 pylab.title(title, size=20)  
 pylab.axis('off')  
  
# Đọc ảnh và cắt giảm độ phân giải (downsample)  
img = imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\fish.jpg")[::2, ::2, :3]  
  
# Tạo một figure để vẽ các ảnh  
pylab.figure(figsize=(15, 10))  
  
# Vẽ ảnh với các giá trị scale khác nhau  
i = 1  
for scale in [50, 100, 200, 400]:  
 pylab.subplot(2, 2, i)  
 segments\_fz = felzenszwalb(img, scale=scale, sigma=0.5, min\_size=200)  
 plot\_image(mark\_boundaries(img, segments\_fz, color=(1, 0, 0)), 'scale=' + str(scale))  
 i += 1  
  
# Tiêu đề của toàn bộ figure  
pylab.suptitle('Felzenszwalb\'s method', size=30)  
  
# Đảm bảo layout đẹp và không bị chồng lên  
pylab.tight\_layout(rect=[0, 0.03, 1, 0.95])  
  
# Hiển thị ảnh  
pylab.show()

from skimage.io import imread

from skimage.segmentation import felzenszwalb

from skimage.color import rgb2gray

from skimage.segmentation import mark\_boundaries

import pylab

* **imread:** Đọc ảnh từ đường dẫn.
* **felzenszwalb:** Hàm thực hiện phân đoạn theo thuật toán Felzenszwalb.
* **mark\_boundaries:** Đánh dấu ranh giới giữa các vùng phân đoạn trên ảnh.
* **pylab:** Thư viện hỗ trợ hiển thị đồ thị và hình ảnh.

def plot\_image(img, title):

pylab.imshow(img)

pylab.title(title, size=20)

pylab.axis('off')

* **Hàm plot\_image:** Hàm tiện ích để hiển thị ảnh với tiêu đề.

img = imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\fish.jpg")[::2, ::2, :3]

* Đọc ảnh từ file và giảm kích thước bằng cách lấy mẫu (downsample) theo cả hai chiều để tăng tốc độ xử lý.

pylab.figure(figsize=(15, 10))

* Tạo một figure để chứa các subplot hiển thị các kết quả khác nhau.

for scale in [50, 100, 200, 400]:

pylab.subplot(2, 2, i)

segments\_fz = felzenszwalb(img, scale=scale, sigma=0.5, min\_size=200)

plot\_image(mark\_boundaries(img, segments\_fz, color=(1, 0, 0)), 'scale=' + str(scale))

i += 1

* **Vòng lặp qua các giá trị scale:**
  + **scale:** Tham số tỷ lệ điều chỉnh kích thước đoạn ảnh. Giá trị nhỏ hơn tạo nhiều đoạn hơn.
  + **sigma:** Giá trị làm mờ Gaussian trước khi phân đoạn, để giảm nhiễu.
  + **min\_size:** Kích thước tối thiểu của một đoạn (đo bằng số pixel).
  + **mark\_boundaries:** Vẽ đường viền đỏ giữa các đoạn trên ảnh đầu vào.
  + **plot\_image:** Hiển thị ảnh với tiêu đề ghi giá trị scale.

pylab.suptitle('Felzenszwalb\'s method', size=30)

pylab.tight\_layout(rect=[0, 0.03, 1, 0.95])

pylab.show()

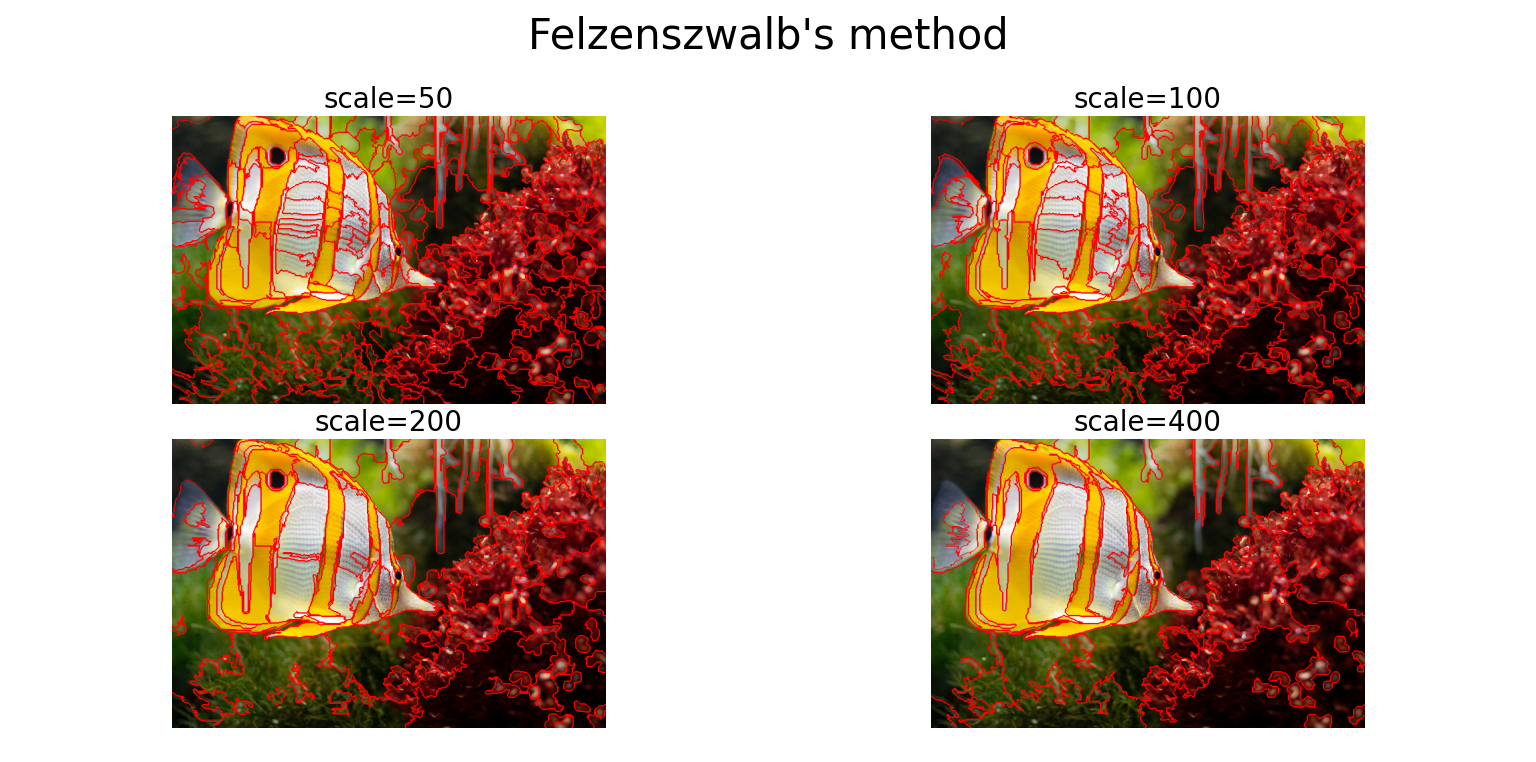
* Hiển thị tiêu đề lớn cho toàn bộ figure.
* **tight\_layout:** Đảm bảo các subplot không bị chồng chéo.
* **show:** Hiển thị toàn bộ các kết quả.

**4.Kết quả thực hiện thuật toán**

Ví dụ 1 :

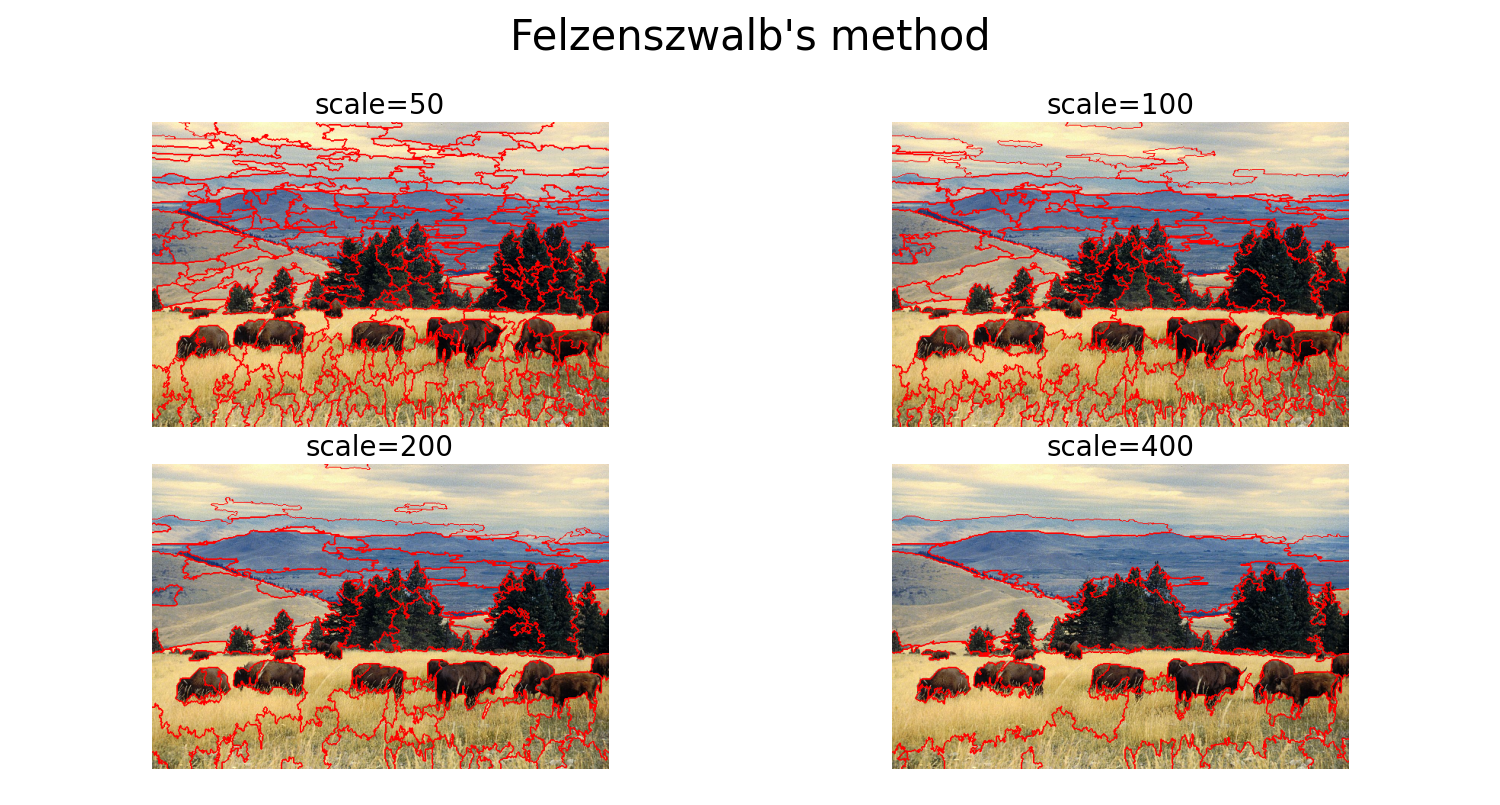
Ảnh gốc : 

Ảnh sau khi thực hiện :



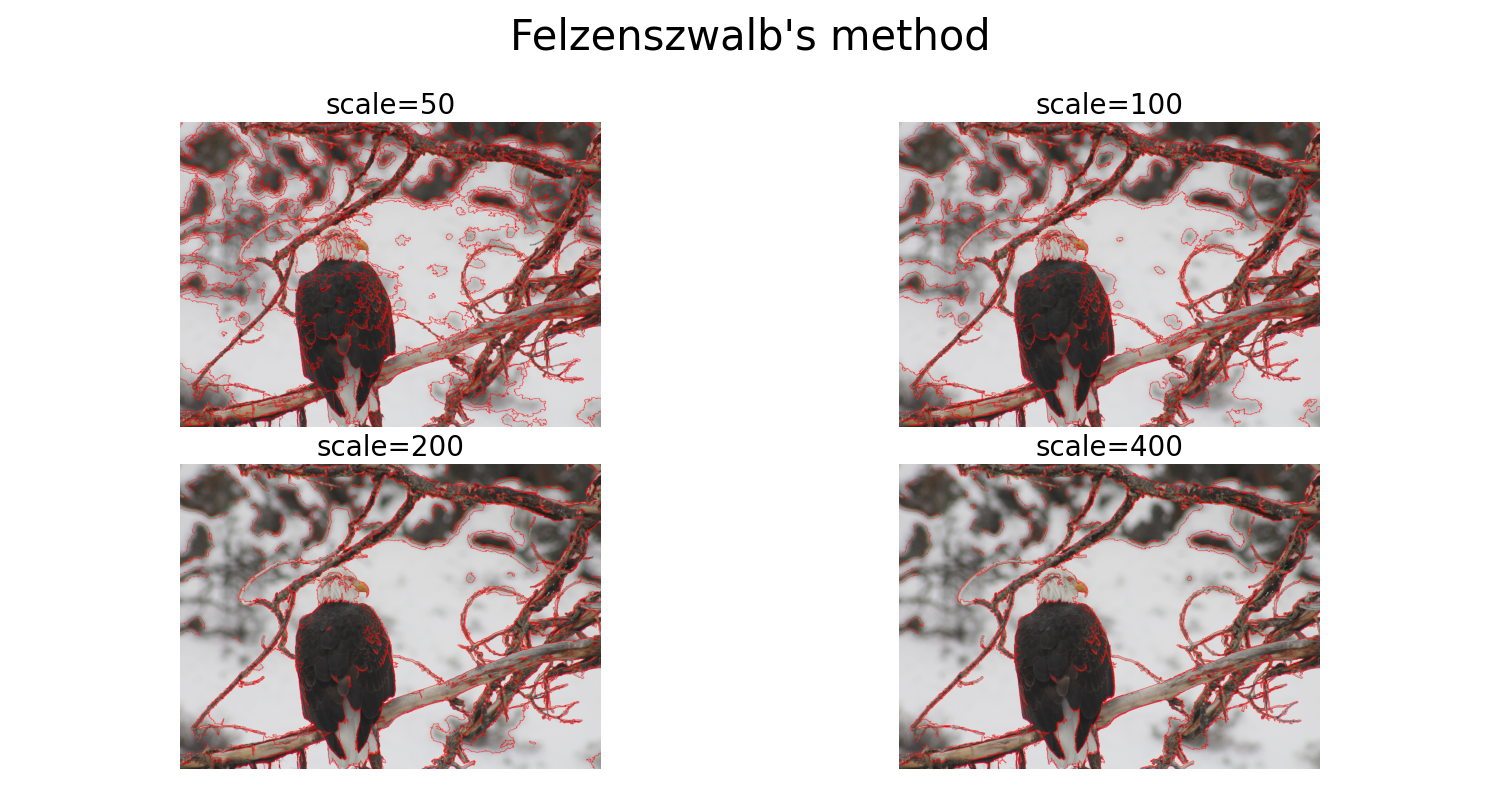
Ví dụ 2 : Ảnh gốc : 

Ảnh sau khi thực hiện :



Ví dụ 3 : Ảnh gốc : 

Ảnh sau khi thực hiện :



THUẬT TOÁN 3 : RAG merging

1, **Giới thiệu thuật toán**

**Region Adjacency Graph (RAG)** là kỹ thuật kết hợp các vùng được phân đoạn quá mức trong ảnh để cải thiện phân đoạn. Kỹ thuật này hoạt động dựa trên:

**SLIC segmentation:** Ban đầu, sử dụng thuật toán SLIC để phân đoạn ảnh thành nhiều vùng nhỏ.

**Region Adjacency Graph:** Tạo đồ thị mà mỗi nút là một vùng trong ảnh, và các cạnh biểu thị sự kết nối giữa các vùng lân cận. Các trọng số trên cạnh thể hiện mức độ khác biệt (ví dụ: màu sắc) giữa các vùng.

**Merging:** Tiến hành hợp nhất các vùng lân cận có sự tương đồng cao (dựa trên màu sắc trung bình). Tiến trình này lặp lại cho đến khi không còn cặp vùng nào đủ giống để hợp nhất.

**2,Các bước thực hiện thuật toán**

**B1 : Phân đoạn bằng SLIC:**

* + **segmentation.slic()**: Phân đoạn ảnh đầu vào thành **superpixel** (vùng nhỏ đồng nhất).
    - **compactness**: Cân bằng giữa độ mượt hình dạng vùng và độ tương đồng màu.
    - **n\_segments**: Số vùng ban đầu (over-segmentation).

**B2 : Tạo đồ thị RAG:**

* + **graph.rag\_mean\_color()**: Tạo **Region Adjacency Graph (RAG)**. Mỗi nút là một vùng trong ảnh, các cạnh nối các vùng lân cận với trọng số là độ khác biệt về màu trung bình giữa hai vùng.

**B3 : Hợp nhất các vùng:**

* + **graph.merge\_hierarchical()**: Tiến hành hợp nhất các vùng trong RAG:
    - **merge\_func**: Quy tắc hợp nhất (màu trung bình của hai vùng được hợp).
    - **weight\_func**: Trọng số giữa các vùng (tính sự khác biệt về màu sắc).
    - **thresh**: Ngưỡng để quyết định hợp nhất (nếu hai vùng có sự khác biệt thấp hơn ngưỡng thì hợp nhất).

**B4 : Tạo kết quả:**

* + **label2rgb()**: Chuyển các nhãn của vùng thành ảnh màu, mỗi vùng có màu trung bình đại diện.
  + **mark\_boundaries()**: Vẽ biên giới của các vùng phân đoạn trên ảnh.

3,Giải thích đoạn mã nguồn :

from skimage import segmentation  
from skimage import graph # Thay đổi từ 'skimage.future import graph' thành 'from skimage import graph'  
from skimage.io import imread  
import numpy as np  
import pylab  
from skimage.color import label2rgb  
  
# Hàm tính trọng số giữa các vùng (mean color)  
def \_weight\_mean\_color(graph, src, dst, n):  
 diff = graph.\_node[dst]['mean color'] - graph.\_node[src]['mean color']  
 diff = np.linalg.norm(diff)  
 return {'weight': diff}  
  
# Hàm hợp nhất màu trung bình  
def merge\_mean\_color(graph, src, dst):  
 graph.\_node[dst]['total color'] += graph.\_node[src]['total color']  
 graph.\_node[dst]['pixel count'] += graph.\_node[src]['pixel count']  
 graph.\_node[dst]['mean color'] = (graph.\_node[dst]['total color'] /  
 graph.\_node[dst]['pixel count'])  
  
# Đọc ảnh  
img = imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\me12.jpg")  
  
# Phân đoạn ảnh bằng SLIC  
labels = segmentation.slic(img, compactness=30, n\_segments=400)  
  
# Xây dựng đồ thị RAG với màu sắc trung bình  
g = graph.rag\_mean\_color(img, labels)  
  
# Hợp nhất các vùng dựa trên đồ thị RAG  
labels2 = graph.merge\_hierarchical(labels, g, thresh=35, rag\_copy=False,  
 in\_place\_merge=True,  
 merge\_func=merge\_mean\_color,  
 weight\_func=\_weight\_mean\_color)  
  
# Chuyển đổi nhãn thành màu sắc trung bình  
out = label2rgb(labels2, img, kind='avg')  
  
# Vẽ các biên giới của các vùng phân đoạn  
out = segmentation.mark\_boundaries(out, labels2, (0, 0, 0))  
  
# Hiển thị kết quả  
pylab.figure(figsize=(20, 10))  
pylab.subplot(121), pylab.imshow(img), pylab.axis('off')  
pylab.subplot(122), pylab.imshow(out), pylab.axis('off')  
pylab.tight\_layout(), pylab.show()

**Import các thư viện**

from skimage import segmentation

from skimage import graph

from skimage.io import imread

import numpy as np

import pylab

from skimage.color import label2rgb

* **segmentation**: Thư viện để thực hiện phân đoạn ảnh (SLIC, đánh dấu biên giới...).
* **graph**: Thư viện cung cấp các công cụ liên quan đến đồ thị (RAG, hợp nhất vùng).
* **imread**: Đọc ảnh từ file.
* **np**: Dùng để xử lý các phép tính ma trận và số học (tính toán màu sắc).
* **pylab**: Dùng để hiển thị ảnh.
* **label2rgb**: Chuyển nhãn của các vùng thành ảnh màu.

**Định nghĩa các hàm quan trọng**

**Hàm \_weight\_mean\_color**

def \_weight\_mean\_color(graph, src, dst, n):

diff = graph.\_node[dst]['mean color'] - graph.\_node[src]['mean color']

diff = np.linalg.norm(diff)

return {'weight': diff}

* **Mục đích**: Tính trọng số giữa hai vùng dựa trên sự khác biệt về màu sắc trung bình.
* **Giải thích:**
  1. **graph.\_node[dst]['mean color'] và graph.\_node[src]['mean color']**: Lấy màu trung bình của hai vùng đích (dst) và nguồn (src) trong đồ thị.
  2. **np.linalg.norm(diff)**: Tính **khoảng cách Euclid** giữa hai màu.
  3. Trả về trọng số (**weight**), biểu thị sự khác biệt màu sắc giữa hai vùng.

**Hàm merge\_mean\_color**

def merge\_mean\_color(graph, src, dst):

graph.\_node[dst]['total color'] += graph.\_node[src]['total color']

graph.\_node[dst]['pixel count'] += graph.\_node[src]['pixel count']

graph.\_node[dst]['mean color'] = (graph.\_node[dst]['total color'] /

graph.\_node[dst]['pixel count'])

* **Mục đích**: Hợp nhất hai vùng dựa trên màu sắc trung bình.
* **Giải thích:**
  1. **graph.\_node[dst]['total color']**: Cộng tổng màu sắc của vùng nguồn (src) vào vùng đích (dst).
  2. **graph.\_node[dst]['pixel count']**: Cộng tổng số pixel của vùng nguồn vào vùng đích.
  3. **graph.\_node[dst]['mean color']**: Cập nhật màu sắc trung bình mới của vùng đích (dst) sau khi hợp nhất.

**Đọc ảnh đầu vào**

img = imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\me12.jpg")

* **imread()**: Đọc ảnh đầu vào từ đường dẫn.
* Kết quả là một ảnh màu dạng ma trận (height × width × 3).

**. Phân đoạn ảnh bằng SLIC**

labels = segmentation.slic(img, compactness=30, n\_segments=400)

* **Mục đích**: Phân đoạn ảnh thành các **superpixel** (vùng nhỏ).
* **Tham số:**
  + **compactness=30**: Điều chỉnh độ mịn của biên các vùng (giá trị càng lớn → vùng càng trơn).
  + **n\_segments=400**: Số lượng vùng nhỏ được tạo ra (phân đoạn quá mức).
* **Kết quả**: labels là một mảng nhãn, trong đó mỗi vùng nhỏ được gán một nhãn duy nhất.

**Tạo đồ thị RAG**

g = graph.rag\_mean\_color(img, labels)

* **rag\_mean\_color()**: Tạo **Region Adjacency Graph (RAG)** từ ảnh và nhãn.
  + **Nút (node):** Đại diện cho các vùng (superpixel).
  + **Cạnh (edge):** Liên kết giữa các vùng lân cận.
  + **Trọng số (weight):** Sự khác biệt màu sắc giữa hai vùng.

**Hợp nhất các vùng dựa trên RAG**

labels2 = graph.merge\_hierarchical(labels, g, thresh=35, rag\_copy=False,

in\_place\_merge=True,

merge\_func=merge\_mean\_color,

weight\_func=\_weight\_mean\_color)

* **merge\_hierarchical()**: Hợp nhất các vùng trong đồ thị RAG dựa trên trọng số.
  + **thresh=35**: Ngưỡng cho phép hợp nhất (các vùng có trọng số < 35 sẽ được hợp).
  + **rag\_copy=False**: Không tạo bản sao đồ thị, giảm bộ nhớ.
  + **in\_place\_merge=True**: Hợp nhất trực tiếp trên đồ thị hiện tại.
  + **merge\_func=merge\_mean\_color**: Hàm dùng để hợp nhất hai vùng.
  + **weight\_func=\_weight\_mean\_color**: Hàm dùng để tính trọng số giữa các vùng.
* **Kết quả**: labels2 là nhãn của các vùng sau khi hợp nhất.

**. Chuyển nhãn thành ảnh màu**

out = label2rgb(labels2, img, kind='avg')

* **label2rgb()**: Chuyển mỗi vùng thành một màu dựa trên màu trung bình của vùng.
* **kind='avg'**: Lấy trung bình màu của vùng để làm đại diện.

**. Đánh dấu biên giới**

out = segmentation.mark\_boundaries(out, labels2, (0, 0, 0))

* **mark\_boundaries()**: Vẽ đường biên đen giữa các vùng phân đoạn.

**. Hiển thị ảnh**

pylab.figure(figsize=(20, 10))

pylab.subplot(121), pylab.imshow(img), pylab.axis('off')

pylab.subplot(122), pylab.imshow(out), pylab.axis('off')

pylab.tight\_layout(), pylab.show()

* Tạo một figure với hai ảnh:
  1. Ảnh gốc (bên trái).
  2. Ảnh phân đoạn (bên phải).
* **tight\_layout()**: Sắp xếp bố cục đẹp, không bị chồng lấn.

**4,Kết quả sau khi thực hiện :**

Ví dụ 1 : 

Ví dụ 2 : 

Ví dụ 3 : 

**Nhận xét ảnh đầu ra**

1. **Ảnh gốc (bên trái):**
   * Là ảnh đầu vào chưa qua xử lý.
   * Dùng để so sánh với kết quả phân đoạn.
2. **Ảnh phân đoạn (bên phải):**
   * Các vùng nhỏ (superpixel) ban đầu từ SLIC được hợp nhất thành các vùng lớn hơn dựa trên độ tương đồng màu sắc.
   * Ranh giới vùng được vẽ đen (nhờ **mark\_boundaries()**).
   * Kết quả cho thấy:
     + Các vùng tương đồng về màu sắc được gom lại thành một vùng lớn.
     + Các chi tiết quan trọng như đường biên giữa các đối tượng trong ảnh vẫn được giữ rõ ràng.
3. **Nhận xét chi tiết:**
   * **Ưu điểm:**
     + Phân đoạn tốt hơn so với SLIC ban đầu, giảm số lượng vùng nhỏ không cần thiết.
     + Phù hợp với ảnh có các vùng đồng nhất màu sắc (ví dụ: cảnh quan, đồ vật lớn).
   * **Hạn chế:**
     + Với ảnh có nhiều chi tiết nhỏ hoặc biên giới mờ, có thể dẫn đến hợp nhất sai các vùng khác biệt.
4. **Tùy chỉnh:**
   * Tham số **thresh** quyết định mức độ hợp nhất:
     + Giá trị nhỏ hơn → Ít vùng được hợp, kết quả chi tiết hơn.
     + Giá trị lớn hơn → Nhiều vùng được hợp, ảnh có thể bị **quá tổng quát hóa**.