Họ tên : Nguyễn Quang Anh

Mã sinh viên : B21DCDT036

CHƯƠNG 5 : CẢI THIỆN ẢNH SỬ DỤNG ĐẠO HÀM ( Image Enhancement Using Derivatives )

THUẬT TOÁN 1 : LÀM SẮC NÉT ẢNH VỚI BỘ LỌC LAPLACE

1,Giới thiệu về thuật toán :

Làm sắc nét ảnh bằng bộ lọc Laplacian là một phương pháp nâng cao các cạnh trong ảnh. Bộ lọc Laplacian giúp làm nổi bật các vùng thay đổi nhanh trong ảnh (như các cạnh của đối tượng), trong khi làm mờ các vùng có độ sáng đồng đều.

2,Các bước thực hiện thuật toán :

B1 : **Chuyển đổi ảnh sang ảnh xám (grayscale)**:

* Ảnh màu được chuyển đổi thành ảnh đen trắng (grayscale) bằng hàm rgb2gray(), giúp giảm độ phức tạp khi áp dụng bộ lọc và tập trung vào cường độ sáng của ảnh.

B2 : **Áp dụng bộ lọc Laplacian**:

* Bộ lọc Laplacian tính toán sự thay đổi của độ sáng ảnh theo hướng bậc hai (đạo hàm bậc hai). Kết quả là ảnh sẽ trở nên sắc nét hơn, đặc biệt là các vùng có sự thay đổi mạnh mẽ về độ sáng (cạnh và chi tiết).

B3: **Cộng ảnh gốc với ảnh sau khi áp dụng Laplacian**:

* Sau khi áp dụng bộ lọc Laplacian, ảnh gốc và ảnh sau khi áp dụng Laplacian được cộng lại với nhau. Điều này giúp giữ lại các chi tiết của ảnh gốc và làm nổi bật các cạnh được tìm thấy bởi bộ lọc Laplacian.

B4: **Áp dụng np.clip() để giới hạn giá trị pixel**:

* Hàm np.clip() được sử dụng để đảm bảo rằng các giá trị pixel trong ảnh sắc nét không vượt quá phạm vi từ 0 đến 1, giúp duy trì độ tương phản hợp lý và tránh xuất hiện các giá trị không hợp lệ.

 B5: **Hiển thị ảnh**:

* Cuối cùng, ảnh gốc và ảnh đã được làm sắc nét được hiển thị cùng nhau để so sánh. Thư viện pylab và matplotlib giúp hiển thị ảnh trong cùng một cửa sổ.

3,Giải thích đoạn mã nguồn :

from skimage.color import rgb2gray # Import rgb2gray from skimage.color  
from skimage.io import imread  
from skimage.filters import laplace  
import numpy as np  
import matplotlib.pyplot as plt  
import pylab  
  
# Định nghĩa hàm plot\_image để hiển thị ảnh  
def plot\_image(img, title):  
 plt.imshow(img, cmap='gray') # Hiển thị ảnh với màu xám  
 plt.title(title) # Đặt tiêu đề cho ảnh  
 plt.axis('off') # Ẩn trục (axis)  
  
# Đọc ảnh và chuyển đổi sang ảnh đen trắng (gray)  
im = rgb2gray(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\me8.JPG"))  
  
# Áp dụng bộ lọc Laplace và làm sắc nét ảnh  
im1 = np.clip(laplace(im) + im, 0, 1)  
  
# Hiển thị ảnh gốc và ảnh sau khi làm sắc nét  
pylab.figure(figsize=(20, 30))  
pylab.subplot(211), plot\_image(im, 'original image')  
pylab.subplot(212), plot\_image(im1, 'sharpened image')  
pylab.tight\_layout()  
pylab.show()

**Nhập các thư viện cần thiết**

* **skimage.color.rgb2gray**: Thư viện này dùng để chuyển đổi ảnh màu (RGB) thành ảnh đen trắng (grayscale). Điều này giúp giảm độ phức tạp và chỉ tập trung vào cường độ sáng của ảnh khi làm sắc nét.
* **skimage.io.imread**: Đọc ảnh từ một tệp tin và trả về dữ liệu ảnh dưới dạng một mảng (array).
* **skimage.filters.laplace**: Áp dụng bộ lọc Laplacian lên ảnh. Bộ lọc này giúp phát hiện các cạnh trong ảnh, làm cho các chi tiết và cạnh trở nên rõ ràng hơn.
* **numpy**: Dùng để xử lý mảng và thực hiện các phép toán số học trên ảnh.
* **matplotlib.pyplot và pylab**: Dùng để hiển thị ảnh.

**Định nghĩa hàm plot\_image**

* Hàm **plot\_image(img, title)** dùng để hiển thị ảnh.
  + **plt.imshow(img, cmap='gray')**: Hiển thị ảnh với bảng màu xám (grayscale), giúp dễ dàng quan sát chi tiết của ảnh đen trắng.
  + **plt.title(title)**: Đặt tiêu đề cho ảnh hiển thị.
  + **plt.axis('off')**: Ẩn các trục của đồ thị để ảnh được hiển thị rõ ràng mà không bị làm phân tán bởi các trục.

**Đọc và chuyển ảnh sang ảnh đen trắng**

* **imread()**: Đọc ảnh từ đường dẫn chỉ định.
* **rgb2gray()**: Chuyển ảnh màu (RGB) thành ảnh xám (grayscale). Quá trình này giúp đơn giản hóa ảnh, giảm độ phức tạp và tập trung vào cường độ sáng.

**Áp dụng bộ lọc Laplace và làm sắc nét ảnh**

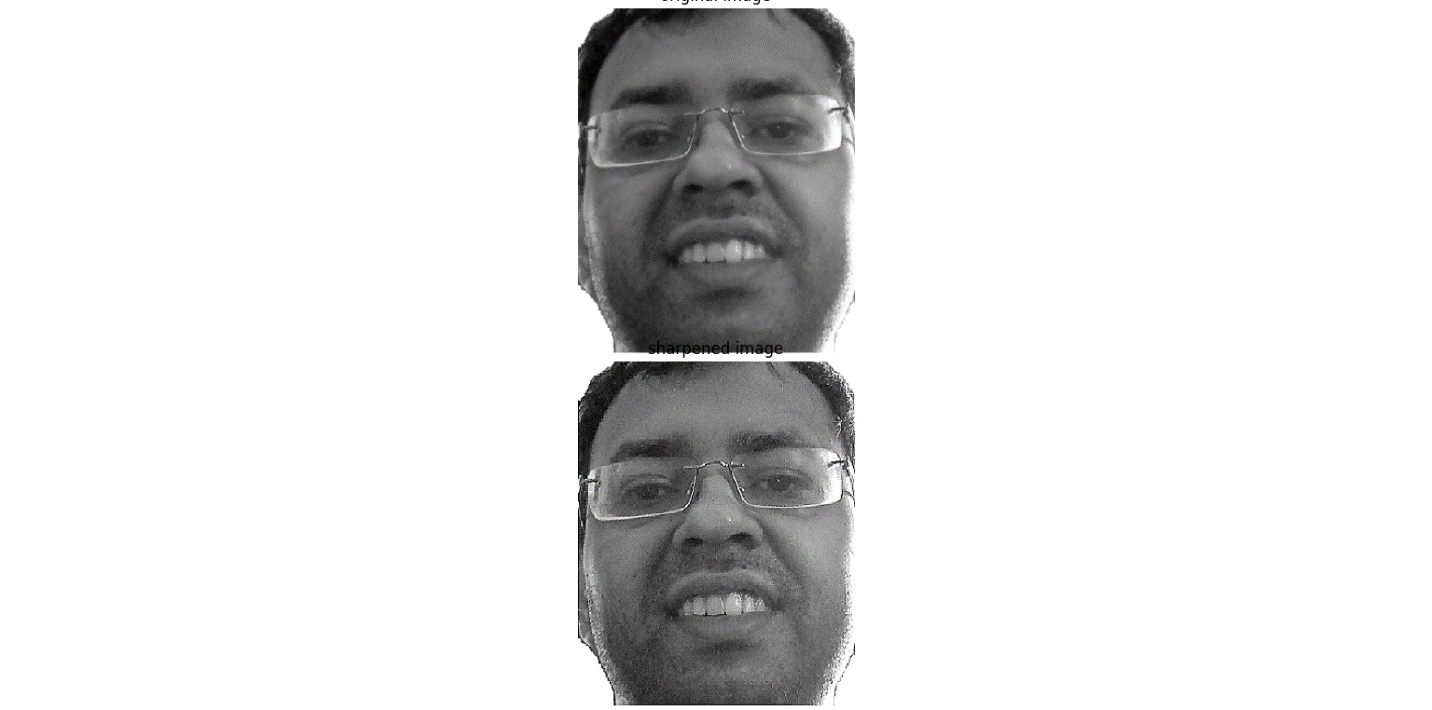
* **laplace(im)**: Áp dụng bộ lọc Laplacian lên ảnh xám. Bộ lọc Laplacian giúp tìm ra các vùng có sự thay đổi mạnh mẽ về cường độ sáng trong ảnh (các cạnh và chi tiết).
* **laplace(im) + im**: Cộng ảnh gốc với ảnh đã qua bộ lọc Laplacian. Phương pháp này giúp tăng cường các chi tiết được phát hiện bởi bộ lọc Laplacian, từ đó làm sắc nét ảnh.
* **np.clip(..., 0, 1)**: Giới hạn các giá trị pixel trong phạm vi [0, 1]. Điều này là cần thiết vì các giá trị pixel có thể vượt quá phạm vi hợp lệ sau khi cộng ảnh gốc và ảnh đã lọc, và np.clip() giúp điều chỉnh lại các giá trị này.

**Hiển thị ảnh gốc và ảnh sau khi làm sắc nét**

* **pylab.figure(figsize=(20, 30))**: Tạo một cửa sổ hiển thị ảnh với kích thước lớn để dễ dàng quan sát.
* **pylab.subplot(211)**: Chia cửa sổ thành 2 hàng, 1 cột (2x1) và chọn vị trí đầu tiên (hàng 1, cột 1) để hiển thị ảnh gốc.
* **plot\_image(im, 'original image')**: Gọi hàm plot\_image để hiển thị ảnh gốc.
* **pylab.subplot(212)**: Chọn vị trí thứ hai (hàng 2, cột 1) để hiển thị ảnh sau khi đã làm sắc nét.
* **plot\_image(im1, 'sharpened image')**: Gọi hàm plot\_image để hiển thị ảnh đã được làm sắc nét.
* **pylab.tight\_layout()**: Đảm bảo rằng các hình ảnh và tiêu đề không bị chồng lên nhau trong cửa sổ hiển thị.
* **pylab.show()**: Hiển thị các hình ảnh trong cửa sổ.

4,Kết quả sau khi thực hiện thuật toán

Ví dụ 1 : 

Ví dụ 2 : 

Ví dụ 3 : 

Ví dụ 4 : 

Nhận xét ảnh đầu ra :

* **Ảnh gốc** sẽ có thể chứa các chi tiết mờ và thiếu rõ ràng, đặc biệt là ở các vùng chuyển tiếp giữa các đối tượng và nền.
* **Ảnh sau khi làm sắc nét**: Bộ lọc Laplacian làm nổi bật các cạnh và chi tiết trong ảnh, giúp các đối tượng và đường nét trong ảnh trở nên rõ ràng hơn. Các chi tiết mờ hoặc bị phẳng trong ảnh gốc sẽ được làm sắc nét hơn, giúp ảnh trở nên sống động hơn và dễ nhận diện các đối tượng trong ảnh.

Tuy nhiên, nếu ảnh có nhiều nhiễu (noise), bộ lọc Laplacian có thể làm tăng sự hiện diện của nhiễu này, vì vậy cần phải sử dụng cẩn thận khi xử lý các ảnh có nhiễu mạnh.

THUẬT TOÁN 2 : PHÁT HIỆN CẠNH VỚI SOBEL TRONG SCIKIT-IMAGE

1.Giới thiệu thuật toán :

**Thuật toán Sobel** là một phương pháp phổ biến trong xử lý ảnh để phát hiện các cạnh trong một bức ảnh. Thuật toán này sử dụng các toán tử đạo hàm bậc nhất để tính toán gradient của ảnh tại mỗi điểm pixel. Bằng cách này, Sobel giúp xác định các thay đổi mạnh mẽ trong độ sáng của ảnh, điều này có thể chỉ ra các cạnh của đối tượng trong ảnh.

Sobel hoạt động bằng cách áp dụng các bộ lọc (hoặc kernel) đặc biệt cho ảnh để tính toán đạo hàm bậc nhất theo hai hướng: **ngang** và **dọc**. Các bộ lọc Sobel này sẽ làm nổi bật các thay đổi lớn về độ sáng trong ảnh, qua đó phát hiện các cạnh.

* **sobel\_h()**: Áp dụng bộ lọc Sobel theo chiều ngang, giúp phát hiện các cạnh có sự thay đổi theo chiều ngang.
* **sobel\_v()**: Áp dụng bộ lọc Sobel theo chiều dọc, giúp phát hiện các cạnh có sự thay đổi theo chiều dọc.
* **sobel()**: Tính toán độ lớn của gradient tổng hợp từ các bộ lọc Sobel ngang và dọc, giúp phát hiện các cạnh trong cả hai hướng.

2,Các bước thực hiện thuật toán :

 B1:**Chuyển ảnh sang ảnh xám (Grayscale)**:

* Ảnh màu được chuyển sang ảnh xám (grayscale) để giảm độ phức tạp và chỉ xử lý các giá trị độ sáng của pixel thay vì các kênh màu.

 B2: **Áp dụng các bộ lọc Sobel**:

* **sobel\_h(im)**: Áp dụng bộ lọc Sobel theo chiều ngang để tìm các cạnh theo phương ngang trong ảnh.
* **sobel\_v(im)**: Áp dụng bộ lọc Sobel theo chiều dọc để tìm các cạnh theo phương dọc trong ảnh.
* **sobel(im)**: Tính toán độ lớn của gradient tổng hợp từ các kết quả của sobel\_h và sobel\_v, để phát hiện các cạnh theo cả hai phương (ngang và dọc).

 B3: **Hiển thị kết quả**:

* Hiển thị ảnh gốc, ảnh sau khi áp dụng bộ lọc Sobel theo chiều ngang, chiều dọc, và ảnh độ lớn gradient (cạnh tổng hợp).

3,Giải thích đoạn mã nguồn :

from skimage import filters  
import pylab  
from skimage.color import rgb2gray  
from skimage.io import imread  
import matplotlib.pyplot as plt # Thêm matplotlib.pyplot  
  
# Định nghĩa hàm plot\_image để hiển thị ảnh  
def plot\_image(img, title):  
 plt.imshow(img, cmap='gray') # Hiển thị ảnh với màu xám  
 plt.title(title) # Đặt tiêu đề cho ảnh  
 plt.axis('off') # Ẩn trục (axis)  
  
# Đọc ảnh và chuyển thành ảnh xám (grayscale)  
im = rgb2gray(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\tajmahal.jpg")) # RGB image to gray scale  
  
# Áp dụng các bộ lọc Sobel  
pylab.gray()  
pylab.figure(figsize=(20, 18))  
  
pylab.subplot(2, 2, 1)  
plot\_image(im, 'original')  
  
pylab.subplot(2, 2, 2)  
edges\_x = filters.sobel\_h(im)  
plot\_image(edges\_x, 'sobel\_x')  
  
pylab.subplot(2, 2, 3)  
edges\_y = filters.sobel\_v(im)  
plot\_image(edges\_y, 'sobel\_y')  
  
pylab.subplot(2, 2, 4)  
edges = filters.sobel(im)  
plot\_image(edges, 'sobel')  
  
# Điều chỉnh khoảng cách giữa các hình ảnh  
pylab.subplots\_adjust(wspace=0.1, hspace=0.1)  
  
# Hiển thị các hình ảnh  
pylab.show()

 **Chuyển ảnh sang ảnh xám**:

* Ảnh màu được chuyển sang ảnh xám bằng rgb2gray(imread(...)). Điều này giúp giảm bớt độ phức tạp trong việc phát hiện cạnh vì chỉ cần xử lý thông tin độ sáng.

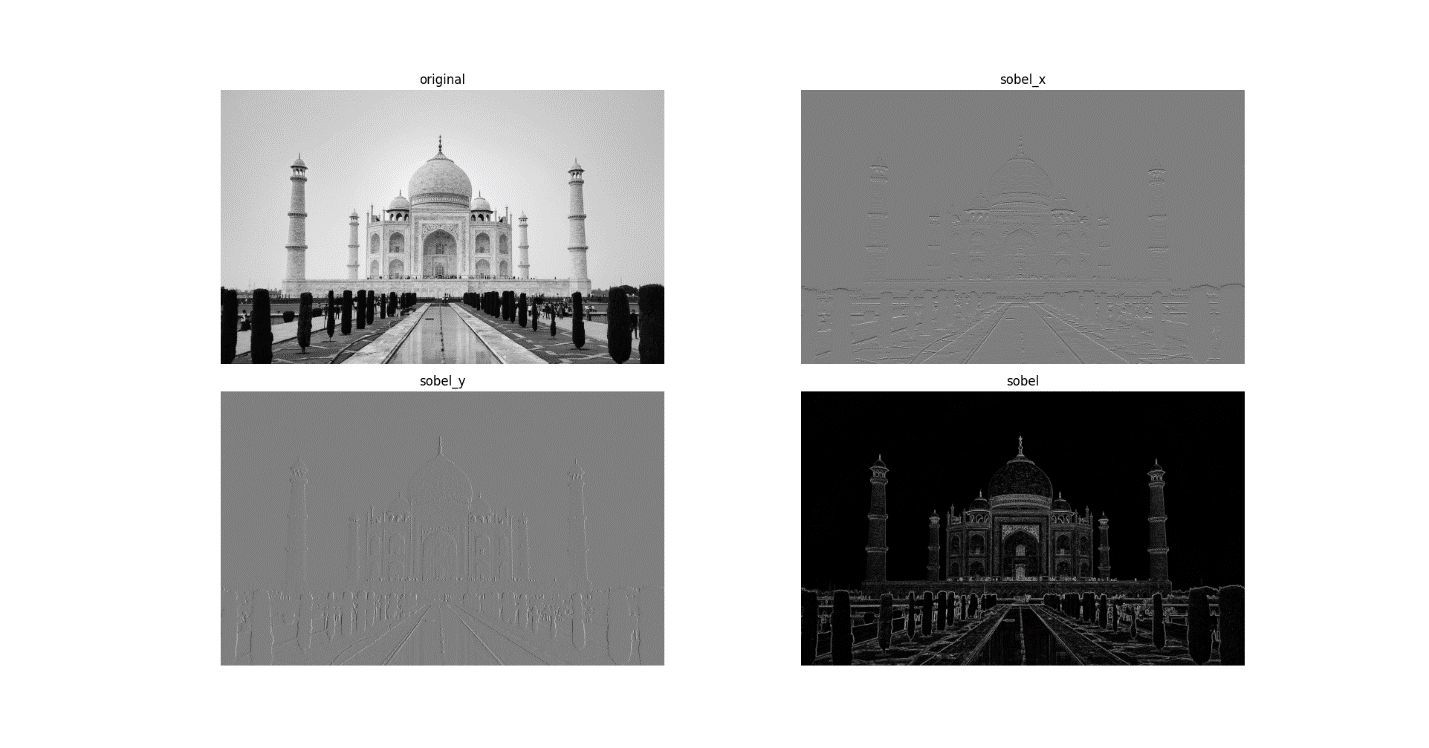
 **Áp dụng bộ lọc Sobel**:

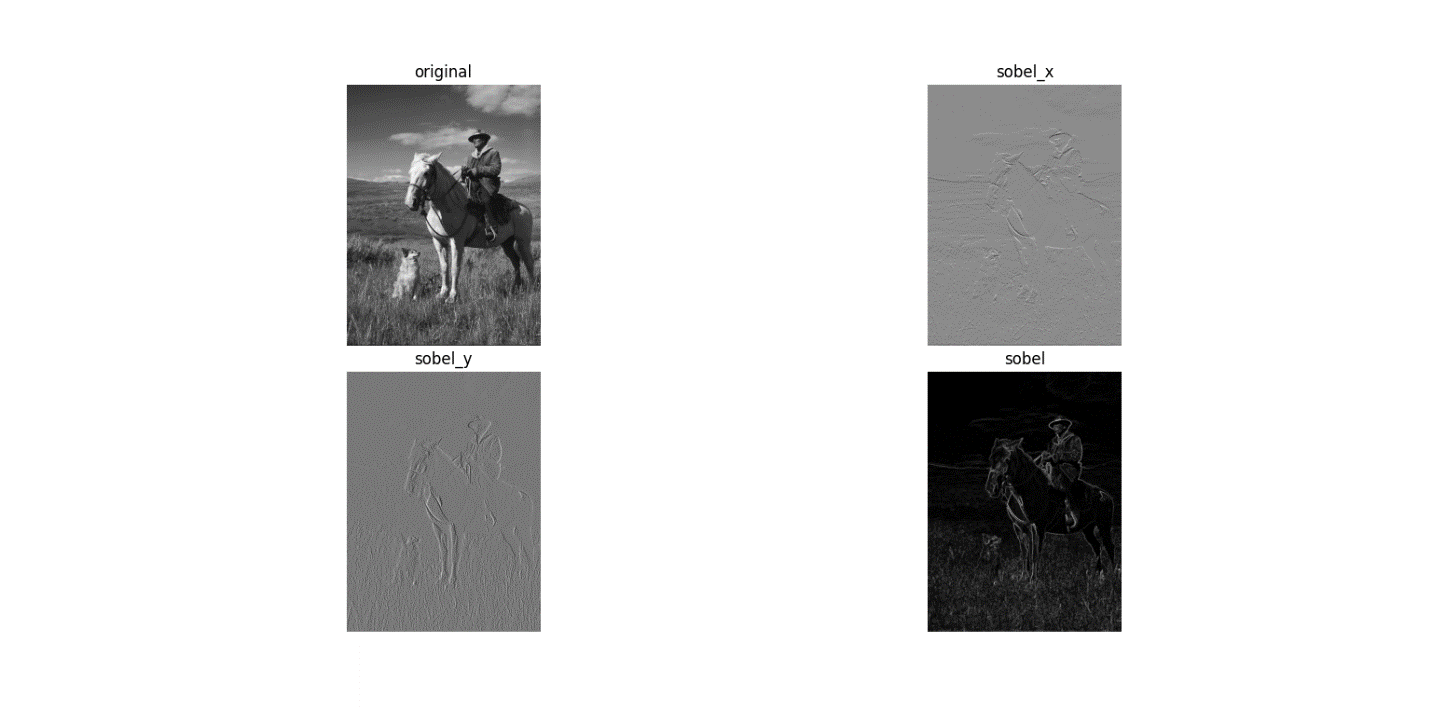
* filters.sobel\_h(im): Tính toán các cạnh theo chiều ngang bằng cách áp dụng bộ lọc Sobel.
* filters.sobel\_v(im): Tính toán các cạnh theo chiều dọc bằng bộ lọc Sobel.
* filters.sobel(im): Tính toán tổng hợp độ lớn của gradient từ các cạnh theo cả hai chiều ngang và dọc.

 **Hiển thị kết quả**:

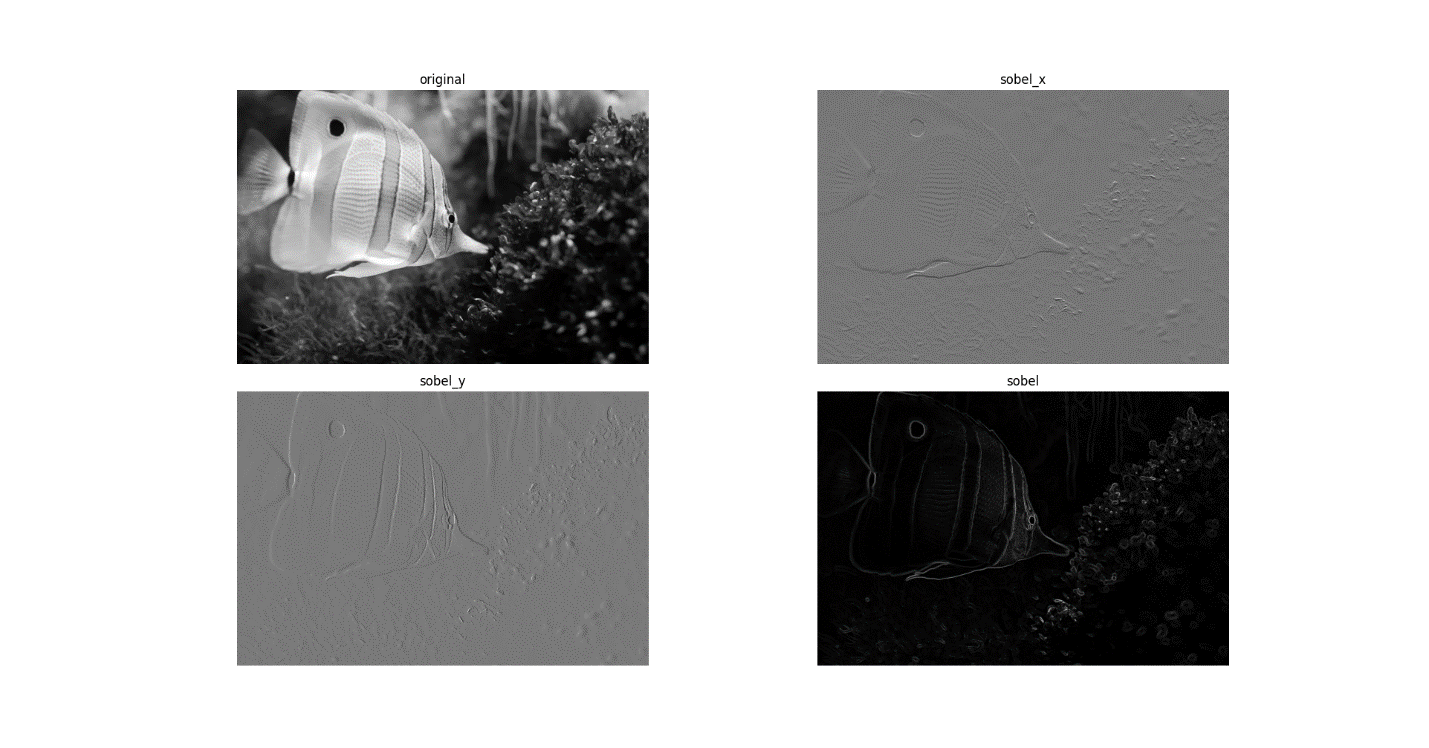
* Kết quả của mỗi bước được hiển thị trong các ô con khác nhau của đồ họa, sử dụng subplot để sắp xếp các ảnh gốc và các ảnh sau khi áp dụng bộ lọc Sobel theo các phương khác nhau

4 , Kết quả sau khi thực hiện thuật toán :

Ví dụ 1 : 

Ví dụ 2 : 

Ví dụ 3 :



**Nhận xét về ảnh đầu ra:**

Kết quả của các bộ lọc Sobel sẽ cho thấy các cạnh trong ảnh theo các phương ngang và dọc:

* **Ảnh gốc**: Hiển thị ảnh gốc của Taj Mahal dưới dạng ảnh xám.
* **sobel\_x**: Cạnh được phát hiện theo phương ngang, làm nổi bật các chuyển động hoặc thay đổi độ sáng theo chiều ngang.
* **sobel\_y**: Cạnh được phát hiện theo phương dọc, làm nổi bật các chuyển động hoặc thay đổi độ sáng theo chiều dọc.
* **sobel**: Độ lớn của gradient tổng hợp từ cả hai phương ngang và dọc, cho phép phát hiện các cạnh trong toàn bộ bức ảnh.

Các cạnh xuất hiện rõ ràng trên ảnh sau khi áp dụng bộ lọc Sobel, với các đường viền của các đối tượng trong ảnh như tòa nhà, cổng, cây cối được làm nổi bật.

THUẬT TOÁN 3 : BỘ LỌC Log VỚI MÔ ĐUN SCIPY ndimage

**1. Giới thiệu về bộ lọc LoG (Laplacian of Gaussian):**

* Bộ lọc **Laplacian of Gaussian (LoG)** kết hợp hai bước xử lý:
  + **Bộ lọc Gaussian**: Làm mịn ảnh để giảm nhiễu.
  + **Bộ lọc Laplacian**: Phát hiện các cạnh bằng cách tính toán sự thay đổi nhanh chóng của độ sáng trong ảnh.
* Bộ lọc LoG có thể phát hiện các cạnh, đồng thời giúp làm mượt các nhiễu trong ảnh, điều này làm cho nó trở thành một công cụ hữu ích trong xử lý ảnh.

**2. Các bước thực hiện thuật toán:**

**Bước 1: Đọc ảnh và chuyển đổi sang ảnh xám (grayscale)**

* Ảnh màu được chuyển đổi sang ảnh xám để đơn giản hóa việc xử lý (vì ảnh xám chỉ chứa thông tin độ sáng).

**Bước 2: Áp dụng bộ lọc Gaussian Laplace (LoG)**

* Hàm ndimage.gaussian\_laplace của SciPy được sử dụng để áp dụng bộ lọc LoG. Tham số sigma điều chỉnh mức độ làm mượt của bộ lọc Gaussian. Giá trị sigma càng lớn thì ảnh càng bị làm mượt mạnh mẽ hơn.

**Bước 3: Hiển thị kết quả**

* Sau khi áp dụng bộ lọc, ảnh kết quả được cắt giá trị pixel trong khoảng [0, 1] (để đảm bảo không có giá trị ngoài phạm vi màu sắc hợp lệ) và hiển thị dưới dạng ảnh xám.

**Bước 4: Lặp lại với các giá trị khác nhau của sigma**

* Bộ lọc được áp dụng nhiều lần với các giá trị khác nhau của sigma để xem sự ảnh hưởng của việc điều chỉnh mức độ làm mượt tới kết quả.

**Bước 5: Hiển thị tất cả các ảnh kết quả**

* Sau khi lặp qua các giá trị sigma, tất cả các ảnh kết quả được hiển thị trong một cửa sổ hình ảnh, với các ảnh được phân chia thành các ô (subplot) trong cùng một figure.

**3,Giải thích đoạn mã nguồn :**

from skimage.color import rgb2gray # Import rgb2gray from skimage.color  
from skimage.io import imread  
from scipy import ndimage  
import pylab  
import numpy as np  
  
# Đọc ảnh và chuyển thành ảnh xám (grayscale)  
img = rgb2gray(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\zebras.jpg"))  
  
# Tạo một hình ảnh với kích thước lớn  
fig = pylab.figure(figsize=(25, 15))  
pylab.gray() # Hiển thị kết quả bộ lọc dưới dạng ảnh xám  
  
# Áp dụng bộ lọc Gaussian Laplace với các giá trị sigma khác nhau  
for sigma in range(1, 10):  
 pylab.subplot(3, 3, sigma)  
 img\_log = ndimage.gaussian\_laplace(img, sigma=sigma) # Áp dụng Gaussian Laplace  
 pylab.imshow(np.clip(img\_log, 0, 1)) # Hiển thị ảnh đã lọc, cắt giá trị trong khoảng [0,1]  
 pylab.axis('off') # Tắt trục  
 pylab.title('LoG with sigma=' + str(sigma), size=20) # Đặt tiêu đề cho từng ảnh  
  
# Hiển thị các ảnh  
pylab.show()

**Nhập các thư viện cần thiết:**

* **rgb2gray**: Dùng để chuyển đổi ảnh màu sang ảnh xám (grayscale).
* **imread**: Đọc ảnh từ đường dẫn đã chỉ định.
* **ndimage**: Thư viện của **SciPy** cung cấp các chức năng xử lý ảnh, bao gồm **gaussian\_laplace** để áp dụng bộ lọc Gaussian Laplace.
* **pylab**: Một phần của **Matplotlib**, dùng để vẽ đồ thị và hiển thị hình ảnh.
* **numpy**: Dùng để thực hiện các phép toán trên mảng, đặc biệt là việc xử lý ảnh dưới dạng mảng.

**Đọc và chuyển ảnh sang ảnh xám:**

* Đọc ảnh từ đường dẫn đã cho ("C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\zebras.jpg").
* Chuyển ảnh màu (RGB) thành ảnh xám bằng cách sử dụng hàm rgb2gray. Điều này giúp giảm độ phức tạp của việc xử lý, vì ảnh xám chỉ có một kênh độ sáng.

**Tạo cửa sổ hình ảnh lớn:**

* fig = pylab.figure(figsize=(25, 15)): Tạo một cửa sổ hình ảnh có kích thước lớn (25x15 inch).
* pylab.gray(): Đặt chế độ hiển thị màu cho các ảnh sau này là **xám** (grayscale), điều này giúp dễ dàng nhận diện các thay đổi độ sáng trong ảnh.

**Áp dụng bộ lọc Gaussian Laplace (LoG) với các giá trị sigma khác nhau:**

* **for sigma in range(1, 10)**: Lặp qua các giá trị sigma từ 1 đến 9. sigma là tham số làm mượt của bộ lọc Gaussian. Các giá trị sigma lớn hơn sẽ làm mượt ảnh mạnh hơn.
* **ndimage.gaussian\_laplace(img, sigma=sigma)**: Áp dụng bộ lọc Gaussian Laplace (LoG) vào ảnh với giá trị sigma hiện tại. Bộ lọc LoG giúp phát hiện các cạnh trong ảnh đồng thời làm mượt các nhiễu.
* **pylab.subplot(3, 3, sigma)**: Chia cửa sổ hình ảnh thành 3x3 ô (subplots) và chọn ô số sigma (từ 1 đến 9).
* **pylab.imshow(np.clip(img\_log, 0, 1))**: Hiển thị ảnh đã lọc. Hàm np.clip giới hạn giá trị pixel của ảnh trong khoảng từ 0 đến 1 để tránh các giá trị ngoài phạm vi hợp lệ.
* **pylab.axis('off')**: Tắt trục (axes) của các ô ảnh, chỉ hiển thị ảnh.
* **pylab.title('LoG with sigma=' + str(sigma), size=20)**: Đặt tiêu đề cho mỗi ô ảnh, hiển thị giá trị của sigma đã sử dụng.

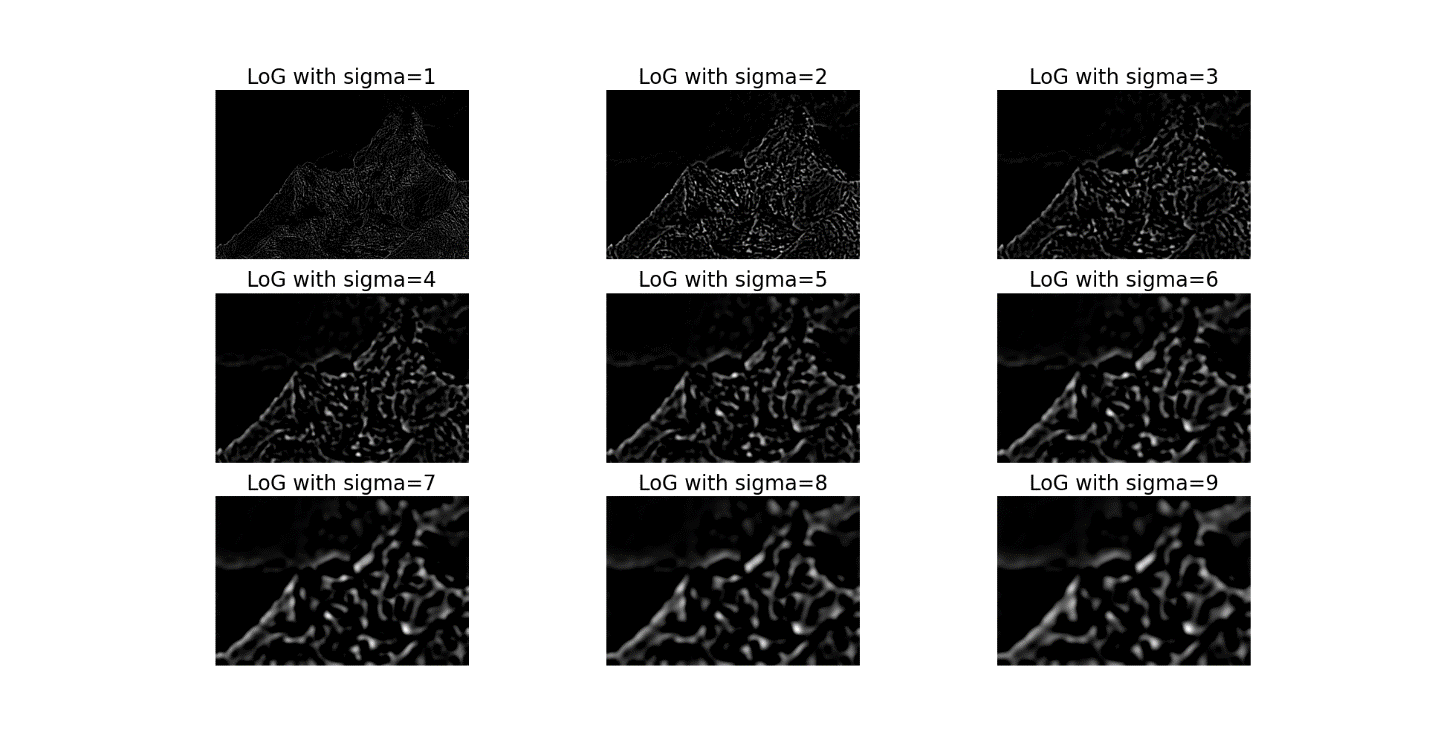
**Hiển thị tất cả các ảnh:**

* Sau khi tất cả các ảnh đã được xử lý và hiển thị, câu lệnh pylab.show() sẽ hiển thị cửa sổ đồ họa với tất cả các ảnh đã được tạo ra.

**4,Kết quả sau khi thực hiện thuật toán : Ví dụ 1 :**

**Input Image **

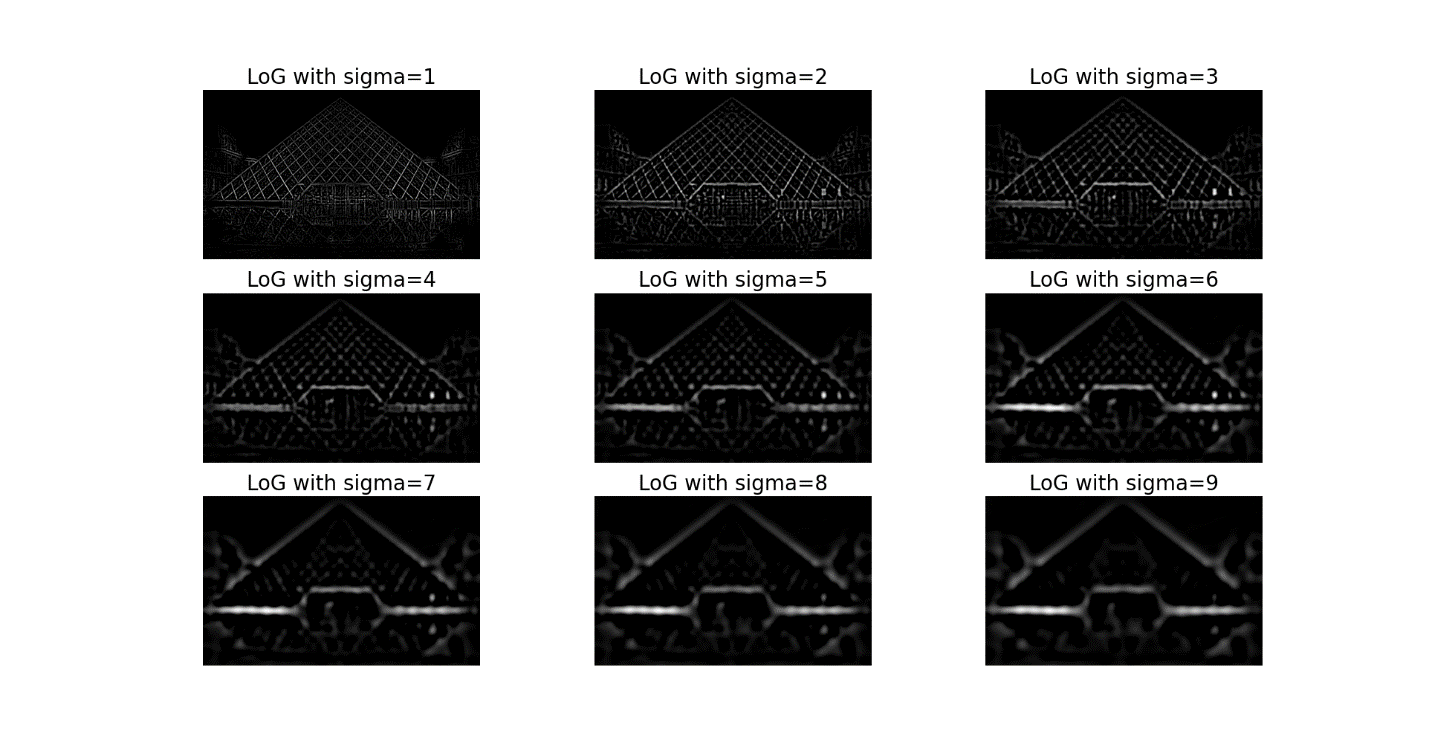
**Output Image :**

****

**Ví dụ 2 : Input Image**

****

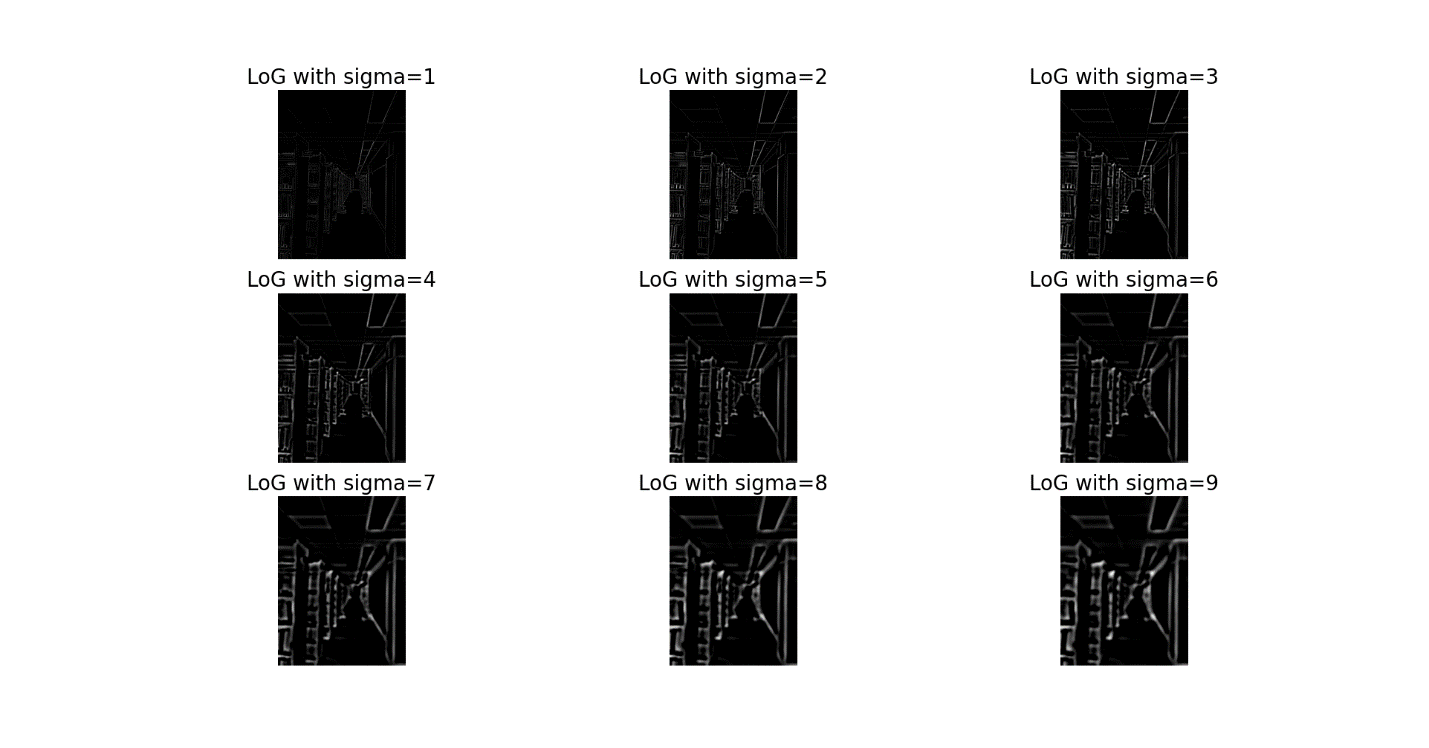
**Output Image :**

****

**Ví dụ 3 :**

**Input Image : **

**Output Image :**

****

**Nhận xét về ảnh đầu ra:**

Kết quả của quá trình áp dụng bộ lọc LoG là các ảnh có các cạnh rõ rệt, với mức độ làm mượt thay đổi tùy theo giá trị sigma. Cụ thể:

* Khi **sigma nhỏ**, ảnh có thể bị nhiễu nhưng giữ được chi tiết hình ảnh cao. Khi sigma nhỏ, bộ lọc Gaussian không làm mờ ảnh quá nhiều, và bộ lọc Laplace có thể phát hiện được các biên rất chi tiết. Ảnh đầu ra sẽ có các biên rõ ràng và các chi tiết nhỏ vẫn được giữ nguyên. Tuy nhiên, ảnh có thể bị nhiễu nhiều hơn vì bộ lọc không làm mờ quá nhiều.
* Khi **sigma lớn**, ảnh trở nên mượt mà hơn, nhưng có thể mất đi một số chi tiết, đặc biệt là các cạnh nhỏ. Khi sigma lớn, bộ lọc Gaussian làm mờ ảnh nhiều hơn, khiến các biên mờ nhạt đi. Kết quả là ảnh đầu ra sẽ không rõ ràng và các chi tiết nhỏ cũng bị mất đi. Bộ lọc Laplace hoạt động tốt hơn trong việc phát hiện biên lớn, nhưng không thể phát hiện các biên nhỏ và chi tiết.
* **Biên sắc nét**: Ở những giá trị sigma nhỏ, các biên trong ảnh vẫn rõ nét, và bạn có thể nhìn thấy các vùng chuyển tiếp đột ngột trong độ sáng. Điều này giúp bạn dễ dàng nhận ra các chi tiết về kết cấu của ảnh, ví dụ như các cạnh của các vật thể hoặc đường biên trong ảnh.
* **Biên mờ**: Với các giá trị sigma lớn, bộ lọc Gaussian làm mờ ảnh nhiều hơn, làm cho các biên trong ảnh trở nên mờ nhạt. Ảnh sẽ mất đi một phần chi tiết về cấu trúc, nhưng sẽ làm mượt các điểm nhiễu nhỏ không quan trọng trong ảnh. Tuy nhiên, các biên lớn vẫn có thể được nhận diện.
* **Sự thay đổi của các giá trị sigma**: Sự thay đổi của sigma sẽ điều chỉnh mức độ mờ của ảnh. Sigma nhỏ tạo ra ảnh sắc nét với các biên rõ ràng hơn, trong khi sigma lớn làm mờ các biên và chi tiết, phù hợp cho việc phát hiện các biên lớn hoặc giảm nhiễu.

Việc điều chỉnh sigma giúp tìm được sự cân bằng giữa việc làm mượt và phát hiện các cạnh rõ ràng, và điều này rất quan trọng trong các ứng dụng như phân tích ảnh hoặc nhận dạng đối tượng.