

DẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH



BÀI TẬP LỚN 2
XỬ LÝ ẢNH SỐ VÀ THỊ GIÁC MÁY TÍNH

LOW PASS
AND HIGH PASS FILTER
FOR IMAGE

GVHD: Ths. Võ Thành Hùng
SV thực hiện: Võ Tân Hưng – 2113623



Mục lục

1 Giới thiệu	2
1.1 Low Pass Filter	2
1.2 High Pass Filter	3
2 Hiện thực	5
2.1 Cơ sở lý thuyết	5
2.1.1 Low Pass Filter	5
2.1.2 High Pass Filter	6
2.2 Hiện thực với python	7
2.2.1 Hàm tích chập cho ảnh với kernel	7
2.2.2 Create Box Kernel	7
2.2.3 Add Gaussian Noise	8
2.2.4 Create Gaussian Kernel	8
2.2.5 Add Salt and Pepper Noise	8
2.2.6 Median Filter	9
2.2.7 Laplacian Kernel	9
2.2.8 Laplacian Filter	9
2.2.9 Sobel Kernel	10
2.2.10 Sobel Filter	10
3 Kết quả	12
3.1 Áp dụng Box Filter	12
3.2 Áp dụng Gaussian Filter	13
3.3 Áp dụng Median Filter	14
3.4 Áp dụng Laplacian Filter	15
3.5 Áp dụng Sobel Filter	16
4 Nhận xét & Kết luận	17
4.1 Nhận xét	17
4.1.1 Bộ Lọc Trung Bình (Box Blur)	17
4.1.2 Bộ Lọc Gaussian	17
4.1.3 Bộ Lọc Trung Vị (Median)	17
4.1.4 Bộ Lọc Laplace Rời Rạc	17
4.1.5 Bộ Lọc Sobel	17
4.2 Kết luận	17

1 Giới thiệu

Trong xử lý ảnh số, việc sử dụng các bộ lọc là một phần quan trọng của quá trình xử lý ảnh. Bộ lọc có thể được sử dụng để cải thiện chất lượng ảnh, loại bỏ nhiễu và tăng cường đặc điểm của hình ảnh. Trong báo cáo này, chúng ta sẽ tìm hiểu về hai loại bộ lọc chính: **Low Pass Filters** và **High Pass Filters**, cùng với ứng dụng của chúng trong xử lý ảnh.

1.1 Low Pass Filter

Low Pass Filter là một loại bộ lọc thông qua các tần số thấp trong một tín hiệu và loại bỏ các tần số cao. Trong xử lý ảnh, việc áp dụng Low Pass Filter dẫn đến việc làm mờ hình ảnh và làm giảm chi tiết. Cụ thể, các bước chính trong việc áp dụng Low Pass Filter bao gồm:

- **Chuẩn bị kernel Low Pass Filter:** Kernel này thường được thiết kế dưới dạng ma trận có trọng số. Các giá trị trong kernel quyết định cách bộ lọc xử lý hình ảnh.

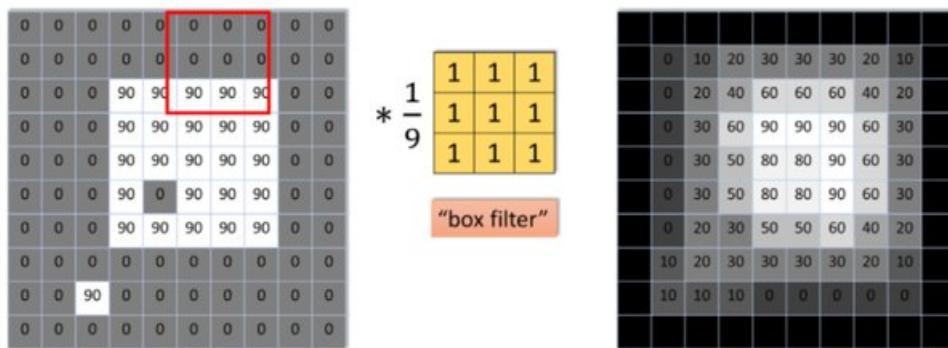
$$\begin{array}{c}
 \text{(a) } F_a \text{ Filter} \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{(b) } F_{hv} \text{ Filter} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{(c) } F_d \text{ Filter} \quad \begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix} \\
 \text{(d) } F_{SV} \text{ Filter} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad \text{(e) } F_{SH} \text{ Filter} \quad \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}
 \end{array}$$

Hình 1: Ví dụ cho các 3×3 kernel

- **Áp dụng kernel vào hình ảnh:** Kernel được trượt qua từng điểm ảnh trên hình ảnh gốc. Mỗi điểm ảnh mới được tính toán dựa trên giá trị của kernel và các điểm ảnh xung quanh.

Averaging filter

$$F(x, y) * H(u, v) = G(x, y)$$



$$G = F * H$$

Hình 2: Áp dụng filter cho hình ảnh

- Kết quả là một hình ảnh mới đã được làm mờ, giảm nhiễu và loại bỏ các chi tiết cụ thể.



Hình 3: Ảnh trộn nén mờ hơn sau khi áp dụng filter

- **Ứng dụng:** làm mờ hình ảnh và giảm nhiễu, làm cho hình ảnh trở nên mịn màng và có dáng vẻ mềm mại hơn. Điều này hữu ích trong việc chụp ảnh portraiture để làm mịn da hoặc trong các bức ảnh nơi nhiễu có thể gây xáo trộn vì thế được sử dụng trong nhiều lĩnh vực như xử lý ảnh y tế, xử lý hình ảnh video, và trong công nghệ camera để cải thiện chất lượng hình ảnh.

1.2 High Pass Filter

High Pass Filter là một loại bộ lọc thông qua các tần số cao trong một tín hiệu và loại bỏ các tần số thấp. Trong xử lý ảnh, High Pass Filter thường được sử dụng để làm nổi bật các chi tiết và cạnh trong hình ảnh. Tương tự như [Low Pass Filter](#), các bước cơ bản để áp dụng High Pass Filter cũng bao gồm:

- **Chuẩn bị kernel High Pass Filter:** Kernel này thường được thiết kế để tìm ra sự khác biệt giữa các điểm ảnh.

$$\frac{1}{3} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -3 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \frac{1}{3} \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -3 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \frac{1}{4} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 2 & -4 & 2 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(a) *HSB*

(b) VBH

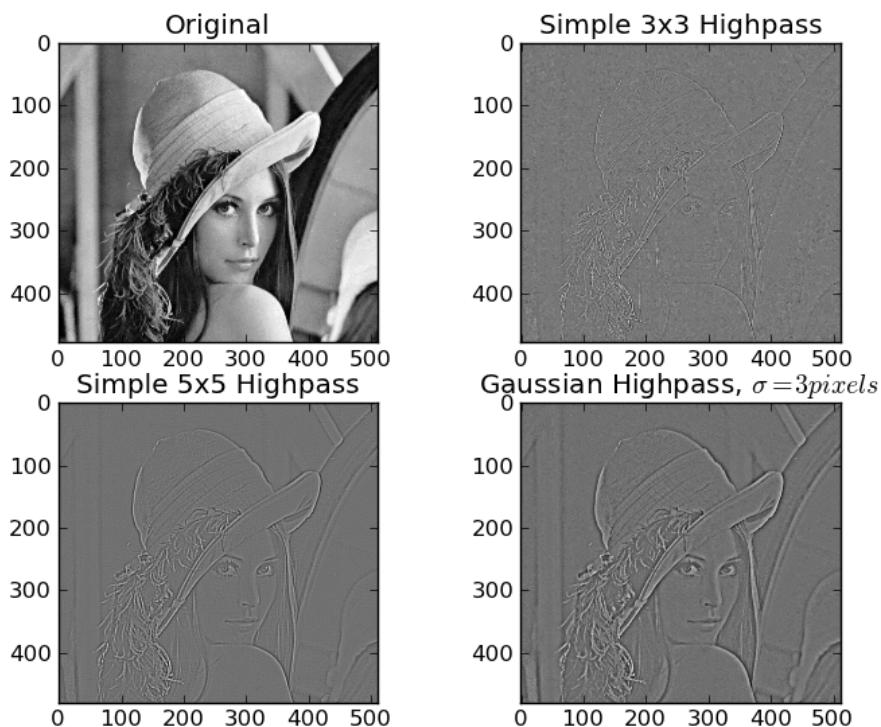
(c) *RF0*

$$\frac{1}{12} \begin{pmatrix} -1 & 2 & -2 & 2 & -1 \\ 2 & -6 & 8 & -6 & 2 \\ -2 & 8 & -12 & 8 & -2 \\ 2 & -6 & 8 & -6 & 2 \\ 1 & -2 & 2 & -2 & 1 \end{pmatrix} \quad \frac{1}{2} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(1) BE1

(c) BFE

- **Áp dụng kernel vào hình ảnh:** Kernel được áp dụng tương tự như trong Low Pass Filter, nhưng ở đây, mục tiêu là làm nổi bật các chi tiết.
 - Kết quả là một hình ảnh mới với các cạnh và chi tiết được nổi bật hơn.



Hình 5: Ví dụ phát hiện cạnh

- **Ứng dụng:** làm nổi bật các cạnh và chi tiết trong hình ảnh bằng cách tăng độ tương phản giữa các vùng khác nhau. Ngoài ra còn giúp trong việc trích xuất các đặc trưng quan trọng từ hình ảnh, giúp trong quá trình nhận dạng và phân loại.



2 Hiện thực

2.1 Cơ sở lý thuyết

2.1.1 Low Pass Filter

Chúng ta sẽ tập trung vào ba loại LPF phổ biến: Box Blur, Gaussian Filter và Median Filter.

Box Filter:

Box Filter là một loại LPF đơn giản, nó làm mờ hình ảnh bằng cách lấy trung bình cộng của các pixel trong một khu vực xung quanh.

Cách xác định kernel cho Box Blur:

- Xác định kích thước của ô (kernel) cho việc làm mờ.
- Đặt giá trị của mỗi ô trong kernel thành 1 (array ones).

$$\frac{1}{n} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \dots & \dots & \dots & \dots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

Ma trận Box Blur kích thước nxn

Gaussian Filter:

Low Pass Gaussian Filter là một bộ lọc tuyến tính (linear filter) sử dụng kernel dựa trên phân phối Gaussian để làm mờ hình ảnh và loại bỏ nhiễu Gaussian.

Cách xác định kernel cho Gaussian Filter:

- Xác định các tham số của ô (kernel) cho việc làm mờ.
- Kernel của Gaussian Low Pass Filter:

$$K = \frac{1}{s} \cdot \begin{bmatrix} g_{11} & g_{12} & \dots & g_{1n} \\ g_{21} & g_{22} & \dots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & \dots & g_{nn} \end{bmatrix}$$

Trong đó:

s là hệ số chuẩn hóa để tổng của tất cả các giá trị trong kernel bằng 1,

g_{ij} là giá trị của hàm Gaussian tại vị trí (i, j) của kernel,

được tính bằng công thức $e^{-\frac{(i-(n+1)/2)^2 + (j-(n+1)/2)^2}{2\sigma^2}}$,

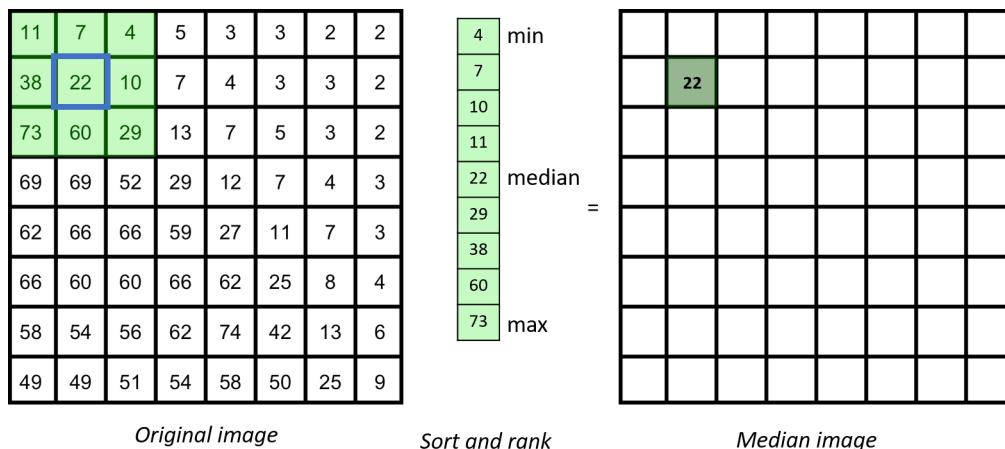
n là kích thước của kernel (số hàng hoặc số cột),

σ là tham số độ rộng của Gaussian, quyết định mức độ làm mờ của filter.

Median Filter:

Median filter dựa trên ý tưởng sử dụng giá trị trung vị của các pixel trong một vùng xung quanh (vùng kernel) để thay thế giá trị của pixel đang xét.

Trong Median filter, một kernel hoặc ô cửa sổ được di chuyển qua từng pixel trong hình ảnh. Tại mỗi pixel, các giá trị pixel trong vùng kernel được sắp xếp theo thứ tự tăng dần hoặc giảm dần, và giá trị trung vị của chúng được chọn là giá trị của pixel tại vị trí đó. Điều này giúp loại bỏ nhiễu và giảm hiệu ứng làm mờ mà các phương pháp khác như mean filter có thể gây ra.



Hình 6: Median Filter với kích thước kernel 3x3

2.1.2 High Pass Filter

HighPass Filter là một loại bộ lọc được sử dụng để làm nổi bật các đặc điểm cực đại trong hình ảnh, như biên cạnh và các đặc điểm sắc nét. Trong phần này, chúng ta sẽ tìm hiểu về hai phương pháp HighPass Filter phổ biến là Laplace và Sobel.

Laplace Filter:

High Pass Filter Laplace thực hiện việc phát hiện các biên cạnh và đặc điểm cực đại bằng cách tìm các điểm trong hình ảnh có độ lớn của độ dốc (gradient) lớn. Cụ thể, High Pass Filter Laplace thực hiện phép tính đạo hàm bậc hai của hình ảnh. Công thức toán học của High Pass Filter Laplace được biểu diễn như sau:

$$\text{Laplace}(f(x, y)) = \nabla^2 f(x, y)$$

Trong đó $f(x, y)$ là hình ảnh gốc và ∇^2 là toán tử Laplace.

Phương pháp trên còn gọi là Laplacian of Gaussian (Laplace liên tục). Để đơn giản thì trong báo cáo này ta chỉ áp dụng kernel cố định cho bài toán phát hiện cạnh hay còn gọi là Laplace rời rạc (Discrete Laplace).

$$L = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Trong đó L là bộ lọc Laplace rời rạc.

Sobel Filter:

High Pass Filter Sobel cũng được sử dụng để phát hiện biên cạnh trong hình ảnh. Nó thực hiện phép tính đạo hàm theo hướng x và y của hình ảnh. Công thức toán học của High Pass Filter Sobel được biểu diễn như sau:



$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * f(x, y)$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * f(x, y)$$

Trong đó $f(x, y)$ là hình ảnh gốc, G_x và G_y là gradient theo hướng x và y, và $*$ biểu thị phép tính chập.

2.2 Hiện thực với python

Có thể xem source tại đây: <https://github.com/hungvo2003vn/Computer-Vision/tree/Ass2>
Xem bài làm trên nhánh: **Ass2**

2.2.1 Hàm tích chập cho ảnh với kernel

Phép nhân tích chập 2 ma trận nếu ta hiện thực bằng các vòng for thì sẽ giảm hiệu năng tính toán, nên thay vì vậy ta sử dụng hàm `convolve2d` thuộc module `scipy.signal` của thư viện `scipy`

```
1 from scipy.signal import convolve2d
2
3 # This function use for box and gaussian filter
4 def conv2d(image, kernel, epoch=1, epochs=1):
5     print(f"Epoch {epoch}/{epochs}", end=" ")
6     print("[", end="", flush=True)
7
8     start_time = time.time()
9
10    # Iterate over each channel and convolve with the kernel
11    m, n = kernel.shape
12    y, x, z = image.shape
13    y = y - m + 1
14    x = x - n + 1
15    new_image = np.zeros((y, x, z), dtype=image.dtype)
16    for c in range(z):
17        new_image[:, :, c] = convolve2d(image[:, :, c], kernel, mode='valid')
18        print("#", end="", flush=True)
19
20    print("] - 100.00% - Total time: {:.2f}s".format(time.time() - start_time))
21
22    return new_image
```

2.2.2 Create Box Kernel

```
1 def create_box_blur_kernel(size=3):
2     # Ensure size is odd
3     size = int(size)
4     if size % 2 == 0:
5         size += 1
```



```
7 # Create the box blur kernel
8 kernel = np.ones((size, size), dtype=np.float32)
9 kernel /= size ** 2 # Normalize to ensure kernel sums up to 1
10
11 return kernel
```

2.2.3 Add Gaussian Noise

Phép thêm nhiễu Gaussian này dựa trên mean và std của từng `image` cũ hể được đưa vào hàm.

```
1 def add_gaussian_noise_rgb(image):
2
3     mean = np.mean(image)
4     std = np.std(image)
5     noise = np.random.normal(mean, std, size=image.shape)
6     brightness = 0.5
7     noisy_image = np.clip((image + noise) * brightness, 0, 255).astype(np.uint8)
8
9     return noisy_image
```

2.2.4 Create Gaussian Kernel

```
1 def create_gaussian_kernel(image, size=3):
2     # Ensure size is odd
3     size = int(size)
4     if size % 2 == 0:
5         size += 1
6
7     # Convert RGB image to grayscale
8     gray_image = np.mean(image, axis=2)
9
10    # Calculate variance and sigma
11    var = np.var(gray_image)
12    sigma = np.sqrt(var)
13
14    # Generate 1D Gaussian kernel
15    kernel_1d = np.linspace(-(size // 2), size // 2, size)
16    kernel_1d = np.exp(-(kernel_1d ** 2) / (2 * sigma ** 2))
17    kernel_1d /= np.sum(kernel_1d)
18
19    # Convert 1D kernel to 2D
20    kernel_2d = np.outer(kernel_1d, kernel_1d)
21
22    return kernel_2d
```

2.2.5 Add Salt and Pepper Noise

```
1 def add_salt_and_pepper_noise(image, salt_prob=0.01, pepper_prob=0.01):
2
3     noisy_image = np.copy(image)
4     salt_mask = np.random.rand(*image.shape[:2]) < salt_prob
5     pepper_mask = np.random.rand(*image.shape[:2]) < pepper_prob
6     noisy_image[salt_mask] = 255
```



```
7     noisy_image[pepper_mask] = 0
8
9     return noisy_image
```

2.2.6 Median Filter

Phép tìm trung vị theo bộ lọc nếu ta hiện thực bằng các vòng for thì sẽ giảm hiệu năng tính toán, nên thay vì vậy ta sử dụng hàm `median_filter` thuộc module `scipy.ndimage` của thư viện `scipy`, tính toán trên từng kênh màu của ảnh đưa vào.

```
1 from scipy.ndimage import median_filter
2
3 def median_filter_rgb(image, size=3, epoch=1, epochs=1):
4     print(f"Epoch {epoch}/{epochs}", end=" ")
5     print("[", end="", flush=True)
6
7     # Separate the image into individual color channels
8     r_channel = image[:, :, 0]
9     g_channel = image[:, :, 1]
10    b_channel = image[:, :, 2]
11
12    start_time = time.time()
13
14    # Apply median filter to each color channel
15    r_filtered = median_filter(r_channel, size=size)
16    print("#", end="", flush=True)
17    g_filtered = median_filter(g_channel, size=size)
18    print("#", end="", flush=True)
19    b_filtered = median_filter(b_channel, size=size)
20    print("#", end="", flush=True)
21
22    # Stack the filtered channels back into an RGB image
23    filtered_image = np.stack((r_filtered, g_filtered, b_filtered), axis=2)
24
25    print("] - 100.00% - Total time: {:.2f}s".format(time.time() - start_time))
26
27    return filtered_image
```

2.2.7 Laplacian Kernel

```
1 # Define high-pass filter kernel (edge detection)
2 high_pass_laplace_kernel = np.array([[-1, -1, -1],
3                                     [-1,  8, -1],
4                                     [-1, -1, -1]])
```

2.2.8 Laplacian Filter

Trước tiên, ta sử dụng hàm `cvtColor` của thư viện `cv2` để scale về ảnh xám. Tiếp theo sử dụng hàm `convolve` thuộc module `scipy.ndimage` của thư viện `scipy` để tăng hiệu năng tính toán thay vì sử dụng các vòng for để tính tích chập 2 ma trận ảnh xám và bộ lọc.

```
1 from scipy.ndimage import convolve
2
```



```
3 def laplace_edge_detection(image, kernel, epoch=1, epochs=1):
4
5     print(f"Epoch {epoch}/{epochs}", end=" ")
6     print("[", end="", flush=True)
7     start_time = time.time()
8
9     # Convert the image to grayscale
10    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
11    print("#", end="", flush=True)
12
13    # Apply Laplacian kernel to the grayscale image
14    edge_detected_image = convolve(gray_image, kernel)
15    print("#", end="", flush=True)
16
17    # Normalize edge-detected image to [0, 255]
18    edge_detected_image = cv2.normalize(edge_detected_image, None, 0, 255, cv2.NORM_MINMAX)
19    print("#", end="", flush=True)
20    print("] - 100.00% - Total time: {:.2f}s".format(time.time() - start_time))
21
22    return edge_detected_image
```

2.2.9 Sobel Kernel

```
1 # Define high-pass filter kernel (edge detection)
2 # Define Sobel kernels
3 high_pass_sobel_x_kernel = np.array([[-1, 0, 1],
4                                     [-2, 0, 2],
5                                     [-1, 0, 1]])
6
7 high_pass_sobel_y_kernel = np.array([[-1, -2, -1],
8                                     [ 0,  0,  0],
9                                     [ 1,  2,  1]])
```

2.2.10 Sobel Filter

Hàm trước tiên chuyển đổi hình ảnh RGB đầu vào thành hình ảnh xám bằng cách sử dụng hàm `rgb_to_gray`. Sau đó, nó áp dụng các kernel Sobel vào hình ảnh xám để thu được gradient theo chiều ngang và dọc. Độ lớn gradient được tính từ các gradient này. Cuối cùng, độ lớn gradient được chuẩn hóa và chuyển đổi thành biểu diễn số nguyên không dấu 8-bit.

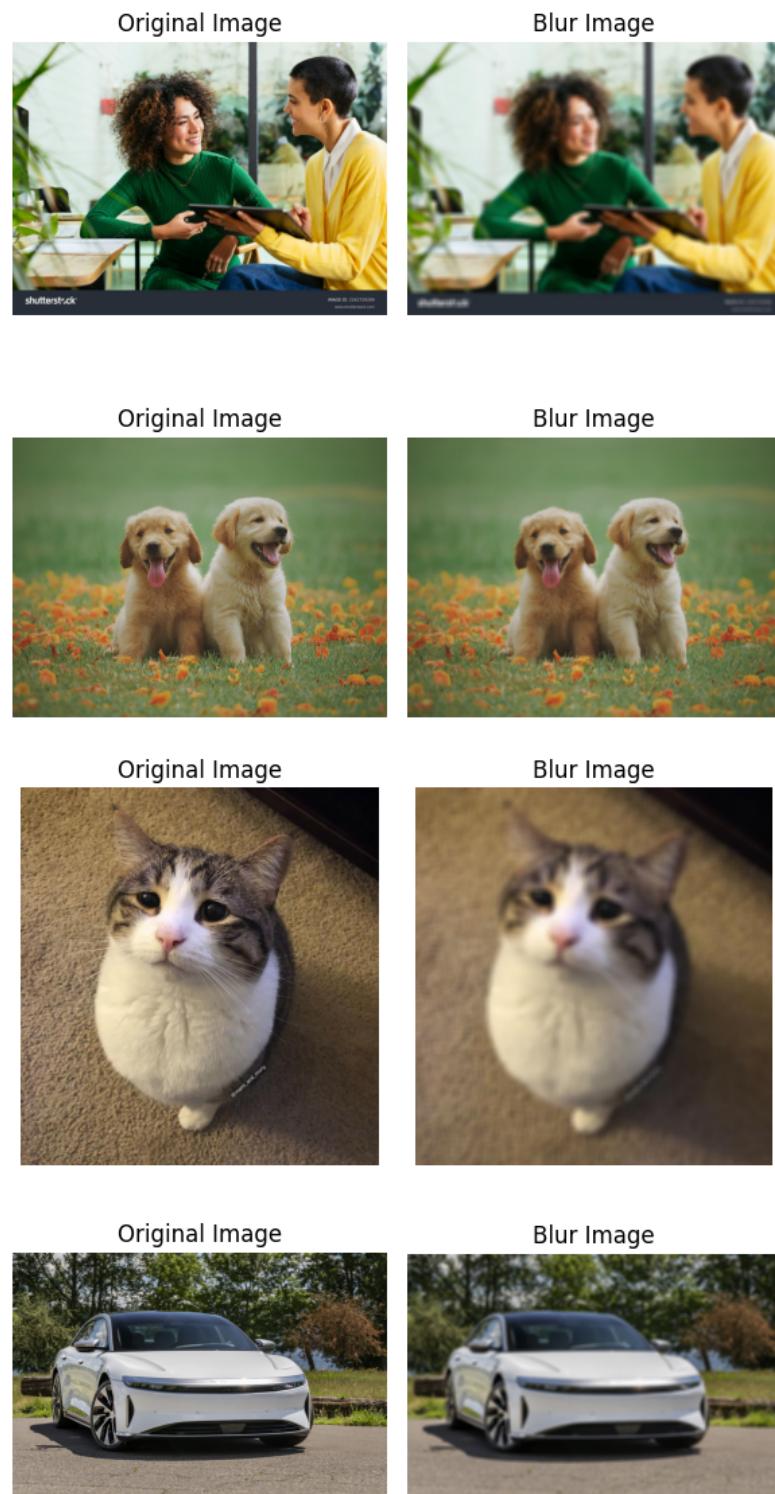
```
1 def sobel_edge_detection(image, kernel, epoch=1, epochs=1):
2
3     print(f"Epoch {epoch}/{epochs}", end=" ")
4     print("[", end="", flush=True)
5     start_time = time.time()
6
7     # Convert the image to grayscale
8     grayscale_image = rgb_to_gray(image)
9     print("#", end="", flush=True)
10
11    # Apply Sobel kernels to the grayscale image
12    sobel_x, sobel_y = kernel
13    gradient_x = convolve(grayscale_image, sobel_x)
14    print("#", end="", flush=True)
15    gradient_y = convolve(grayscale_image, sobel_y)
```



```
16     print("#", end="", flush=True)
17
18     # Calculate gradient magnitude
19     gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)
20     print("#", end="", flush=True)
21
22     gradient_magnitude = gradient_magnitude.astype(np.uint8)
23     print("#", end="", flush=True)
24     print("] - 100.00% - Total time: {:.2f}s".format(time.time() - start_time))
25
26 return gradient_magnitude
```

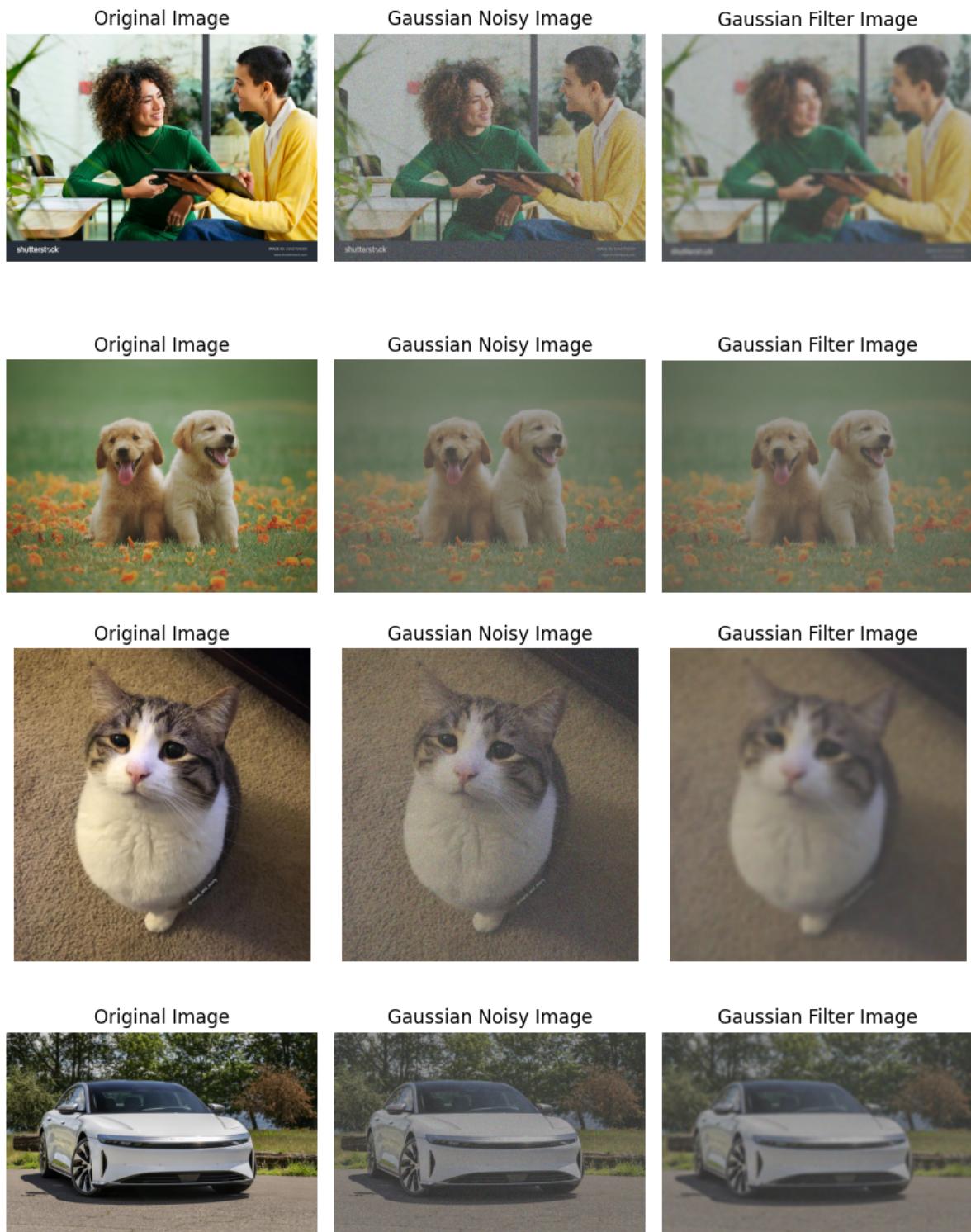
3 Kết quả

3.1 Áp dụng Box Filter



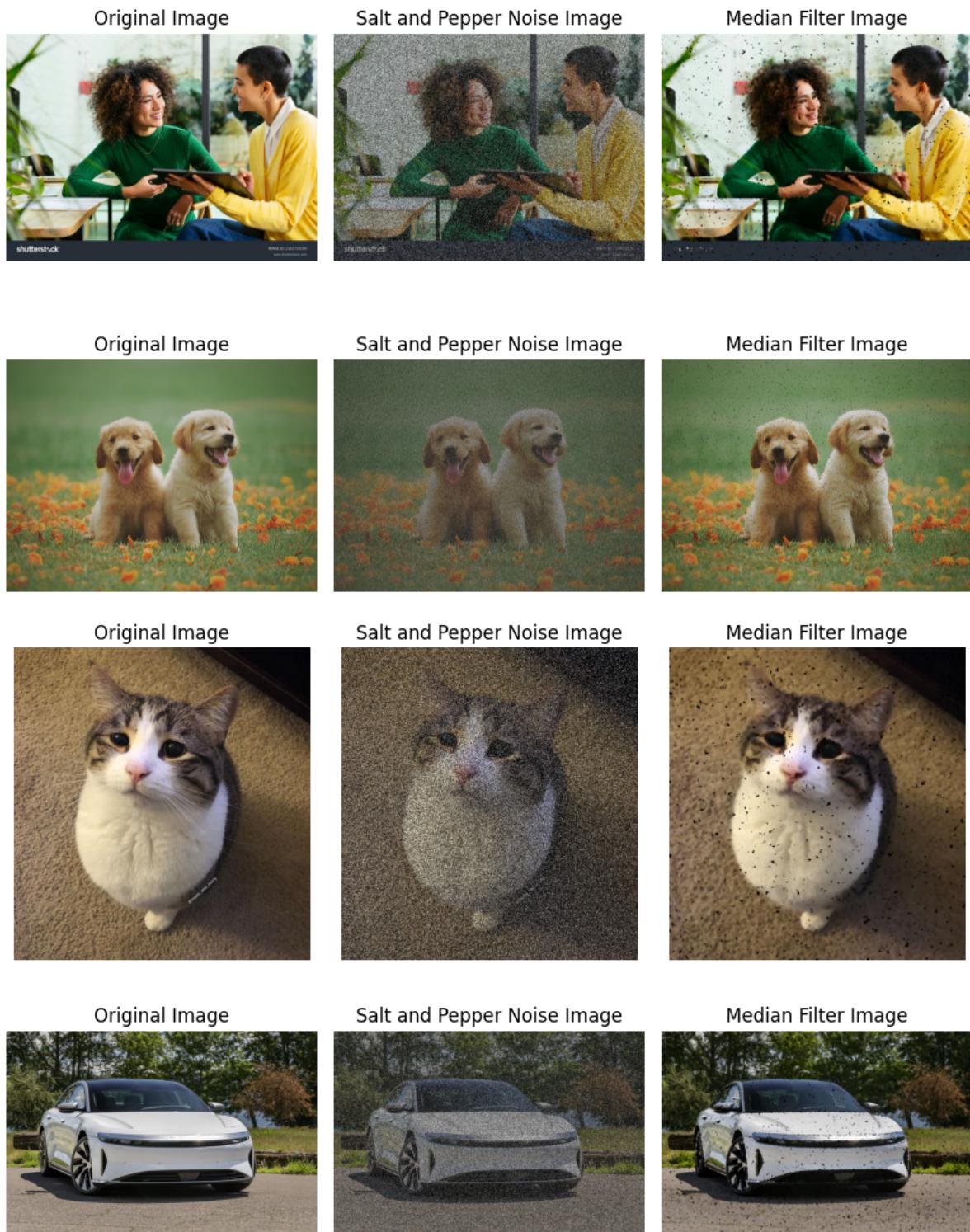
Hình 7: Ảnh sau khi làm mờ bằng Box Filter

3.2 Áp dụng Gaussian Filter



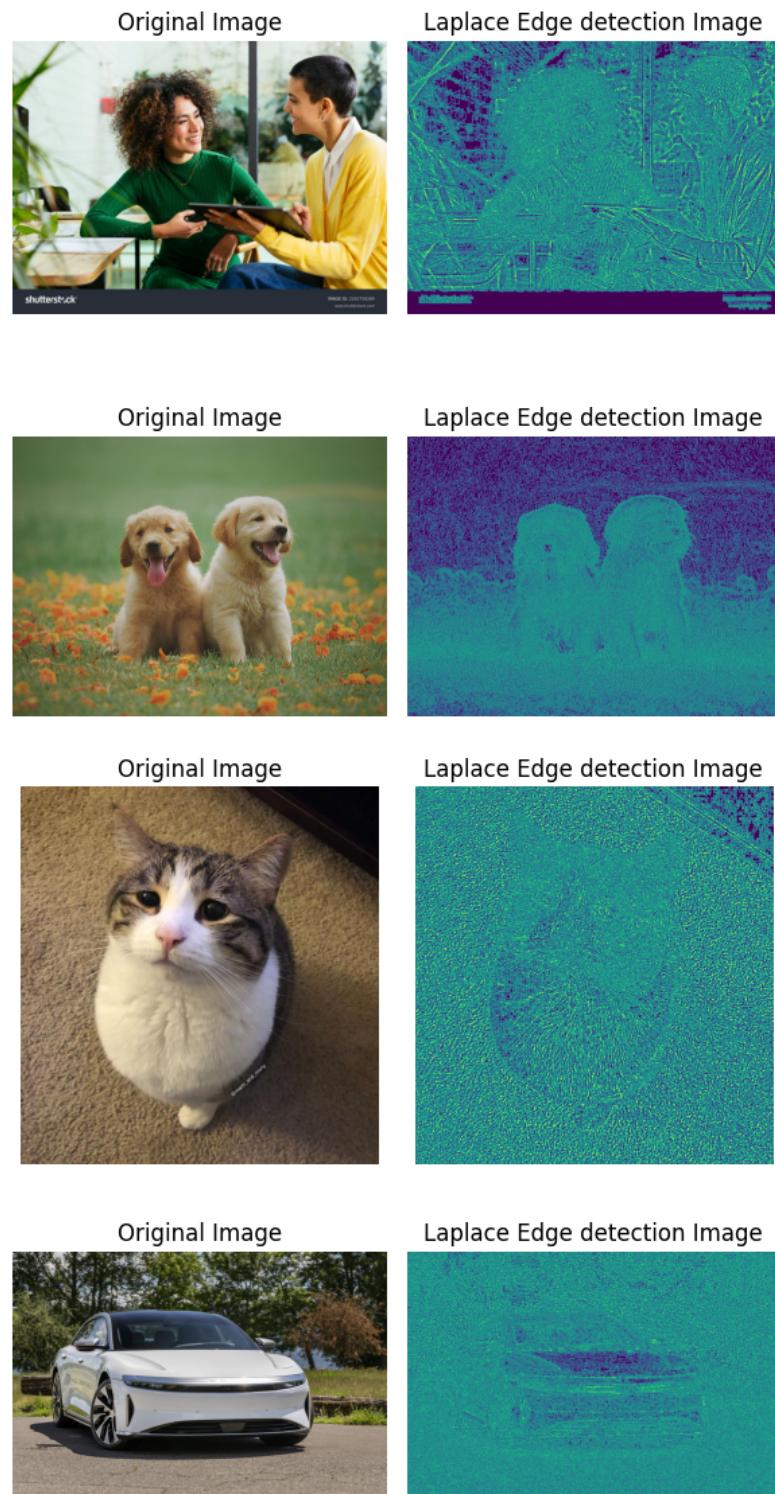
Hình 8: Ảnh sau khi khử nhiễu bằng Gaussian Filter

3.3 Áp dụng Median Filter



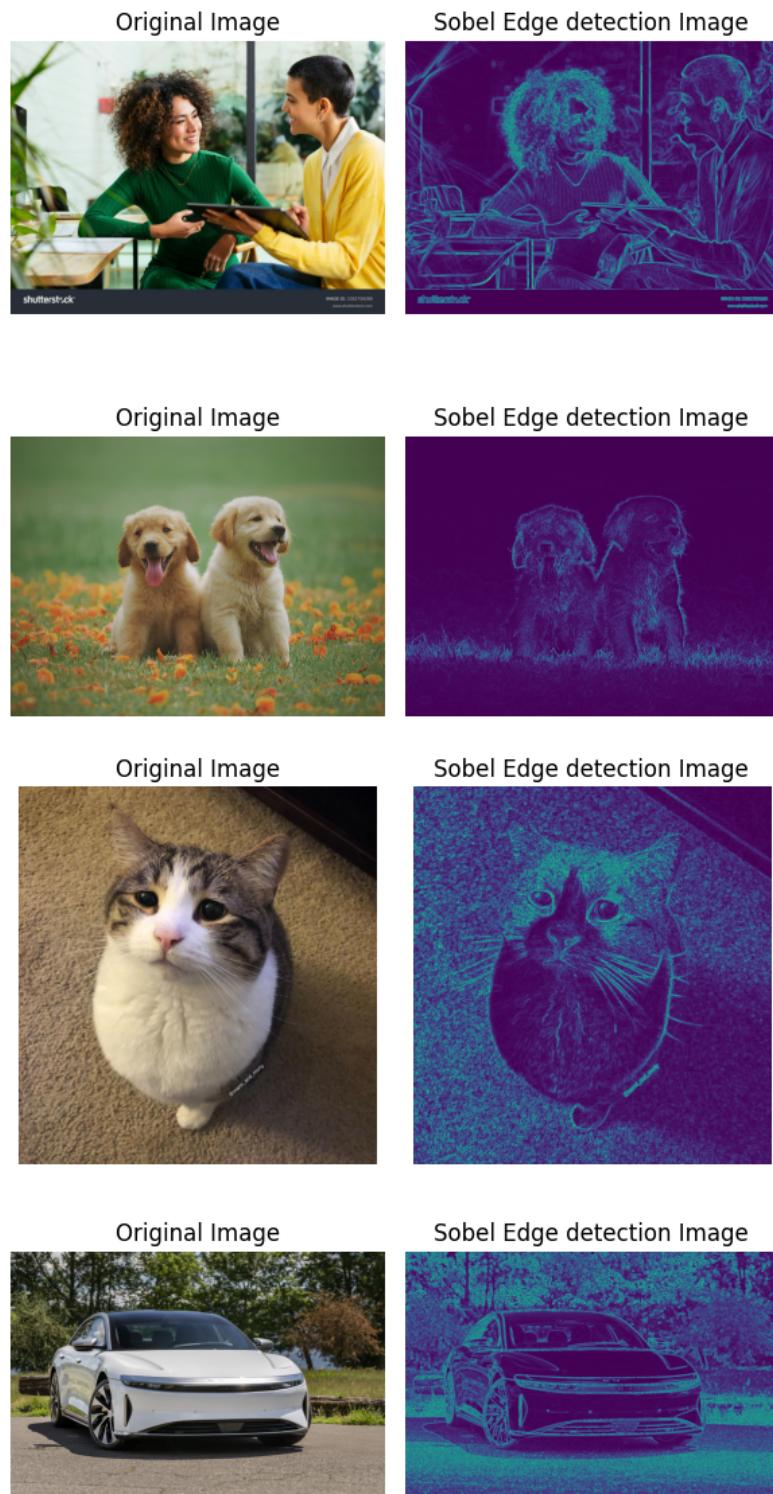
Hình 9: Ảnh sau khi xử lý nhiễu muối tiêu bằng Median Filter

3.4 Áp dụng Laplacian Filter



Hình 10: Các cạnh được phát hiện sau khi áp dụng Laplacian Filter rồi rác

3.5 Áp dụng Sobel Filter



Hình 11: Các cạnh được phát hiện sau khi áp dụng Sobel Filter



4 Nhận xét & Kết luận

Trong xử lý hình ảnh, bộ lọc là một phần quan trọng để thực hiện các phép biến đổi và phát hiện đặc trưng trong hình ảnh. Dưới đây là một số nhận xét và kết luận về các bộ lọc phổ biến như bộ lọc trung bình (box blur), bộ lọc Gaussian, bộ lọc trung vị (median), bộ lọc Laplace rời rạc và bộ lọc Sobel.

4.1 Nhận xét

4.1.1 Bộ Lọc Trung Bình (Box Blur)

Bộ lọc trung bình là một bộ lọc đơn giản được sử dụng để làm mờ hình ảnh bằng cách lấy trung bình của các pixel trong một vùng cụ thể của hình ảnh. Bộ lọc này hiệu quả để loại bỏ nhiễu nhỏ và làm mờ hình ảnh, nhưng nó có thể làm mất thông tin chi tiết.

4.1.2 Bộ Lọc Gaussian

Bộ lọc Gaussian sử dụng hàm Gaussian để làm mờ hình ảnh. Điều này cho phép nó tạo ra một hiệu ứng mờ min hơn so với bộ lọc trung bình và giữ lại nhiều thông tin chi tiết hơn. Bộ lọc Gaussian thường được sử dụng trong nhiều ứng dụng xử lý hình ảnh như làm mịn hình ảnh và giảm nhiễu.

4.1.3 Bộ Lọc Trung Vị (Median)

Bộ lọc trung vị sử dụng giá trị trung vị của các pixel trong một vùng cụ thể để làm mờ hình ảnh. Điều này làm giảm hiệu ứng mờ mịn của hình ảnh và giữ lại các biên rõ ràng hơn so với bộ lọc trung bình và Gaussian. Bộ lọc trung vị đặc biệt hiệu quả trong việc loại bỏ nhiễu đa dạng và giữ lại các chi tiết quan trọng.

4.1.4 Bộ Lọc Laplace Rời Rạc

Bộ lọc Laplace rời rạc được sử dụng để phát hiện biên trong hình ảnh bằng cách nhấn mạnh sự thay đổi đột ngột trong độ sáng. Bộ lọc này thường tạo ra các kết quả với đường biên sắc nét, nhưng cũng dễ bị ảnh hưởng bởi nhiễu.

4.1.5 Bộ Lọc Sobel

Bộ lọc Sobel là một phương pháp phổ biến để phát hiện biên trong hình ảnh bằng cách áp dụng hai kernel Sobel cho gradient theo chiều ngang và dọc. Bộ lọc này tạo ra các kết quả với các đường biên rõ ràng và ít bị ảnh hưởng bởi nhiễu.

4.2 Kết luận

Các bộ lọc trên hình ảnh đều có những ứng dụng và đặc điểm riêng. Bộ lọc trung bình, Gaussian và trung vị thích hợp cho việc làm mờ hình ảnh và giảm nhiễu, mỗi bộ lọc có đặc điểm và hiệu suất khác nhau trong việc làm mịn và giữ lại chi tiết. Trong khi đó, bộ lọc Laplace rời rạc và Sobel thường được sử dụng để phát hiện biên trong hình ảnh, với Sobel thường cho kết quả tốt hơn và ít bị ảnh hưởng bởi nhiễu so với Laplace rời rạc. Việc lựa chọn bộ lọc thích hợp phụ thuộc vào mục đích cụ thể của ứng dụng và yêu cầu về hiệu suất của hệ thống.



Tài liệu

- [1] Wikipedia, *Grayscale*, <https://en.wikipedia.org/wiki/Grayscale>.
- [2] Nttuan8, *Giới thiệu về xử lý ảnh*, <https://nttuan8.com/bai-5-gioi-thieu-ve-xu-ly-anh/>.
- [3] OpenCV, *Does anybody know where I can find the source code for cvtColor() function in OpenCV ?*, <https://forum-of-OpenCV>.
- [4] Davide Scaramuzza, *Lecture 04 Image Filtering*, https://rpg.ifi.uzh.ch/docs/teaching/2017/04_filtering.pdf
- [5] Geeks for Geeks, *Difference between Low pass filter and High pass filter*, <https://www.geeksforgeeks.org/difference-between-low-pass-filter-and-high-pass-filter/>