HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP LỚN

ĐỀ TÀI: Bộ lọc ảnh nâng cao chất lượng ảnh số

Môn học	**:**	**Lập trình mạng**
Giảng viên hướng dẫn	**:**	**Phạm Hoàng Việt**
Nhóm lớp	**:**	**11**
Số nhóm BTL	**:**	**28**
Thành viên trong nhóm	**:**	**Lê Đình Tuấn-B22DCCN756**

Hà Nội – 2025

PHẦN MỞ ĐẦU

1. Giới thiệu đề tài

Trong xử lý ảnh số, chất lượng ảnh đầu vào thường bị suy giảm do nhiễu, mờ, hoặc điều kiện chiếu sáng không tốt. Việc nâng cao chất lượng ảnh và làm nổi bật đường biên là bước tiền xử lý quan trọng cho nhiều bài toán thị giác máy tính như: nhận dạng đối tượng, phát hiện cạnh biên, phân đoạn ảnh, tiền xử lý ảnh y tế, nhận dạng văn bản…

Trong số các kỹ thuật cổ điển, các bộ lọc làm mịn tuyến tính và phi tuyến (Mean, Gaussian, Median) được dùng để giảm nhiễu, trong khi các toán tử biên (Sobel, Prewitt, Laplacian) dùng để phát hiện vùng có biến thiên cường độ mạnh. Đề tài này tập trung xây dựng một công cụ web cho phép:

- Làm mịn ảnh bằng Mean, Gaussian, Median;
- Phát hiện biên bằng Sobel, Prewitt, Laplacian;
- Người dùng có thể tải ảnh hoặc ma trận xám (CSV), điều chỉnh tham số, quan sát trực quan ảnh trước–sau xử lý và tải kết quả.

Toàn bộ các thuật toán được cài đặt từ đầu bằng NumPy, không sử dụng các hàm lọc sẵn có của các thư viện xử lý ảnh như OpenCV, skimage…, tương tự định hướng “from scratch” trong báo cáo tham khảo.

2. Mục tiêu đề tài

Các mục tiêu chính:

1. Nghiên cứu lý thuyết về:
    - Bộ lọc làm mịn: Mean, Gaussian, Median;
    - Toán tử phát hiện biên: Sobel, Prewitt, Laplacian;
    - Ảnh xám, khái niệm padding, phép chập 2D (convolution / correlation).
2. Cài đặt thuật toán từ đầu:
    - Tự xây dựng hàm chập 2D chap_2d cho mọi kernel;
    - Hiện thực Mean, Gaussian, Median, Sobel, Prewitt, Laplacian chỉ dùng NumPy;
    - Tối ưu tốc độ bằng vectorization (as_strided, einsum) để đảm bảo thời gian xử lý nhanh.

3. Xây dựng công cụ web:
    - Giao diện trực quan với hai tab: “Làm mịn ảnh” và “Phát hiện biên”;
    - Cho phép người dùng tải ảnh/CSV, chọn bộ lọc, chỉnh kernel, sigma, padding;
    - Hiển thị song song ảnh gốc và ảnh sau xử lý; cho phép tải kết quả PNG/CSV.
4. Thực nghiệm, so sánh và đánh giá:
    - Thử nghiệm trên nhiều ảnh nhiễu muối tiêu, nhiễu Gaussian, ảnh có biên rõ;
    - So sánh chất lượng làm mịn của Mean/Gaussian/Median;
    - So sánh đáp ứng biên của Sobel, Prewitt, Laplacian;
    - Đánh giá thời gian xử lý, mức độ giữ chi tiết, mức độ làm nổi biên.

3. Phạm vi thực hiện

Để phù hợp thời gian và yêu cầu môn học, đề tài giới hạn:

- Ảnh đầu vào: ảnh xám 8-bit (0–255). Ảnh màu sẽ được chuyển về ảnh xám trước khi xử lý.
- Làm mịn:
    - Mean filter: kernel vuông kích thước lẻ (3,5,7,…);
    - Gaussian filter: kernel vuông, sigma điều chỉnh được;
    - Median filter: kernel vuông kích thước lẻ, cài bằng hai vòng lặp.
- Phát hiện biên:
    - Sobel & Prewitt: dùng hai kernel Gx, Gy; biên = độ lớn gradient;
    - Laplacian: dùng kernel 4-neighborhood, lấy trị tuyệt đối đáp ứng.
- Công nghệ:
    - Ngôn ngữ: Python 3.x;
    - Thư viện: NumPy, Pillow (PIL), Gradio;

- Không sử dụng các hàm lọc ảnh sẵn có của OpenCV, scikit-image,… (chỉ sử dụng NumPy cho toán học, PIL cho I/O ảnh và Gradio cho giao diện).

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1.1. Tổng quan về nâng cao chất lượng ảnh và phát hiện biên

1. Nâng cao chất lượng ảnh (image enhancement)
   Là quá trình biến đổi ảnh đầu vào để kết quả dễ quan sát và phân tích hơn. Một hướng cơ bản là giảm nhiễu và làm mịn, vừa giúp loại bỏ nhiễu cao tần, vừa chuẩn bị ảnh cho các bước sau (phát hiện biên, phân đoạn…).
2. Phát hiện biên (edge detection)
   Biên là những vị trí mà cường độ sáng thay đổi đột ngột. Phát hiện biên giúp:
   - Xác định hình dạng đối tượng;
   - Hỗ trợ trích chọn đặc trưng;
   - Là tiền đề cho phân đoạn ảnh, nhận dạng đối tượng.

Trong pipeline phổ biến, người ta thường làm mịn trước rồi mới phát hiện biên để giảm ảnh hưởng của nhiễu lên toán tử đạo hàm.

1.2. Ảnh xám, ma trận ảnh và phép chập 2D

- Ảnh xám 8-bit được biểu diễn bởi ma trận $I \in \mathbb{R}^{M \times N}$, giá trị mỗi pixel $I(x, y)$nằm trong [0, 255].
- Kernel lọc là ma trận nhỏ $K \in \mathbb{R}^{k \times k}$(thường klẻ).
- Chập 2D (convolution/correlation): tại mỗi vị trí (x,y), giá trị mới được tính bằng:

$$I'(x,y) = \sum_{i=-r}^{r} \sum_{j=-r}^{r} K(i,j) \cdot I(x+i, y+j)$$

với $k = 2r + 1$.

Do biên ảnh không đủ lân cận, ta phải padding ảnh bằng cách thêm lề:

- Zero padding: thêm đường viền giá trị 0;
- Replicate: lặp lại hàng/cột biên;

- Reflect: phản xạ qua biên như gương.

Trong code, phép chập được cài đặt trong hàm chap_2d, dùng numpy.pad để thêm lề và einsum để tính nhân–cộng cho mọi cửa sổ cùng lúc.

1.3. Bộ lọc làm mịn

1.3.1. Bộ lọc trung bình (Mean filter)

- Kernel:

$$K_{\text{mean}} = \frac{1}{k^2}\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}_{k\times k}$$

- Ý tưởng: mỗi pixel mới là trung bình cộng của toàn bộ pixel trong cửa sổ k×k.
- Tác dụng:
  - Giảm nhiễu ngẫu nhiên,
  - Nhưng dễ làm mờ biên và mất chi tiết nhỏ.

1.3.2. Bộ lọc Gaussian

- Kernel Gaussian 2D:

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

- Trong cài đặt, ta lấy lưới toạ độ (x, y)xung quanh tâm, áp dụng công thức trên và chuẩn hoá sao cho tổng kernel = 1.
- Tham số:
  - Kích thước kernel k(lẻ),
  - Độ lệch chuẩn σ.

- So với mean, Gaussian ưu tiên trọng số lớn hơn ở gần tâm, nên làm mịn "mượt" và giữ biên tốt hơn.

1.3.3. Bộ lọc Median

- Với cửa sổ $k \times k$, ta sắp xếp các giá trị trong cửa sổ rồi lấy trung vị (median).
- Đây là bộ lọc phi tuyến, rất hiệu quả cho nhiễu muối tiêu:
    - Những điểm nhiễu đơn lẻ (rất sáng hoặc rất tối) sẽ bị "đa số" các pixel xung quanh loại bỏ,
    - Biên vẫn được bảo toàn tốt hơn so với mean/gauss.

1.4. Toán tử phát hiện biên

1.4.1. Sobel và Prewitt – gradient bậc nhất

- Cả Sobel và Prewitt đều dùng hai kernel để xấp xỉ đạo hàm theo phương x, y:

Ví dụ Sobel:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Prewitt có dạng tương tự nhưng trọng số đều (–1,0,1).

- Áp dụng:
    - Chập ảnh với $G_x \rightarrow F_x$,
    - Chập ảnh với $G_y \rightarrow F_y$,
    - Biên: $M = \sqrt{F_x^2 + F_y^2}$.
- Các điểm có giá trị Mlớn là nơi cường độ thay đổi mạnh → biên.

1.4.2. Laplacian – đạo hàm bậc hai

- Kernel Laplacian dạng 4-neighborhood:

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- Áp dụng chập:

$$R = | I * L |$$

- Laplacian nhấn mạnh các vùng có biến đổi cường độ dạng "cực trị" (đỉnh hoặc đáy), cho biên mỏng và nhạy với nhiễu. Thường nên làm mịn bằng Gaussian trước rồi mới áp Laplacian.

CHƯƠNG 2: THIẾT KẾ VÀ CÀI ĐẶT

2.1. Kiến trúc hệ thống

Hệ thống được thiết kế theo kiểu 3 lớp giống phong cách báo cáo tham khảo:

- Lớp giao diện (Presentation layer)
    - File: ung_dung.py
    - Công nghệ: Gradio
    - Chức năng:
        - Cho phép tải ảnh/CSV;
        - Chọn bộ lọc và tham số;
        - Hiển thị ảnh gốc và ảnh sau xử lý;
        - Cho phép tải ảnh/CSV kết quả.
- Lớp xử lý (Business logic layer)
    - Thư mục bo_loc:
        - cong_cu_chap.py: cài đặt padding & chập 2D;
        - lam_min.py: các bộ lọc làm mịn;
        - bien.py: các toán tử phát hiện biên.
    - Chịu trách nhiệm thực hiện toàn bộ thuật toán.
- Lớp dữ liệu (Data / I/O layer)
    - Thư mục tien_ich:

        - io_anh.py: đọc ảnh/CSV, chuyển về ảnh xám, chuẩn hoá; lưu kết quả PNG/CSV.
    - Sử dụng Pillow để đọc ảnh, NumPy để biểu diễn ma trận.

2.2. Mô tả các module chính

2.2.1. Module tien_ich/io_anh.py

- doc_anh_hoac_csv(tap_tin)
    - Nếu là ảnh: đọc bằng PIL, chuyển sang ảnh xám (convert("L")), thu nhỏ nếu cạnh > 1024, trả về mảng float32 0–255.
    - Nếu là CSV: dùng np.loadtxt, nếu max ≤ 1 thì nhân 255, clip [0,255].
- chuan_hoa_uint8(anh)
    - Thay NaN/Inf = 0, clip [0,255], ép kiểu uint8.
- luu_png(anh) / luu_csv(anh)
    - Lưu tạm ảnh/ma trận ra file để Gradio dùng làm output tải về.

2.2.2. Module bo_loc/cong_cu_chap.py

- them_le(anh, k, kieu)
    - Thêm lề cho ảnh theo ba chế độ zero, replicate, reflect dùng np.pad.
- chap_2d(anh, nhan, kieu_padding)
    - Hàm cốt lõi thực hiện chập 2D:
        - Pad ảnh;
        - Dùng as_strided tạo tensor cửa sổ trượt;
        - Dùng np.einsum("ij,xyij->xy", ...) để tính tổng nhân kernel với từng cửa sổ;
        - Trả về ảnh kết quả (float).

2.2.3. Module bo_loc/lam_min.py

- nhan_trung_binh(k) – tạo kernel mean.
- nhan_gauss(k, sigma) – tạo kernel Gaussian 2D.

- loc_trung_binh(anh, k, padding) – chập ảnh với kernel mean.
- loc_gauss(anh, k, sigma, padding) – chập ảnh với kernel Gaussian.
- loc_median(anh, k, padding) – padding rồi duyệt for i,j, lấy np.median trong cửa sổ.

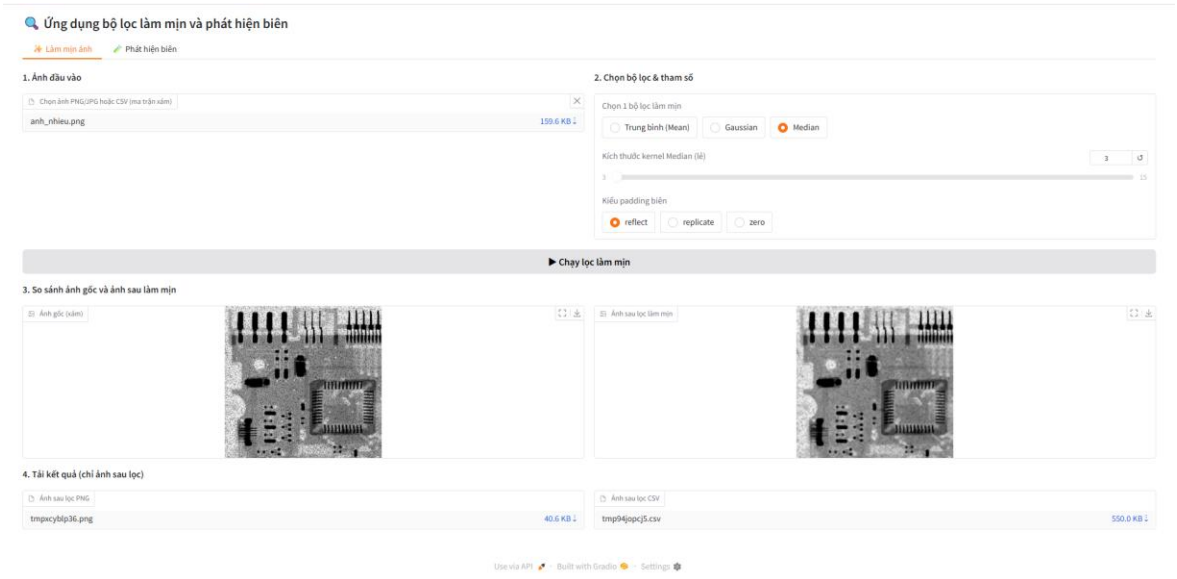
2.2.4. Module bo_loc/bien.py

- nhan_sobel(), nhan_prewitt(), nhan_laplacian() – trả về các kernel tương ứng.
- bien_do_gradient(anh, gx, gy, padding, chuan_hoa) – tính biên Sobel/Prewitt.
- dap_ung_laplacian(anh, padding, chuan_hoa) – biên Laplacian.

2.3. Thiết kế giao diện người dùng

Công cụ được xây dựng bằng Gradio với hai tab:

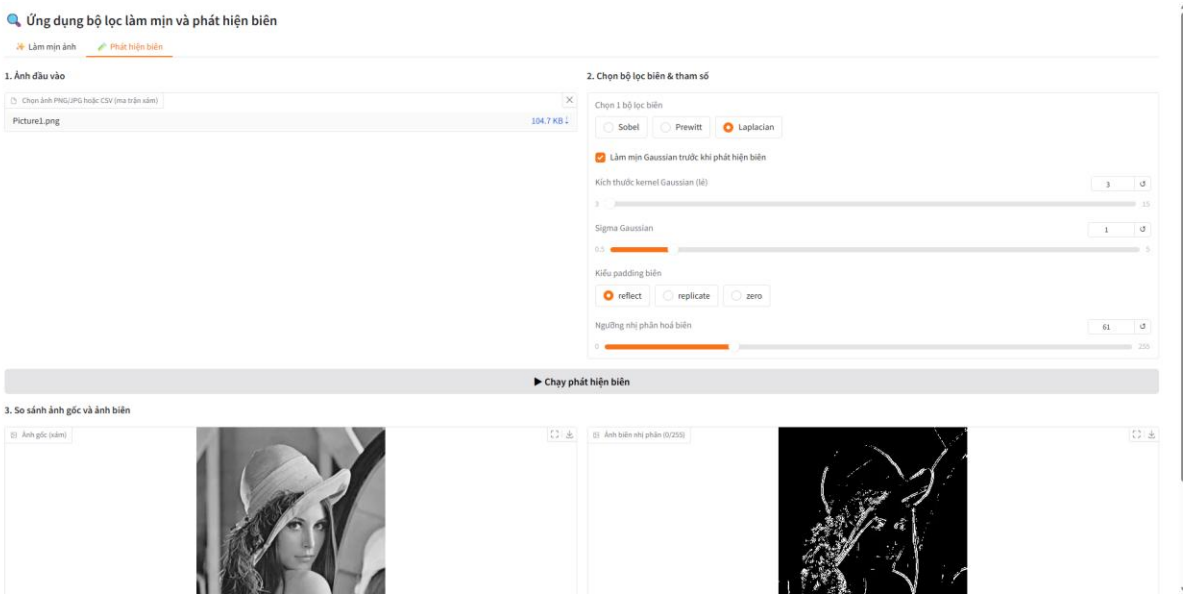
1. Tab “Làm mịn ảnh”



   - Khu vực 1: chọn ảnh đầu vào (PNG/JPG hoặc CSV).
   - Khu vực 2: chọn bộ lọc:
     - Radio: Mean / Gaussian / Median;
     - Các slider tham số chỉ hiển thị khi cần:
       - Mean: kích thước kernel;
       - Gaussian: kích thước + sigma;

        - Median: kích thước kernel Median;
    - Radio kiểu padding: reflect / replicate / zero.
- Nút “Chạy lọc làm mịn”.
- Khung hiển thị:
    - Ảnh gốc;
    - Ảnh sau làm mịn.
- Khu vực tải kết quả:
    - Ảnh sau lọc PNG;
    - Ma trận sau lọc CSV.

2. Tab “Phát hiện biên”



- Chọn ảnh đầu vào.
- Chọn toán tử biên: Sobel / Prewitt / Laplacian.
- Tuỳ chọn: “Làm mịn Gaussian trước khi phát hiện biên” (kernel & sigma).
- Nút “Chạy phát hiện biên”.
- Khung hiển thị:
    - Ảnh gốc;

        - Ảnh biên thu được.
    - Khu tải kết quả: biên PNG / CSV.

CHƯƠNG 3: KẾT QUẢ THỰC NGHIỆM

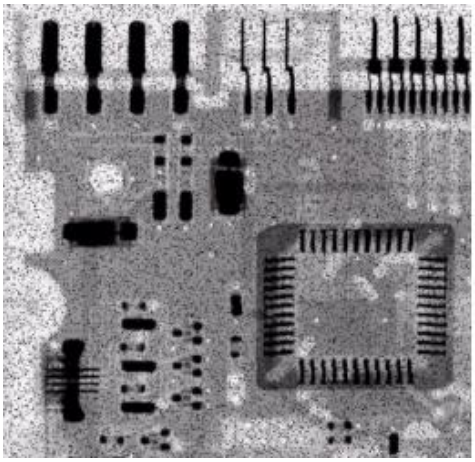
3.1. Môi trường thử nghiệm

- Máy tính: ghi cấu hình CPU, RAM của bạn.
- Hệ điều hành: Windows 10/11.
- Python 3.x; thư viện NumPy, Pillow, Gradio các phiên bản tương ứng.
- Ảnh test:
    - Ảnh mạch in có nhiễu muối tiêu mạnh;
    - Ảnh phong cảnh có biên rõ (toà nhà, bờ đường…);
    - Một số ảnh kích thước trung bình (khoảng 512×512).

3.2. Kết quả làm mịn

3.2.1. So sánh Mean – Gaussian – Median trên ảnh có nhiễu muối tiêu

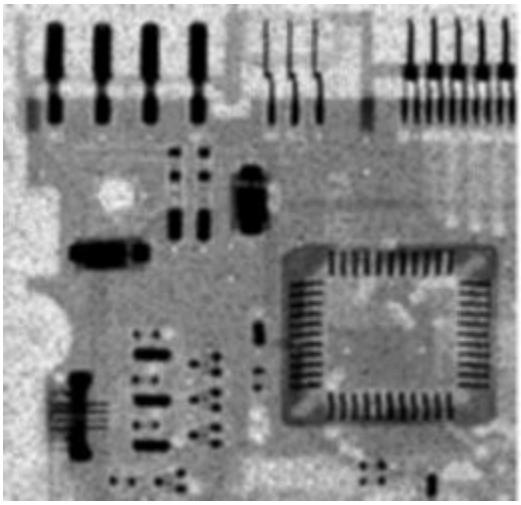
- Thiết lập:
    - Ảnh mạch in bị nhiễu muối tiêu .



(Hình 3.1 – ảnh gốc).

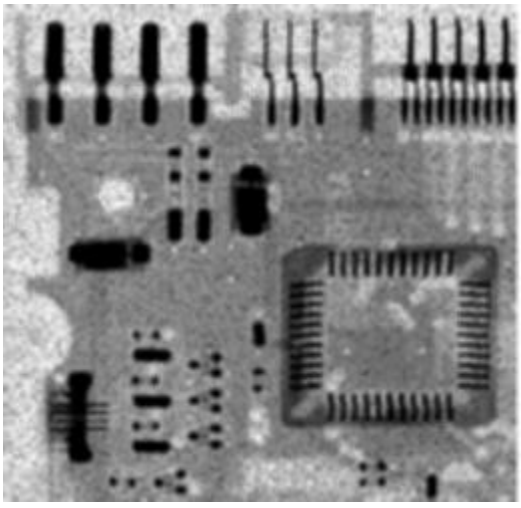
    - Mean: kernel 5×5; Gaussian: kernel 5×5, sigma=1.0; Median: kernel 5×5, padding reflect.

- Kết quả:
    - Hình 3.2: ảnh sau lọc Mean.



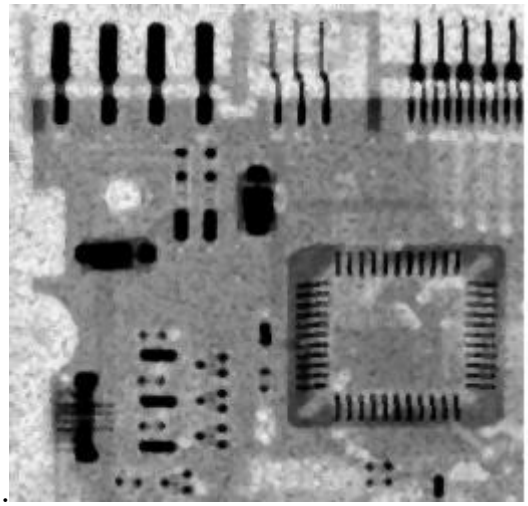
Hình 3.2: ảnh sau lọc Mean.

    - Hình 3.3: ảnh sau lọc Gaussian.



Hình 3.3: ảnh sau lọc Gaussian.

    - Hình 3.4: ảnh sau lọc Median



Hình 3.4: ảnh sau lọc Median

- Nhận xét gợi ý:
    - Mean làm giảm nhiễu nhưng biên linh kiện mờ đi rõ rệt.
    - Gaussian mịn hơn Mean, các vùng đồng nhất trông mượt, nhưng chân linh kiện vẫn bị "béo".
    - Median loại bỏ nhiễu muối tiêu rất hiệu quả, biên chân linh kiện giữ được tốt hơn → thích hợp cho nhiễu muối tiêu.

Thuật toán	**Nguyên lý chính**	**Ưu điểm**	**Nhược điểm**	**Khi nên dùng**
Mean (trung bình)	Mỗi pixel mới = trung bình cộng của các pixel trong cửa sổ k×k	Cài đặt đơn giản, tính nhanh, giảm nhiễu ngẫu nhiên	Làm mờ biên mạnh, mất chi tiết nhỏ, dễ "bệt"	Khi cần demo khái niệm làm mịn cơ bản, ảnh không cần giữ chi tiết cao
Gaussian	Trọng số lớn ở gần tâm, nhỏ dần ra xa theo phân bố Gaussian	Làm mịn "mượt", ít tạo khối vuông, giữ biên tốt hơn Mean	Vẫn làm mờ biên nếu kernel / σ quá lớn, không khử nhiễu muối tiêu tốt bằng Median	Khi cần tiền xử lý trước phát hiện biên / phân đoạn, cần làm mịn

Thuật toán	Nguyên lý chính	Ưu điểm	Nhược điểm	Khi nên dùng
				nhưng vẫn giữ tương đối chi tiết
Median	Mỗi pixel mới = **trung vị** của các giá trị trong cửa sổ k×k	Khử nhiễu muối tiêu rất tốt, giữ biên tốt hơn Mean/Gauss, ít làm mờ cạnh sắc	Tính toán nặng (vòng for), kernel lớn làm ảnh "phẳng", chậm với ảnh to	Khi ảnh bị nhiễu muối tiêu (lốm đốm đen trắng), muốn giữ biên rõ, chấp nhận tốn thời gian hơn

3.2.2. Ảnh hưởng của kích thước kernel

- Thử Median với k = 3, 5, 7, 9.
- Nhận xét:
    - k=3: nhiễu còn khá nhiều nhưng chi tiết giữ tốt;
    - k=5,7: cân bằng giữa khử nhiễu và giữ chi tiết;
    - k=9: nhiễu gần như mất hết nhưng ảnh trông "phẳng", nhiều chi tiết nhỏ bị xoá.

kernel càng lớn → càng mịn nhưng càng mờ.

3.3. Kết quả phát hiện biên

3.3.1. So sánh Sobel – Prewitt – Laplacian

- Ảnh gốc: ảnh vật thể có biên rõ

- Tham số:
    - Padding reflect;
    - Có/không làm mịn Gaussian trước.
- Kết quả:
    - Sobel: biên dày, rõ, ít nhiễu;

    - Prewitt: tương tự Sobel nhưng nhạy nhiễu hơn một chút;

- Laplacian: cho biên mỏng, nhưng nếu không làm mịn trước thì xuất hiện nhiều biên giả do nhiễu.

Thuật toán	**Dựa trên gradient gì**	**Ưu điểm**	**Nhược điểm**	**Khi nên dùng**
Prewitt	Xấp xỉ $\partial I/\partial x$, $\partial I/\partial y$ bằng kernel 3×3 với trọng số đều (–1, 0, 1)	Đơn giản, dễ hiểu, dùng tốt để minh hoạ khái	Nhạy với nhiễu hơn Sobel, biên kém "mượt" hơn một chút	Khi cần minh hoạ cơ bản về đạo hàm xấp xỉ, bài lý thuyết/giáo trình

Thuật toán	Dựa trên gradient gì	Ưu điểm	Nhược điểm	Khi nên dùng
		niệm gradient		
Sobel	Gradient dùng kernel 3×3, hàng giữa trọng số lớn hơn (–2, 0, 2)	Mượt hơn Prewitt, ổn định hơn với nhiễu, biên rõ và tương đối dày	Vẫn nhạy với nhiễu nếu không làm mịn trước, không cho biên mỏng như Laplacian	Khi cần bản đồ biên dùng trong nhận dạng/segmentation, thường kết hợp với Gaussian trước đó
Laplacian	Đạo hàm bậc 2 (tính tổng chênh lệch với hàng xóm), kernel như [[0,1,0],[1,-4,1],[0,1,0]]	Phát hiện biên theo mọi hướng, biên mỏng, nhạy với vùng chuyển mạnh	Rất nhạy với nhiễu, dễ sinh biên giả, thường phải làm mịn trước; không cho hướng biên	Khi cần biên mỏng, nhấn mạnh đường bao, và đã có bước làm mịn tốt (Gaussian + Laplacian)

3.3.2. Ảnh hưởng của bước làm mịn trước khi phát hiện biên

- Chạy Sobel/Prewitt/Laplacian:
    - Trực tiếp trên ảnh gốc;
    - Sau khi làm mịn bằng Gaussian 5×5, sigma=1.0.
- Nhận xét:
    - Khi không làm mịn, bản đồ biên có nhiều điểm lốm đốm (nhiễu).
    - Khi làm mịn Gaussian trước, biên mượt hơn, ít biên giả; tuy nhiên một số chi tiết nhỏ có thể bị mất.

3.4. Đánh giá công cụ

Liên hệ lại với rubric:

1. Mức độ hoàn thiện & chức năng

- Tool thực hiện đầy đủ các chức năng làm mịn (Mean, Gaussian, Median) và phát hiện biên (Sobel, Prewitt, Laplacian).
- Cho phép tải/lưu ảnh PNG & CSV.
- Thời gian xử lý nhanh (ảnh ~512×512 xử lý trong vài chục ms đến vài trăm ms tuỳ bộ lọc Median/size).

2. Tính sáng tạo & ứng dụng
    - Giao diện web chia tab rõ ràng;
    - Cho phép người dùng điều chỉnh tham số kernel, sigma, padding để quan sát ảnh hưởng lên kết quả;
    - Có thể dùng như công cụ minh hoạ trong học phần Xử lý ảnh số.
3. Kỹ thuật lập trình & giao diện
    - Kiến trúc module rõ ràng: tien_ich, bo_loc, ung_dung;
    - Các hàm xử lý thuần NumPy, dễ tái sử dụng;
    - Giao diện Gradio đơn giản, trực quan, hiển thị song song ảnh gốc và ảnh sau xử lý.

CHƯƠNG 4: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1. Kết luận

Đề tài đã:

- Nghiên cứu và cài đặt thành công các bộ lọc làm mịn (Mean, Gaussian, Median) và toán tử phát hiện biên (Sobel, Prewitt, Laplacian) từ đầu bằng NumPy;
- Xây dựng công cụ web cho phép người dùng dễ dàng thử nghiệm các bộ lọc, điều chỉnh tham số và quan sát trực quan ảnh trước–sau;
- Thực nghiệm trên nhiều ảnh khác nhau, rút ra:
    - Median phù hợp với nhiễu muối tiêu, Gaussian cho kết quả làm mịn tự nhiên;
    - Sobel/Prewitt hiệu quả cho phát hiện biên hướng, Laplacian nhạy với nhiễu nhưng cho biên mỏng;

- Làm mịn trước khi phát hiện biên giúp giảm biên giả nhưng có thể làm mất chi tiết nhỏ.

4.2. Hạn chế

- Chỉ xử lý ảnh xám, chưa hỗ trợ trực tiếp ảnh màu;
- Chưa có đánh giá định lượng (PSNR, SSIM) mà chủ yếu đánh giá bằng mắt;
- Lọc Median cài đặt bằng vòng lặp nên với kernel rất lớn sẽ chậm hơn so với các kỹ thuật tối ưu hơn.

4.3. Hướng phát triển

- Mở rộng sang ảnh màu (lọc từng kênh hoặc trong không gian HSV);
- Cài đặt thêm các bộ lọc nâng cao: Bilateral, Non-local Means;
- Thêm thuật toán biên Canny;
- Thêm chức năng xử lý batch cho nhiều ảnh và xuất báo cáo tự động;
- Cải thiện giao diện (thêm zoom, pan, so sánh nhiều bộ lọc trên cùng một ảnh).