TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

----- ☙ 🕮 ❧ -----

TIỂU LUẬN
PHÂN TÍCH XỬ LÝ ẢNH

ĐỀ TÀI: "IMAGE RESTORATION AND RECONSTRUCTION."

Giảng viên hướng dẫn: **PGS.TS.Phạm Thế Bảo**

Nhóm : 3
Thành viên:
Hồ Văn Quyến – 3122410352
Trần Thanh Phương – 3122410333
Huỳnh Quang Quân – 3122410341
Võ Anh Tuấn – 3122410453

Năm học: 2024 - 2025
Thành phố Hồ Chí Minh, tháng 4, năm 2025

NHẬN XÉT, ĐÁNH GIÁ CỦA GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

MỤC LỤC

MỤC LỤC 3
GIỚI THIỆU 1
I. MÔ HÌNH QUÁ TRÌNH SUY GIẢM/ KHÔI PHỤC HÌNH ẢNH 2
II. MÔ HÌNH NHIỄU 3
1. Nhiễu Gaussian 3
2.Nhiễu Rayleigh 4
3.Nhiễu Erlang (gamma) 5
4.Nhiễu phân phối mũ (Exponential) 6
5.Nhiễu đồng đều (Uniform) 7
6.Nhiễu Salt and Pepper (Nhiễu Impulse) 7
7.Nhiễu tuần hoàn (Periodic) 8
III. PHỤC HỒI KHI CHỈ CÓ NHIỄU-LỌC KHÔNG GIAN 9
1.Bộ lọc trung bình (Mean Filters) 9
a) Bộ lọc trung bình số học *(Arithmetic Mean Filter):* 9
b) Bộ lọc trung bình hình học *(Geometric Mean Filter):* 10
c) Bộ lọc trung bình hài hòa *(Harmonic Mean Filter):* 10
d) Bộ lọc trung bình sóng hài Contra *(Contra-harmonic Mean Filter):* 10
2.Bộ lọc thống kê thứ tự *(Order-Statistic Filters) :* 11
a) Bộ lọc trung vị *(Median Filter):* 11
b) Bộ lọc tối đa và tối thiểu *(Max and Min Filters):* 12
c) Bộ lọc điểm giữa *(Midpoint Filter):* 12
d) Bộ lọc trung bình cắt alpha *(Alpha-trimmed Mean Filter):* 13
IV. BỘ LỌC THÍCH ỨNG (Adaptive Filters) 13
1.Bộ lọc giảm nhiễu cục bộ, thích nghi 13
2.Bộ lọc trung vị thích nghi *(Adaptive Median Filter) :* 14
V. GIẢM NHIỄU ĐỊNH KỲ BẰNG CÁCH LỌC TRONG MIỀN TẦN SỐ 15
1.Bộ lọc Loại bỏ Dải (Band Reject Filter): 15
2.Bộ lọc băng thông (Band Pass Filter): 16
3.Bộ lọc Rãnh (Notch Filters): 17
VI. XUỐNG CẤP TUYẾN TÍNH, VỊ TRÍ BẤT BIẾN 18
VII. ƯỚC TÍNH HÀM SUY THOÁI 20
1.Ước tính bằng quan sát hình ảnh: 20
2.Ước tính bằng thử nghiệm: 20
3.Ước tính bằng mô hình hóa: 21
VIII. LỌC NGHỊCH ĐẢO (Inverse Filtering) 23
IX. LỌC SAI SỐ BÌNH PHƯƠNG TỐI THIỂU (Wiener Filtering) 23
X. LỌC BÌNH PHƯƠNG BÉ NHẤT CÓ RÀNG BUỘC (Constrained Least Square Filtering) 25
XI. BỘ LỌC TRUNG BÌNH HÌNH HỌC (Geometric Mean Filter) 28
TÀI LIỆU THAM KHẢO 29
KẾT LUẬN 30

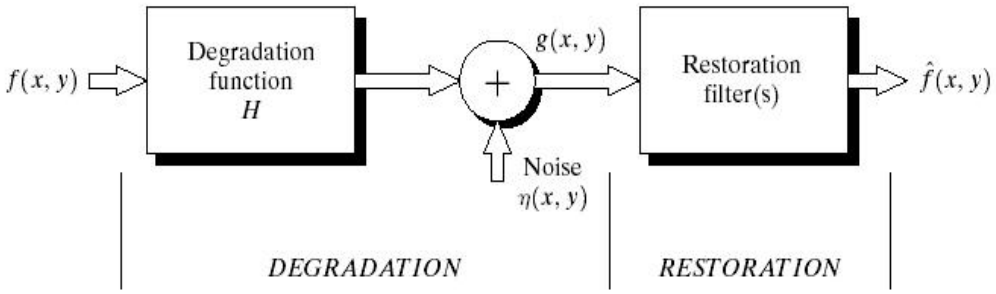
IMAGE RESTORATION AND RECONSTRUCTION

GIỚI THIỆU

Phục hồi hình ảnh là một lĩnh vực quan trọng trong xử lý ảnh kỹ thuật số, nhằm khôi phục chất lượng của các hình ảnh bị suy giảm do các yếu tố như nhiễu, mờ, hoặc lỗi thiết bị. Nhóm chúng tôi đã nghiên cứu các kỹ thuật phục hồi để tìm hiểu cách tái tạo hình ảnh gốc một cách chính xác nhất. Không giống như các phương pháp nâng cao hình ảnh mang tính chủ quan, phục hồi hình ảnh sử dụng các mô hình toán học khách quan, dựa trên thông tin về quá trình suy giảm để xây dựng lại hình ảnh. Trong bài tiểu luận này, chúng tôi sẽ trình bày các mô hình suy giảm, các loại nhiễu phổ biến, và các phương pháp lọc để khôi phục hình ảnh.

I. MÔ HÌNH QUÁ TRÌNH SUY GIẢM/ KHÔI PHỤC HÌNH ẢNH

Quá trình suy giảm hình ảnh xảy ra khi chất lượng hình ảnh gốc bị ảnh hưởng bởi các yếu tố như mờ hoặc nhiễu, dẫn đến hình ảnh đầu ra kém chất lượng. Mô hình này được thể hiện trong hình dưới đây:



Hình 1. Mô hình quá trình suy giảm / khôi phục hình ảnh

Giả sử f(x,y) là hình ảnh đầu vào và g(x, y) là hình ảnh bị suy giảm với một số kiến thức về hàm suy giảm H và một số kiến thức về thành phần nhiễu cộng thêm η(x, y). Mục tiêu của phục hồi là thu được một ước lượng $\hat{f}$ (x, y) của hình ảnh gốc. Nếu H là một quá trình tuyến tính, không thay đổi theo vị trí, thì hình ảnh bị suy giảm trong miền không gian được cho bởi:

$$g(x, y) = f(x, y) * h(x, y) + \eta(x, y)$$

Trong đó h(x, y) là biểu diễn không gian của hàm suy giảm. Hình ảnh bị suy giảm trong miền tần số được biểu diễn dưới dạng:

$$G(u, v) = F(u, v)\, H(u, v) + N(u, v)$$

Các ký hiệu bằng chữ cái in hoa là Biến đổi Fourier của các thành phần tương ứng trong miền không gian.

II. MÔ HÌNH NHIỄU

Nhiễu trong hình ảnh kỹ thuật số là nguyên nhân chính gây suy giảm chất lượng, thường xuất hiện trong quá trình thu nhận hoặc truyền tải hình ảnh.

- **Thu nhận hình ảnh**: Hiệu suất của cảm biến bị ảnh hưởng bởi các yếu tố như ánh sáng yếu, nhiệt độ, hoặc chất lượng thiết bị.
- **Truyền tải hình ảnh**: Nhiễu được thêm vào do lỗi kênh truyền tín hiệu.

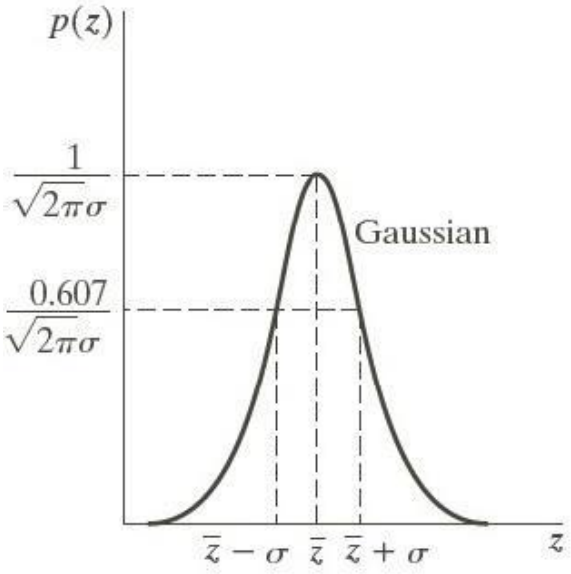
Nhiễu được xem là các biến ngẫu nhiên, được mô tả bằng hàm mật độ xác suất (PDF). Dưới đây là các loại nhiễu chính mà nhóm đã nghiên cứu:

1. Nhiễu Gaussian

Nhiễu Gaussian, hay nhiễu chuẩn, thường xuất hiện do sai số ngẫu nhiên của cảm biến. Hàm PDF của nó là:

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\bar{z})^2/2\sigma^2}$$

Trong đó z biểu thị cường độ, $\bar{z}$ là giá trị trung bình và σ là độ lệch chuẩn, còn bình phương của nó $\text{square}(\sigma^2)$ được gọi là phương sai của z. Khoảng 70% giá trị của nhiễu Gaussian nằm trong khoảng $[(\bar{z}-\sigma), (\bar{z}+\sigma)]$ và 95% nằm trong khoảng $[(\bar{z}-\sigma), (\bar{z}+\sigma)]$.

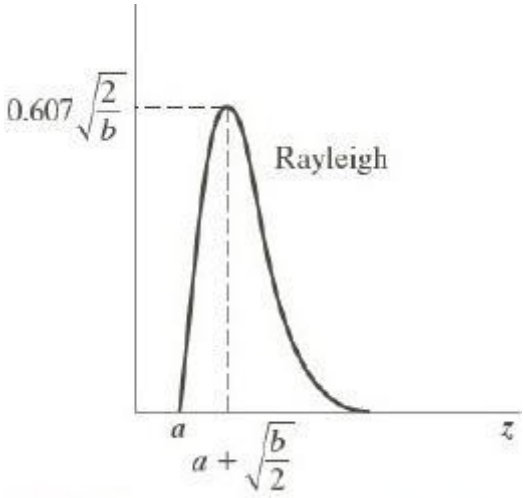


2.Nhiễu Rayleigh

Nhiễu Rayleigh thường gặp trong các ứng dụng như radar hoặc đo lường khoảng cách. Hàm PDF của nó là:

$$p(z) = \begin{cases} \frac{2}{b}(z-a)e^{(z-a)^2/b} & \text{if } z \geq a \\ 0 & \text{if } z < a \end{cases}$$

$$Mean : \bar{z} = a + \sqrt{\pi b/4} \qquad Varience : \sigma^2 = \frac{b(4-\pi)}{4}$$



Ứng dụng:

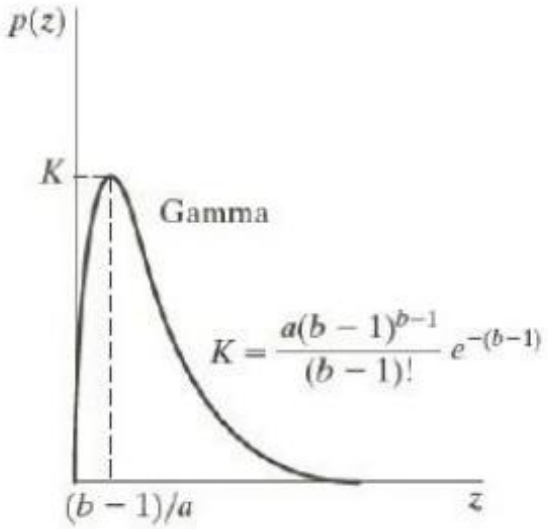
- Nó được sử dụng để mô tả hiện tượng nhiễu trong hình ảnh phạm vi.
- Nó mô tả lỗi trong dụng cụ đo.
- Nó mô tả tiếng ồn bị ảnh hưởng trong radar.
- Nó xác định nhiễu xảy ra khi tín hiệu được truyền qua bộ lọc thông dải.

3.Nhiễu Erlang (gamma)

Nhiễu Erlang phù hợp để mô tả các hệ thống có nhiều nguồn nhiễu chồng lấn. Hàm PDF của nó là:

$$p(z) = \begin{cases} \frac{a^b z^{b-1}}{(b-1)!} e^{-az} & \text{for } z \geq 0 \quad a > 0 \\ 0 & \text{for } z < 0 \quad b \text{ is a positive integer} \end{cases}$$

$$Mean : \bar{z} = \frac{b}{a} \qquad Varience : \sigma^2 = \frac{b}{a^2}$$



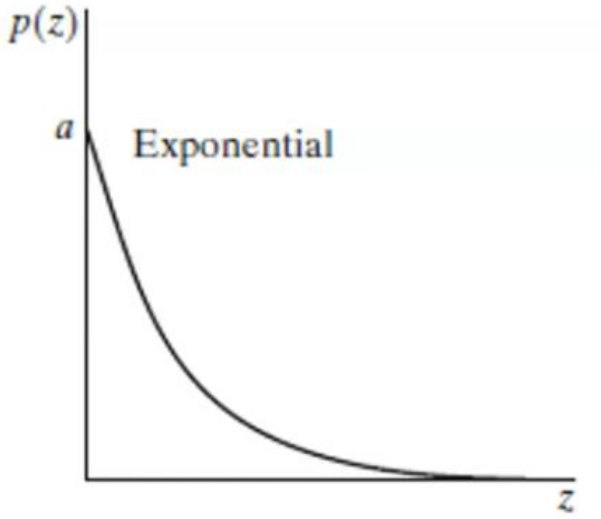
4.Nhiễu phân phối mũ (Exponential)

Nhiễu Exponential mô tả sự suy giảm tín hiệu ngẫu nhiên. Hàm mật độ xác suất của nhiễu Exponential được cho bởi:

$$p(z) = \begin{cases} ae^{-az} & \text{for } z \geq 0 \\ 0 & \text{for } z < 0 \end{cases}$$

p(z) đạt giá trị tối đa tại z = 0

$$Mean : \bar{z} = \frac{1}{a} \qquad Varience : \sigma^2 = \frac{1}{a^2}$$



Ứng dụng:

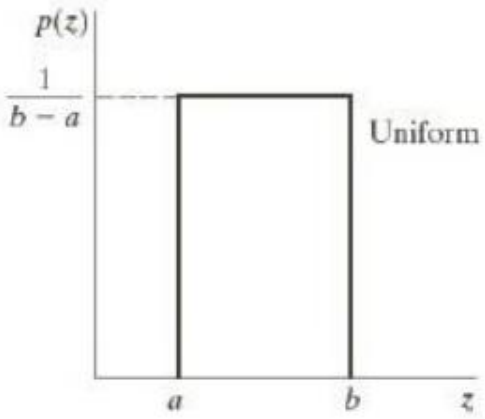
- Nó được sử dụng để mô tả kích thước của giọt mưa.
- Nó được sử dụng để mô tả sự dao động của công suất nhận được phản xạ từ một số mục tiêu nhất định
- Nó được ứng dụng trong hình ảnh Laser.

5.Nhiễu đồng đều (Uniform)

Nhiễu đồng đều phân bố đều trong một khoảng giá trị. Hàm mật độ xác suất của nhiễu đồng nhất được cho bởi:

$$p(z) = \begin{cases} \frac{1}{b-a} & \text{for } a \le z \le b \\ 0 & \text{otherwise} \end{cases}$$

$$Mean : \bar{z} = \frac{a+b}{2} \qquad Varience : \sigma^2 = \frac{(b-a)^2}{12}$$

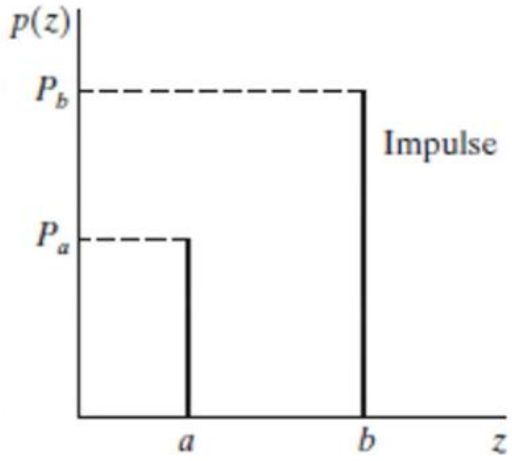


6.Nhiễu Salt and Pepper (Nhiễu Impulse)

Nhiễu Salt and Pepper xuất hiện dưới dạng các điểm sáng (muối) hoặc tối (tiêu). Hàm mật độ xác suất của nhiễu Muối và Tiêu được cho bởi:

$$p(z) = \begin{cases} P_a & \text{for } z = a \\ P_b & \text{for } z = b \\ 0 & \text{otherwise} \end{cases}$$

Nếu $P_a = P_b$ thì gọi là nhiễu đơn cực.



Nếu b > a, mức xám b sẽ xuất hiện dưới dạng một chấm sáng trong hình ảnh. Mức a sẽ xuất hiện như một chấm tối. Nhiễu muối và tiêu còn được gọi là nhiễu xung lưỡng cực (hai cực), nhiễu dữ liệu drop-out và tăng đột biến.

7.Nhiễu tuần hoàn (Periodic)

Nhiễu tuần hoàn trong hình ảnh xảy ra do nhiễu điện hoặc nhiễu cơ điện trong quá trình thu nhận hình ảnh. Đây là loại nhiễu phụ thuộc không gian duy nhất và các tham số được ước lượng bằng phổ Fourier của hình ảnh. Nhiễu tuần hoàn có xu hướng tạo ra các đỉnh tần số, thường có thể được phát hiện ngay cả bằng phân tích trực quan. Trung bình và phương sai được định nghĩa như sau:

$$\bar{z} = \sum_{i=0}^{L-1} z_i p_S(z_i)$$

$$\sigma^2 = \sum_{i=0}^{L-1} (z_i - \bar{z})^2 p_S(z_i)$$

III. PHỤC HỒI KHI CHỈ CÓ NHIỄU-LỌC KHÔNG GIAN

Khi hình ảnh chỉ bị nhiễu mà không có biến dạng khác, mô hình được đơn giản hóa thành:

$$g(x, y) = f(x, y) + \eta(x, y)$$

và trong miền tần số:

$$G(u, v) = F(u, v) + N(u, v)$$

Các thành phần nhiễu là không xác định, vì vậy việc trừ chúng khỏi g(x, y) hoặc G(u, v) không phải là cách tiếp cận thực tế. Trong trường hợp nhiễu tuần hoàn, có thể ước lượng N(u, v) từ phổ G(u, v). Do đó, N(u, v) có thể được trừ khỏi G(u, v) để thu được ước lượng của hình ảnh gốc. Lọc không gian có thể được thực hiện khi chỉ có nhiễu cộng thêm.

1.Bộ lọc trung bình (Mean Filters)

a) Bộ lọc trung bình số học *(Arithmetic Mean Filter):*

Đây là bộ lọc trung bình đơn giản nhất. Gọi S_{xy} là tập hợp các tọa độ trong vùng hình ảnh phụ có kích thước m*n với trung tâm tại điểm (x, y). Bộ lọc trung bình số học tính giá trị trung bình của hình ảnh bị nhiễu g(x, y) trong khu vực được xác định bởi S_{xy} . Giá trị của hình ảnh phục hồi f tại bất kỳ điểm (x, y) nào là trung bình số học được tính bằng cách sử dụng các pixel trong vùng được xác định bởi S_{xy} :

$$\hat{f}(x,y) = \frac{1}{mn} \sum_{(s,t)\in S_{xy}} g(s,t)$$

Thao tác này có thể được thực hiện bằng mặt nạ tích chập trong đó tất cả các hệ số có giá trị 1/mn. Bộ lọc trung bình làm mờ các biến đổi cục bộ trong hình ảnh, nhiễu được giảm bớt nhờ hiệu ứng làm mờ.

b) Bộ lọc trung bình hình học *(Geometric Mean Filter):*

Hình ảnh được phục hồi bằng cách lấy tích các pixel trong vùng lân cận, nâng lên lũy thừa 1 / $m\ n$:

$$\hat{f}(x,y) = \left[\prod_{(s,t)\in S_{xy}} g(s,t) \right]^{\frac{1}{mn}}$$

Ở đây, mỗi pixel được phục hồi là tích của các pixel trong cửa sổ hình ảnh phụ, được nâng lên lũy thừa 1/mn. Bộ lọc trung bình hình học đạt được độ làm mịn tương đương với bộ lọc trung bình số học, nhưng nó có xu hướng làm mất chi tiết hình ảnh trong quá trình này.

c) Bộ lọc trung bình hài hòa *(Harmonic Mean Filter):*

Phương pháp này sử dụng nghịch đảo của các pixel trong vùng lân cận:

$$\hat{f}(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}}$$

Bộ lọc trung bình điều hòa hoạt động tốt với nhiễu muối nhưng không hiệu quả với nhiễu tiêu. Nó cũng hoạt động tốt với các loại nhiễu khác.

d) Bộ lọc trung bình sóng hài Contra *(Contra-harmonic Mean Filter):*

Bộ lọc trung bình đối điều hòa tạo ra hình ảnh phục hồi dựa trên biểu thức:

$$\hat{f}(x,y) = \frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}$$

Trong đó Q được gọi là bậc của bộ lọc và bộ lọc này rất phù hợp để giảm ảnh hưởng của nhiễu muối và tiêu. Với giá trị Q dương, bộ lọc loại bỏ nhiễu tiêu. Với giá trị Q âm, nó loại bỏ nhiễu muối. Nó không thể thực hiện cả hai cùng lúc. Bộ lọc đối điều hòa trở thành bộ lọc trung bình số học nếu Q = 0 và trở thành bộ lọc điều hòa nếu Q = -1.

2.Bộ lọc thống kê thứ tự *(Order-Statistic Filters) :*

Bộ lọc thống kê thứ tự là các bộ lọc không gian có phản hồi dựa trên việc sắp xếp các pixel trong khu vực hình ảnh được bao phủ bởi bộ lọc. Phản hồi của bộ lọc tại bất kỳ điểm nào được xác định bởi kết quả xếp hạng.

a) Bộ lọc trung vị *(Median Filter):*

Đây là bộ lọc thống kê thứ tự nổi tiếng nhất. Nó thay thế giá trị của một pixel bằng trung vị của các mức xám trong vùng lân cận của pixel đó:

$$\hat{f}(x,y) = \underset{(s,t)\in S_{xy}}{\text{median}}\{g(s,t)\}$$

Giá trị của pixel tại (x, y) được bao gồm trong phép tính trung vị. Bộ lọc trung vị rất phổ biến vì với một số loại nhiễu ngẫu nhiên, chúng cung cấp khả năng giảm nhiễu tuyệt vời với ít mờ hơn đáng kể so với các bộ lọc làm mịn có kích thước tương tự. Chúng hiệu quả với nhiễu xung lưỡng cực và đơn cực.

b) Bộ lọc tối đa và tối thiểu *(Max and Min Filters):*

Bộ lọc trung vị đại diện cho phân vị thứ 50 của một tập hợp số đã được xếp hạng. Nếu sử dụng phân vị thứ 100, nó được gọi là "Bộ lọc Tối đa" (Max Filter). Nó được định nghĩa như sau:

$$\hat{f}(x,y) = \max_{(s,t)\in S_{xy}} \{g(s,t)\}$$

Bộ lọc này được sử dụng để tìm điểm sáng nhất trong hình ảnh. Nhiễu tiêu trong hình ảnh có giá trị rất thấp, nó được giảm bởi bộ lọc tối đa bằng cách sử dụng quá trình chọn tối đa trong khu vực phụ S_{xy}.

Bộ lọc phân vị thứ 0 là "Bộ lọc Tối thiểu" (Min Filter):

$$\hat{f}(x,y) = \min_{(s,t)\in S_{xy}} \{g(s,t)\}$$

Bộ lọc này hữu ích để tìm điểm tối nhất trong hình ảnh. Nó cũng giảm nhiễu muối nhờ thao tác tối thiểu.

c) Bộ lọc điểm giữa *(Midpoint Filter):*

Bộ lọc điểm giữa chỉ đơn giản tính điểm giữa của giá trị tối đa và tối thiểu trong khu vực được bao phủ bởi bộ lọc:

$$\hat{f}(x,y) = \frac{1}{2}\left[\max_{(s,t)\in S_{xy}} \{g(s,t)\} + \min_{(s,t)\in S_{xy}} \{g(s,t)\}\right]$$

Nó kết hợp thống kê thứ tự và trung bình. Bộ lọc này hoạt động tốt nhất với nhiễu phân bố ngẫu nhiên như nhiễu Gaussian hoặc nhiễu đồng nhất.

d) Bộ lọc trung bình cắt alpha *(Alpha-trimmed Mean Filter):*

Nếu chúng ta xóa d/2 giá trị cường độ thấp nhất và d/2 giá trị cường độ cao nhất của g(s,t) trong vùng lân cận S_{xy}. Gọi $g_r(s,t)$ đại diện cho mn - d pixel còn lại. Bộ lọc được hình thành bằng cách lấy trung bình các pixel còn lại được gọi là bộ lọc trung bình cắt alpha:

$$\hat{f}(x,y) = \frac{1}{mn-d} \sum_{(s,t)\in S_{xy}} g_r(s,t)$$

Giá trị của d có thể dao động từ 0 đến mn - 1. Nếu d = 0, bộ lọc này trở thành bộ lọc trung bình số học. Nếu d = mn - 1, bộ lọc trở thành bộ lọc trung vị. Với các giá trị khác của d, bộ lọc trung bình cắt alpha hữu ích cho nhiều loại nhiễu, chẳng hạn như kết hợp nhiễu muối-tiêu và nhiễu Gaussian.

IV. BỘ LỌC THÍCH ỨNG (Adaptive Filters)

Bộ lọc thích nghi thay đổi hành vi dựa trên các đặc tính thống kê của hình ảnh trong vùng lọc S_{xy}.

1.Bộ lọc giảm nhiễu cục bộ, thích nghi

Các thước đo thống kê đơn giản nhất của một biến ngẫu nhiên là trung bình và phương sai của nó. Trung bình cung cấp thước đo cường độ trung bình trong vùng mà trung bình được tính toán, và phương sai cung cấp thước đo độ tương phản trong vùng đó.

Giả sử bộ lọc hoạt động trên một vùng cục bộ S_{xy}. Phản hồi của bộ lọc tại bất kỳ điểm (x, y) nào dựa trên bốn đại lượng:

(a) g(x, y), Giá trị pixel nhiễu tại (x, y).
(b) σ_η^2, Phương sai nhiễu tổng thể.

(c) m_L, trung bình cục bộ của các pixel trong S_{xy}.

(d) σ_L^2, phương sai cục bộ của các pixel trong S_{xy}.

Do đó, hành vi của bộ lọc là:

- Nếu σ_η^2 bằng 0, bộ lọc chỉ cần trả về giá trị của g(x, y).
- Nếu phương sai cục bộ cao so với σ_η^2 (tức là $\sigma_L^2 > \sigma_\eta^2$), bộ lọc nên trả về giá trị gần với g(x, y).
- Nếu hai phương sai bằng nhau, bộ lọc trả về giá trị trung bình số học của các pixel trong S_{xy}.

Một bộ lọc thích nghi để thu được hình ảnh phục hồi là:

$$\hat{f}(x,y) = g(x,y) - \frac{\sigma_\eta^2}{\sigma_L^2}[g(x,y) - m_L]$$

Đại lượng duy nhất cần biết hoặc ước lượng là phương sai của nhiễu tổng thể σ_η^2. Các tham số khác được tính từ các pixel trong S_{xy}.

2.Bộ lọc trung vị thích nghi *(Adaptive Median Filter) :*

Bộ lọc trung vị thích nghi được sử dụng để bảo toàn chi tiết trong khi làm mịn nhiễu không xung. Tuy nhiên, nó thay đổi kích thước của S_{xy} trong quá trình lọc, tùy thuộc vào một số điều kiện. Đầu ra của bộ lọc là một giá trị duy nhất dùng để thay thế giá trị của pixel tại (x, y). Hãy xem xét các tham số sau:

z_{min} = giá trị cường độ tối thiểu trong S_{xy}

z_{max} = giá trị cường độ tối đa trong S_{xy}

z_{med} = trung vị của các giá trị cường độ trong S_{xy}

z_{xy} = giá trị cường độ tại tọa độ (x, y)

S_{max} = kích thước tối đa cho phép của S_{xy}

Thuật toán Lọc Trung vị Thích nghi hoạt động trong hai giai đoạn, được ký hiệu là Giai đoạn A và Giai đoạn B như sau:

Giai đoạn A:

A1 = z_{med} - z_{min}

A2 = z_{med} - z_{max}

Nếu A1 > 0 và A2 < 0, chuyển sang Giai đoạn B

Ngược lại, tăng kích thước cửa sổ

Nếu window_size ≤ S_{max}, lặp lại Giai đoạn A

Ngược lại, xuất ra z_{med}

Giai đoạn B:

B1 = z_{xy} - z_{min}

B2 = z_{xy} - z_{max}

Nếu B1 > 0 và B2 < 0, xuất ra z_{xy}

Ngược lại, xuất ra z_{med}

V. GIẢM NHIỄU ĐỊNH KỲ BẰNG CÁCH LỌC TRONG MIỀN TẦN SỐ

Nhiễu tuần hoàn trong hình ảnh xuất hiện dưới dạng các vụ nổ năng lượng tập trung trong biến đổi Fourier tại các vị trí tương ứng với tần số của nhiễu tuần hoàn. Điều này có thể được loại bỏ bằng cách sử dụng các bộ lọc chọn lọc.

1.Bộ lọc Loại bỏ Dải (Band Reject Filter):

Hàm truyền của bộ lọc loại bỏ dải được định nghĩa như sau:

Ideal	Butterworth	Gaussian
$H(u,v) = \begin{cases} 0 & \text{if } D_0 - \frac{W}{2} \le D \le D_0 + \frac{W}{2} \\ 1 & \text{otherwise} \end{cases}$	$H(u,v) = \frac{1}{1 + \left[\frac{DW}{D^2 - D_0^2}\right]^{2n}}$	$H(u,v) = 1 - e^{-\left[\frac{D^2 - D_0^2}{DW}\right]^2}$

Trong đó:

- W là chiều rộng của dải,
- D(u, v) là khoảng cách từ trung tâm của bộ lọc,
- D_0 là tần số cắt,
- n là bậc của bộ lọc Butterworth.

Bộ lọc loại bỏ dải rất hiệu quả trong việc loại bỏ nhiễu tuần hoàn và hiệu ứng rung chuông thường nhỏ. Các biểu đồ phối cảnh của các bộ lọc này là:

a b c

Hình 2. Biểu đồ phối cảnh của (a) Lý tưởng (b) Butterworth và (c) Bộ lọc loại bỏ dải Gaussian

2.Bộ lọc băng thông (Band Pass Filter):

Bộ lọc thông dải thực hiện thao tác ngược lại với bộ lọc loại bỏ dải. Bộ lọc thông dải được thu từ bộ lọc loại bỏ dải như sau:

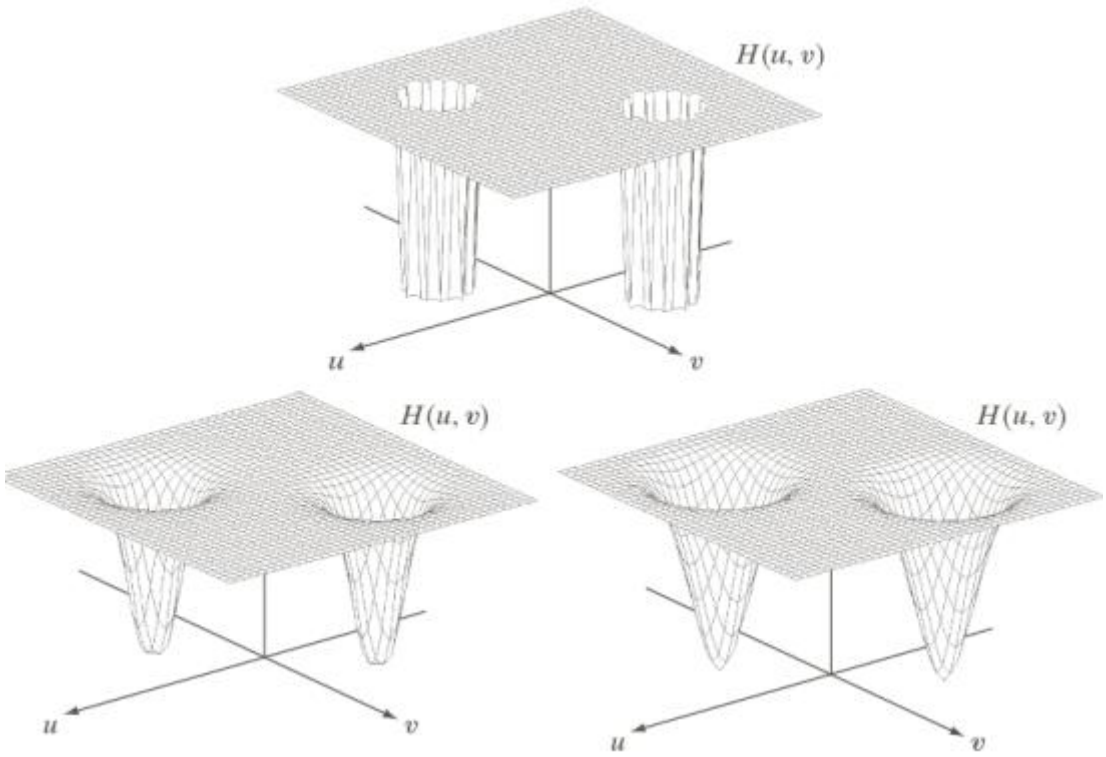
$$H_{BP}(u,v) = 1 - H_{BR}(u,v)$$

3.Bộ lọc Rãnh (Notch Filters):

Bộ lọc rãnh loại bỏ (hoặc cho qua) các tần số trong một vùng lân cận được xác định trước quanh trung tâm của hình chữ nhật tần số. Nó được xây dựng dưới dạng tích của các bộ lọc thông cao có trung tâm đã được dịch chuyển đến trung tâm của các rãnh. Dạng tổng quát được định nghĩa như sau:

$$H_{NR}(u,v) = \prod_{k=1}^{Q} H_k(u,v) H_{-k}(u,v)$$

Trong đó $H_k(u,v)$ và $H_{-k}(u,v)$ là các bộ lọc thông cao có trung tâm tại (u_k, v_k) và $(-u_k, -v_k)$ tương ứng. Các trung tâm này được xác định tương ứng với trung tâm của hình chữ nhật tần số (M/2, N/2).



Hình 3. Biểu đồ phối cảnh của (a) Lý tưởng (b) Butterworth và (c) Bộ lọc loại bỏ Notch Gaussian

Bộ lọc thông rãnh (Notch Pass Filter - NP) được thu từ bộ lọc loại bỏ rãnh (Notch Reject Filter - NR) bằng cách:

$$H_{NP}(u,v) = 1 - H_{NR}(u,v)$$

VI. XUỐNG CẤP TUYẾN TÍNH, VỊ TRÍ BẤT BIẾN

Mối quan hệ đầu vào - đầu ra trước giai đoạn phục hồi được biểu diễn như sau:

$$g(x,y) = H[f(x,y)] + \eta(x,y)$$

Giả sử rằng $\eta(x, y) = 0$, thì:

$$g(x, y) = H[f(x, y)]$$

Nếu H là tuyến tính:

$$H[af_1(x,y) + bf_2(x,y)] = aH[f_1(x,y)] + bH[f_2(x,y)]$$

Trong đó a và b là các số vô hướng, $f_1(x, y)$ và $f_2(x, y)$ là bất kỳ hai hình ảnh đầu vào nào. Nếu a = b = 1:

$$H[f_1(x,y) + f_2(x,y)] = H[f_1(x,y)] + H[f_2(x,y)]$$

Đây được gọi là tính chất cộng. Tính chất này nói rằng, nếu H là một toán tử tuyến tính, phản hồi đối với tổng của hai đầu vào bằng tổng của hai phản hồi.

Một toán tử có mối quan hệ đầu vào - đầu ra g(x, y) = H[f(x, y)] được gọi là không đổi theo vị trí nếu:

$$H[f(x - \alpha, y - \beta)] = g(x - \alpha, y - \beta)$$

Điều này cho thấy phản hồi tại bất kỳ điểm nào trong hình ảnh chỉ phụ thuộc vào giá trị của đầu vào tại điểm đó, không phụ thuộc vào vị trí của nó. Nếu tín hiệu xung có thể được xem xét:

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)\delta(x-\alpha,y-\beta)d\alpha d\beta$$

Xung (impulse)

$$g(x,y) = H[f(x,y)] = H\left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)\delta(x-\alpha,y-\beta)d\alpha d\beta\right]$$

Do tính tuyến tính:

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)H[\delta(x-\alpha,y-\beta)]d\alpha d\beta$$

Gọi: *h(x, y) = H[δ(x, y)]*

$$h(x,\alpha,y,\beta) = H[\delta(x-\alpha,y-\beta)]$$

hàm phản hồi xung - point spread function (Impulse reponse)

Nếu không đổi theo vị trí:

$$H[\delta(x-\alpha,y-\beta)] = h(x-\alpha,y-\beta)$$

Do đó:

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)h(x-\alpha,y-\beta)d\alpha d\beta$$

Khi có nhiễu: $\eta(x,y) \neq 0$

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)h(x-\alpha,y-\beta)d\alpha d\beta + \eta(x,y)$$

VII. ƯỚC TÍNH HÀM SUY THOÁI

Có ba cách chính để ước lượng hàm suy giảm nhằm sử dụng trong phục hồi hình ảnh: (1) Quan sát, (2) Thực nghiệm, và (3) Mô hình hóa toán học.

1.Ước tính bằng quan sát hình ảnh:

Trong quá trình này, hàm suy giảm được ước lượng bằng cách quan sát hình ảnh. Chọn một hình ảnh phụ có nội dung tín hiệu mạnh. Gọi hình ảnh phụ được quan sát là $g_s(x,y)$ và hình ảnh phụ đã xử lý là $\hat{f}(x,y)$. Hàm suy giảm ước lượng có thể được biểu diễn như sau:

$$H_s(u,v) = \frac{G_s(u,v)}{\hat{F}_s(u,v)}$$

Từ các đặc tính của phương trình này, sau đó chúng ta suy ra hàm suy giảm hoàn chỉnh *H(u, v)* dựa trên việc xem xét tính không đổi theo vị trí.

2.Ước tính bằng thử nghiệm:

Hình ảnh bị suy giảm có thể được ước lượng chính xác khi thiết bị giống hệt với thiết bị dùng để thu được hình ảnh bị suy giảm. Các hình ảnh tương tự với hình ảnh bị suy giảm có thể được thu thập với các cài đặt hệ thống khác nhau cho đến khi chúng bị suy giảm gần giống nhất với hình ảnh mà chúng ta

muốn phục hồi. Bây giờ, thu được phản hồi xung của sự suy giảm bằng cách chụp ảnh một xung bằng cùng cài đặt hệ thống.

Một xung được mô phỏng bằng một chấm sáng, càng sáng càng tốt để giảm ảnh hưởng của nhiễu. Sau đó, hình ảnh suy giảm có thể được biểu diễn như sau:

$$H(u,v) = \frac{G(u,v)}{A}$$

Trong đó *G(u, v)* là biến đổi Fourier của hình ảnh được quan sát và A là một hằng số mô tả cường độ của xung.

3.Ước tính bằng mô hình hóa:

Hàm suy giảm hình ảnh có thể được ước lượng bằng mô hình hóa bao gồm các điều kiện môi trường gây ra sự suy giảm. Nó có thể được biểu diễn như sau:

$$H(u,v) = \frac{G(u,v)}{A}$$

Trong đó k là một hằng số phụ thuộc vào bản chất của sự nhiễu loạn. Gọi f(x, y) là một hình ảnh trải qua chuyển động phẳng và x_o(t) cùng y_o(t) là các thành phần thay đổi theo thời gian theo hướng x và y. Hình ảnh mờ toàn bộ g(x, y) được biểu diễn như sau:

$$g(x,y) = \int_0^T f[x - x_0(t), y - y_0(t)]dt$$

Biến đổi Fourier của g(x, y) là G(u, v):

$$G(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y)e^{-j2\pi(ux+vy)}dxdy$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \left[\int_{0}^{T} f[x - x_0(t), y - y_0(t)] dt \right] e^{-j2\pi(ux+vy)} dx dy$$

Đảo ngược thứ tự tích phân, ta được:

$$G(u,v) = \int_{0}^{T} \left[\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f[x - x_0(t), y - y_0(t)] e^{-j2\pi(ux+vy)} dx dy \right] dt$$

$$G(u,v) = \int_{0}^{T} F(u,v) e^{-j2\pi[ux_0(t)+vy_0(t)]} dt$$

$$= F(u,v) \int_{0}^{T} e^{-j2\pi[ux_0(t)+vy_0(t)]} dt$$

Định nghĩa *H(u, v)*:

$$H(u,v) = \int_{0}^{T} e^{-j2\pi[ux_0(t)+vy_0(t)]} dt$$

Do đó, biểu thức trên có thể được viết lại là: *G(u, v) = H(u, v) F(u, v)*

Nếu các biến chuyển động x_o(t) và y_o(t) được biết, thì hàm suy giảm H(u, v) trở thành:

$$H(u,v) = \frac{T}{\pi(ua+vb)} \sin[\pi(ua+vb)] e^{-j\pi(ua+vb)}$$

VIII. LỌC NGHỊCH ĐẢO (Inverse Filtering)

Cách tiếp cận đơn giản nhất để phục hồi là lọc ngược trực tiếp, trong đó chúng ta có thể ước lượng F(u, v) của biến đổi của hình ảnh gốc chỉ bằng cách chia biến đổi của hình ảnh bị suy giảm G(u, v) cho hàm suy giảm:

$$\hat{F}(u,v) = \frac{G(u,v)}{H(u,v)}$$

Chúng ta biết rằng: *G(u, v) = H(u, v) F(u, v) + N(u, v).* Do đó:

$$\hat{F}(u,v) = F(u,v) + \frac{N(u,v)}{H(u,v)}$$

Từ biểu thức trên, chúng ta có thể thấy rằng ngay cả khi biết hàm suy giảm, chúng ta không thể khôi phục chính xác hình ảnh không bị suy giảm vì N(u, v) không được biết. Nếu hàm suy giảm có giá trị bằng 0 hoặc rất nhỏ, thì tỷ số N(u, v) / H(u, v) có thể dễ dàng chi phối $\hat{F}(x,y)$. Để tránh nhược điểm này, chúng ta giới hạn giá trị tần số của bộ lọc gần gốc tọa độ vì giá trị H(u, v) là tối đa tại gốc.

IX. LỌC SAI SỐ BÌNH PHƯƠNG TỐI THIỂU (Wiener Filtering)

Trong quá trình lọc này, nó kết hợp cả hàm suy giảm và các đặc tính thống kê của nhiễu. Hình ảnh và nhiễu trong phương pháp này được xem là các biến ngẫu nhiên. Mục tiêu là tìm một ước lượng $\hat{f}$ của hình ảnh không bị hỏng sao cho sai số bình phương trung bình giữa chúng được giảm thiểu. Sai số này được đo bằng:

$$e^2 = E\{(f - \hat{f})^2\}$$

Trong đó E {} là giá trị kỳ vọng của đối số. Giả định rằng nhiễu và hình ảnh không tương quan; một trong hai có trung bình bằng 0; các mức cường độ trong ước lượng là hàm tuyến tính của các mức trong hình ảnh bị suy giảm. Dựa trên các điều kiện này, giá trị tối thiểu của hàm sai số trong miền tần số được cho bởi:

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)S_f(u,v)}{S_f(u,v)|H(u,v)|^2 + S_\eta(u,v)}\right]G(u,v)$$

$$= \frac{H^*(u,v)}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)}G(u,v)$$

$$= \left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)}\right]G(u,v)$$

Kết quả này được gọi là "Bộ lọc Wiener". Các số hạng bên trong dấu ngoặc thường được gọi là bộ lọc sai số bình phương trung bình nhỏ nhất hoặc bộ lọc sai số bình phương nhỏ nhất. Nó không có vấn đề tương tự như bộ lọc nghịch đảo có số 0 trong hàm suy giảm, trừ khi toàn bộ mẫu số bằng không cho cùng giá trị của u và v. H(u, v) = hàm suy giảm H*(u, v) = liên hợp phức của H(u, v)

$S_\eta(u,v) = |N(u,v)|^2$ phổ công suất của nhiễu

$S_f(u,v) = |F(u,v)|^2$ phổ công suất của hình ảnh không bị suy giảm

Tỷ số tín hiệu trên nhiễu trong miền tần số:

$$\text{SNR} = \frac{\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}|F(u,v)|^2}{\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}|N(u,v)|^2}$$

Tỷ số tín hiệu trên nhiễu trong miền không gian:

$$\text{SNR} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[f(x,y)-\hat{f}(x,y)]^2}$$

Sai số bình phương trung bình được tính bằng biểu thức:

$$\text{MSE} = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[f(x,y)-\hat{f}(x,y)]^2$$

Biểu thức được sửa đổi để ước lượng f bằng cách sử dụng lọc sai số bình phương tối thiểu:

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)}\cdot\frac{|H(u,v)|^2}{|H(u,v)|^2+K}\right]G(u,v)$$

X. LỌC BÌNH PHƯƠNG BÉ NHẤT CÓ RÀNG BUỘC (Constrained Least Square Filtering)

Vấn đề với bộ lọc Wiener là cần biết phổ công suất của nhiễu và hình ảnh. Hình ảnh bị suy giảm trong miền không gian được cho bởi:

$$g(x, y) = f(x, y) * h(x, y) + \eta(x, y)$$

Dưới dạng vector-ma trận:

$$\mathbf{g} = \mathbf{Hf} + \eta$$

Trong đó g, f, η là các vector có kích thước MNx1.H là ma trận có kích thước MN x MN, rất lớn và rất nhạy với nhiễu. Tính tối ưu của phục hồi dựa trên thước đo độ mịn: sử dụng toán tử Laplacian. Phục hồi phải bị ràng buộc bởi các tham số để tìm hàm tiêu chí tối thiểu C được định nghĩa như sau:

$$C = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\nabla^2 f(x,y)]^2$$

Tuân theo ràng buộc:

$$\|\mathbf{g} - \mathbf{H}\hat{\mathbf{f}}\|^2 = \|\eta\|^2$$

Giải pháp trong miền tần số cho bài toán tối ưu hóa này được cho bởi biểu thức:

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)}{|H(u,v)|^2 + \gamma |P(u,v)|^2} \right] G(u,v)$$

Trong đó γ là một tham số phải được điều chỉnh để thỏa mãn ràng buộc và P(u, v) là biến đổi Fourier của hàm:

$$p(x,y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Có thể điều chỉnh tham số γ tương tác cho đến khi đạt được kết quả chấp nhận được. Quy trình tính γ bằng lặp được thực hiện như sau. Định nghĩa một vector dư r là:

$$\mathbf{r} = \mathbf{g} - \mathbf{H}\hat{\mathbf{f}}$$

Do đó $\widehat{F}(u,v)$ là hàm của γ, thì r cũng là hàm của tham số này và là một hàm tăng đơn điệu của γ:

$$\phi(\gamma) = \mathbf{r}^T\mathbf{r} = \|\mathbf{r}\|^2$$

Chúng ta muốn điều chỉnh γ sao cho:

$$\|\mathbf{r}\|^2 = \|\eta\|^2 \pm a$$

Trong đó a là yếu tố độ chính xác. Nếu $||r||^2 = ||\eta||^2$, ràng buộc được thỏa mãn. Vì φ(γ) là đơn điệu, việc tìm giá trị mong muốn của γ không khó. Phương pháp tiếp cận:

1. Chỉ định giá trị ban đầu của γ.
2. Tính $||\mathbf{r}||^2$.
3. Dừng nếu phương trình (5.9-8) được thỏa mãn; ngược lại quay lại bước 2 sau khi tăng γ nếu $||r||^2 < ||\eta||^2$ - a hoặc giảm γ nếu $||r||^2 < ||\eta||^2$ + a. Sử dụng giá trị mới của γ trong phương trình (5.9-4) để tính lại ước lượng tối ưu $\widehat{F}$(u, v).

Phương sai và trung bình của toàn bộ hình ảnh là:

$$\sigma_\eta^2 = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\eta(x,y) - m_\eta]^2$$

$$m_\eta = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\eta(x,y)$$

Do đó nhiễu là:

$$\|\eta\|^2 = MN[\sigma_\eta^2 + m_\eta^2]$$

XI. BỘ LỌC TRUNG BÌNH HÌNH HỌC (Geometric Mean Filter)

Bộ lọc trung bình hình học là một dạng tổng quát nhẹ của bộ lọc Wiener dưới dạng:

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)}{|H(u,v)|^2}\right]^\alpha \left[\frac{H^*(u,v)}{|H(u,v)|^2 + \beta\left[\frac{S_\eta(u,v)}{S_f(u,v)}\right]}\right]^{1-\alpha} G(u,v)$$

Trong đó α và β là các hằng số thực dương. Dựa trên giá trị của và β, bộ lọc trung bình hình học thực hiện các hành động khác nhau:

α = 1 => bộ lọc ngược (inverse filter)

α = 0 => bộ lọc Wiener tham số (bộ lọc Wiener chuẩn khi β = 1)

α = 1/2 => trung bình hình học thực sự

α = 1/2 và β = 1 => bộ lọc cân bằng phổ

TÀI LIỆU THAM KHẢO

1. Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th ed.). Pearson.
2. Jain, A. K. (1989). *Fundamentals of Digital Image Processing*. Prentice-Hall.
3. Pratt, W. K. (2007). *Digital Image Processing: PIKS Scientific Inside* (4th ed.). Wiley-Interscience.
4. Lim, J. S. (1990). *Two-Dimensional Signal and Image Processing*. Prentice-Hall.
5. Oppenheim, A. V., & Schafer, R. W. (2009). *Discrete-Time Signal Processing* (3rd ed.). Pearson.
6. Bovik, A. (2010). *Handbook of Image and Video Processing* (2nd ed.). Academic Press.
7. Wiener, N. (1949). *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. MIT Press.
8. Jain, R. (1995). *Machine Vision*. McGraw-Hill.
9. Castleman, K. R. (1996). *Digital Image Processing*. Prentice-Hall.
10. Schalkoff, R. J. (1989). *Digital Image Processing and Computer Vision*. Wiley.

KẾT LUẬN

Tài liệu này trình bày các kỹ thuật phục hồi hình ảnh, từ mô hình suy giảm đến các phương pháp lọc không gian và tần số. Nhóm đã thử nghiệm thực tế trên ảnh chụp và nhận thấy bộ lọc trung vị và Wiener cho kết quả tốt nhất với nhiễu Salt and Pepper và Gaussian. Trong tương lai, chúng tôi khuyến nghị nghiên cứu thêm các kỹ thuật học sâu để cải thiện hiệu quả phục hồi.

TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

----- ☙ 🕮 ❧ -----

TIỂU LUẬN
PHÂN TÍCH XỬ LÝ ẢNH

ĐỀ TÀI: "CT Liver Segmentation via PVT-based Encoding and Refined Decoding"

Giảng viên hướng dẫn: **PGS.TS.Phạm Thế Bảo**

Nhóm : 3
Thành viên:
Hồ Văn Quyến – 3122410352
Trần Thanh Phương – 3122410333
Huỳnh Quang Quân – 3122410341
Võ Anh Tuấn – 3122410453

Năm học: 2024 - 2025
Thành phố Hồ Chí Minh, tháng 4, năm 2025

MỤC LỤC

LỜI NÓI ĐẦU 4
I. GIỚI THIỆU 5
1. Bối cảnh và tầm quan trọng 5
2. Thách thức 5
3. Mục tiêu dự án 5
II. PHƯƠNG PHÁP 7
1. Kiến trúc Mô hình (PVT_LiverSegImproved) 7
2. Xử lý dữ liệu (Data Handling) 11
3. Huấn luyện (Training) 16
III. THỰC NGHIỆM VÀ KẾT QUẢ 23
1. Cài đặt (Set up) 23
2. Quy trình Thực thi (Workflow) 23
3. Đánh giá (Evaluation) 27
4. Phân tích (Analysis) 27
IV. THẢO LUẬN 30
1. Điểm mạnh 30
2. Hạn chế và Hướng phát triển 30
V. KẾT LUẬN 32

NHẬN XÉT, ĐÁNH GIÁ CỦA GIẢNG VIÊN

LỜI NÓI ĐẦU

Dự án này trình bày và đánh giá một phương pháp học sâu tiên tiến để phân vùng tự động gan trên ảnh cắt lớp vi tính (CT). Phương pháp cốt lõi dựa trên kiến trúc mạng nơ-ron kết hợp bộ mã hóa Pyramid Vision Transformer phiên bản 2 (pvt_v2_b2), được huấn luyện trước trên ImageNet để trích xuất đặc trưng đa tỷ lệ hiệu quả, với bộ giải mã theo kiểu U-Net được thiết kế để phục hồi chi tiết không gian và tạo ra bản đồ phân vùng chính xác. Dự án bao gồm hai quy trình huấn luyện song song: một quy trình sử dụng dữ liệu gốc .nii được xử lý "on-the-fly", và một quy trình tối ưu hóa sử dụng dữ liệu đã được tiền xử lý và lưu trữ dưới dạng .npz để tăng tốc độ huấn luyện. Các kỹ thuật chính được áp dụng bao gồm augmentation dữ liệu (lật, xoay), hàm loss kết hợp (Dice + BCE), tối ưu hóa Adam, điều chỉnh learning rate (ReduceLROnPlateau), và dừng sớm (EarlyStopping) để đảm bảo huấn luyện ổn định và hiệu quả. Một kịch bản dự đoán (inference.py) được cung cấp để áp dụng mô hình đã huấn luyện lên dữ liệu thử nghiệm và trực quan hóa kết quả phân vùng trên lát cắt trung tâm. Hiệu suất mô hình được theo dõi chặt chẽ thông qua các chỉ số Dice Coefficient và Pixel Accuracy trên tập huấn luyện và kiểm định trong suốt quá trình huấn luyện.

I. GIỚI THIỆU

1. Bối cảnh và tầm quan trọng

Phân vùng gan chính xác từ ảnh CT là một nhiệm vụ cơ bản và thiết yếu trong thực hành lâm sàng. Nó đóng vai trò quan trọng trong việc:

- **Chẩn đoán:** Xác định kích thước, hình dạng, và thể tích gan, hỗ trợ phát hiện các bất thường như gan nhiễm mỡ, xơ gan, hoặc khối u.
- **Lập kế hoạch điều trị:** Xác định kích thước, hình dạng, và thể tích gan, hỗ trợ phát hiện các bất thường như gan nhiễm mỡ, xơ gan, hoặc khối u.
- **Theo dõi:** Đánh giá sự thay đổi thể tích gan hoặc khối u theo thời gian để theo dõi đáp ứng điều trị.

Việc phân vùng thủ công bởi các chuyên gia X-quang, mặc dù là tiêu chuẩn vàng, nhưng tốn kém thời gian, công sức và có thể thay đổi giữa những người đọc khác nhau. Do đó, các phương pháp tự động hóa, đặc biệt là dựa trên học sâu, có tiềm năng lớn để cải thiện hiệu quả và tính nhất quán.

2. Thách thức

- **Biến đổi hình thái:** Hình dạng và kích thước gan thay đổi đáng kể giữa các bệnh nhân.
- **Cường độ tương phản thấp:** Ranh giới giữa gan và các cơ quan lân cận (dạ dày, thận, cơ hoành) đôi khi không rõ ràng.
- **Bệnh lý:** Sự hiện diện của khối u hoặc các bệnh lý khác có thể làm thay đổi hình dạng và mật độ của gan.
- **Nhiễu ảnh:** Chất lượng ảnh CT có thể bị ảnh hưởng bởi nhiễu hoặc các tạo tác (artifacts).

3. Mục tiêu dự án

Dự án này đặt ra các mục tiêu cụ thể:

- Triển khai và huấn luyện một mô hình phân vùng 2D dựa trên kiến trúc kết hợp PVT encoder và U-Net decoder.
- Đánh giá hiệu quả của việc sử dụng PVT (một kiến trúc dựa trên Transformer) làm bộ mã hóa cho bài toán phân vùng ảnh y tế này.

- So sánh hiệu quả và tốc độ của hai quy trình huấn luyện: sử dụng dữ liệu tiền xử lý (.npz) và xử lý dữ liệu gốc (.nii) on-the-fly.
- Xây dựng một pipeline hoàn chỉnh từ tiền xử lý, huấn luyện, đánh giá đến dự đoán.
- Cung cấp cơ sở để phát triển các mô hình phân vùng 3D hoặc phân vùng tổn thương trong tương lai.

II. PHƯƠNG PHÁP

1. Kiến trúc Mô hình (PVT_LiverSegImproved)

Mô hình PVT_LiverSegImproved là trọng tâm của dự án, được thiết kế theo kiến trúc encoder-decoder:

- **Bộ mã hóa (Encoder):**
    - **Backbone:** Sử dụng pvt_v2_b2 từ thư viện tim. PVTv2 được chọn vì khả năng tạo ra các bản đồ đặc trưng đa tỷ lệ với hiệu quả tính toán tốt hơn so với các Transformer truyền thống và một số CNN. Nó sử dụng cửa sổ chú ý được điều chỉnh (spatial reduction attention) để giảm chi phí tính toán.
    - **Pretrained Weights:** Khởi tạo với trọng số được huấn luyện trước trên ImageNet (pretrained=True), giúp mô hình học nhanh hơn và tổng quát hóa tốt hơn trên dữ liệu y tế, vốn thường có số lượng hạn chế.
    - **Feature Extraction:** Chỉ lấy các đặc trưng từ 3 tầng cuối của PVT (features_only=True, self.backbone.feature_info[-3:]). Các tầng này cung cấp các đặc trưng ngữ nghĩa mạnh mẽ ở các độ phân giải khác nhau, phù hợp cho việc kết hợp trong bộ giải mã.
    - **Input Handling:** Do PVT được huấn luyện trước trên ảnh RGB (3 kênh), đầu vào là ảnh CT thang độ xám (1 kênh) được lặp lại 3 lần (x.repeat(1, 3, 1, 1)) để phù hợp với trọng số đã huấn luyện trước.
- **Khối Trung tâm (Center Block / Bottleneck:**
    - Áp dụng một ConvBlock (bao gồm 2 lớp Conv 3x3, BatchNorm, ReLU) cho bản đồ đặc trưng có độ phân giải thấp nhất (f3) từ encoder. Mục đích là tăng độ sâu của mạng tại điểm "thắt cổ chai", cho phép mô hình học các biểu diễn phức tạp hơn trước khi bắt đầu quá trình giải mã.
- **Bộ giải mã (Decoder):**
    - Áp dụng một ConvBlock (bao gồm 2 lớp Conv 3x3, BatchNorm, ReLU) cho bản đồ đặc trưng có độ phân giải thấp nhất (f3) từ encoder. Mục

đích là tăng độ sâu của mạng tại điểm "thắt cổ chai", cho phép mô hình học các biểu diễn phức tạp hơn trước khi bắt đầu quá trình giải mã.

- **Kiểu U-Net:** Sử dụng cấu trúc giải mã đối xứng với encoder, tương tự U-Net.
- **Up-sampling:** Sử dụng nn.Upsample(scale_factor=2, mode='bilinear', align_corners=False). Nội suy song tuyến tính (bilinear) là một phương pháp tăng mẫu đơn giản, nhanh và thường đủ hiệu quả, tránh được các tạo tác dạng bàn cờ (checkerboard artifacts) có thể xảy ra với transposed convolution. align_corners=False là cài đặt được khuyến nghị.
- **Skip Connections:** Đặc trưng từ encoder (f1, f2) được kết nối trực tiếp với các tầng tương ứng trong decoder thông qua phép ghép nối (torch.cat). Điều này cực kỳ quan trọng, cho phép bộ giải mã tái sử dụng các đặc trưng chi tiết, độ phân giải cao từ bộ mã hóa, giúp khôi phục lại các ranh giới phân vùng bị mất trong quá trình down-sampling của encoder.
- **Kích thước không khớp:** Nếu kích thước không gian của đặc trưng từ skip connection và đặc trưng đã upsample không khớp (có thể xảy ra do phép pooling/convolution trong encoder), đặc trưng upsample được nội suy lại bằng F.interpolate để khớp kích thước trước khi ghép nối.
- **Refinement Blocks:** Mỗi tầng giải mã (UpBlock) sử dụng một ConvBlock để hợp nhất thông tin từ skip connection và tầng trước, đồng thời giảm số kênh đặc trưng.

- **Đầu ra (Output Layer):**
  - Một lớp nn.Conv2d với kernel 1x1 cuối cùng đóng vai trò như một bộ phân loại pixel, chuyển đổi bản đồ đặc trưng 64 kênh từ tầng giải mã cuối thành bản đồ đầu ra 1 kênh (logits cho xác suất gan).
  - F.interpolate được sử dụng lần cuối để đảm bảo kích thước đầu ra khớp chính xác với kích thước ảnh đầu vào (256x256 sau transform).

```
# file: LiTSpreprocessed.py (Tương tự trong LiTS_full.py, inference.py)

import torch.nn as nn
import timm
import torch.nn.functional as F
import torch

# ... các import khác ...

class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(ConvBlock, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )
    def forward(self, x):
        return self.conv(x)

class UpBlock(nn.Module):
    def __init__(self, in_channels, skip_channels, out_channels):
        super(UpBlock, self).__init__()
        self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=False)
        # Khối Conv xử lý kênh sau khi concat (upsampled + skip)
        self.conv_block = ConvBlock(in_channels + skip_channels, out_channels)
    def forward(self, x, skip):
        x = self.up(x) # Upsample feature map từ tầng dưới
        # Đảm bảo kích thước không gian khớp với skip feature map
        if x.shape[2:] != skip.shape[2:]:
            x = F.interpolate(x, size=skip.shape[2:], mode='bilinear',
align_corners=False)
        # Ghép nối (concatenate) theo chiều kênh
        x = torch.cat([x, skip], dim=1)
        # Đưa qua ConvBlock để học và giảm kênh
        return self.conv_block(x)

class PVT_LiverSegImproved(nn.Module):
    def __init__(self, backbone_name="pvt_v2_b2", num_classes=1):
        super(PVT_LiverSegImproved, self).__init__()
        # Load backbone PVT đã huấn luyện trước, chỉ lấy features
        self.backbone = timm.create_model(backbone_name, pretrained=True,
features_only=True)
```

```
        # Lấy thông tin số kênh của 3 tầng đặc trưng cuối cùng từ backbone
        self.encoder_channels = [feat['num_chs'] for feat in
self.backbone.feature_info][-3:]

        # Center (Bottleneck) block xử lý feature map sâu nhất (độ phân giải thấp
nhất)
        self.center = ConvBlock(self.encoder_channels[2], 256)

        # Decoder blocks: upsample, concat skip connection, rồi qua ConvBlock
        # Tầng 1: Từ center (256 kênh) + skip connection f2 (encoder_channels[1]) -
> 128 kênh
        self.up1 = UpBlock(256, self.encoder_channels[1], 128)
        # Tầng 2: Từ up1 (128 kênh) + skip connection f1 (encoder_channels[0]) ->
64 kênh
        self.up2 = UpBlock(128, self.encoder_channels[0], 64)

        # Tầng upsample cuối cùng (không có skip connection)
        self.up3 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=False)
        # Lớp Conv 1x1 cuối cùng để tạo ra output mask (1 kênh)
        self.out_conv = nn.Conv2d(64, num_classes, kernel_size=1)

    def forward(self, x):
        B, C, H, W = x.shape
        # Nếu ảnh đầu vào là grayscale (1 kênh), lặp lại thành 3 kênh để phù hợp
backbone pretrained
        if C == 1:
            x = x.repeat(1, 3, 1, 1)

        # Lấy các feature map từ encoder
        features = self.backbone(x)
        f1 = features[-3] # Tầng nông nhất (độ phân giải cao)
        f2 = features[-2]
        f3 = features[-1] # Tầng sâu nhất (độ phân giải thấp)

        # Qua Center block
        center = self.center(f3)

        # Qua các tầng Decoder
        d1 = self.up1(center, f2) # Giải mã tầng 1
        d2 = self.up2(d1, f1)    # Giải mã tầng 2
        d3 = self.up3(d2)       # Upsample lần cuối

        # Tạo output mask
        out = self.out_conv(d3)
        # Đảm bảo output có cùng kích thước H, W với input ban đầu
        out = F.interpolate(out, size=(H, W), mode='bilinear', align_corners=False)
```

```
    return out
```

2. Xử lý dữ liệu (Data Handling)

- **Dữ liệu Gốc (taskliver_03):** Định dạng NIfTI (.nii), phổ biến trong lưu trữ ảnh y tế 3D. Giả định cấu trúc gồm thư mục train (chứa volumes và segmentations) và test**.**
- **Tiền xử lý (preprocess.py):** Mục đích là chuẩn bị dữ liệu tối ưu cho việc huấn luyện mô hình 2D, giảm tải I/O và tính toán trong quá trình train.

```
def preprocess_volume(volume_path, seg_path, output_dir, slices_per_volume=5, target_size=(256, 256)):

    # Load volume và segmentation từ file raw (.nii)
    volume_nib = nib.load(volume_path)
    seg_nib = nib.load(seg_path)
    volume_data = volume_nib.get_fdata().astype(np.float32)
    seg_data = seg_nib.get_fdata().astype(np.float32)

    # Chuẩn hóa volume: min-max scaling về [0,1]
    vol_min, vol_max = volume_data.min(), volume_data.max()
    if vol_max - vol_min > 0:
        volume_data = (volume_data - vol_min) / (vol_max - vol_min)

    # Chuyển segmentation về nhị phân (threshold 0.5) - Chỉ lấy gan
    seg_data = (seg_data > 0.5).astype(np.float32)

    # Xác định các lát cắt trung tâm cần lấy
    D = volume_data.shape[2]  # Giả sử volume_data có shape (H, W, D)
    mid_slice = D // 2
    half = slices_per_volume // 2
    start = max(0, mid_slice - half)
    end = min(D, mid_slice + half + 1)

    base_filename = os.path.splitext(os.path.basename(volume_path))[0]  # ví dụ: volume-01
    for slice_idx in range(start, end):
        # Lấy lát cắt 2D từ volume và segmentation
        image_2d = volume_data[:, :, slice_idx]
        mask_2d = seg_data[:, :, slice_idx]

        # Resize: dùng cv2.resize (nội suy tuyến tính cho image, nearest cho mask)
        image_resized = cv2.resize(image_2d, target_size, interpolation=cv2.INTER_LINEAR)
```

```
        mask_resized = cv2.resize(mask_2d, target_size,
interpolation=cv2.INTER_NEAREST)

        # Thêm kênh: chuyển từ (H, W) thành (1, H, W)
        image_resized = np.expand_dims(image_resized, axis=0)
        mask_resized = np.expand_dims(mask_resized, axis=0)

        # Kiểm tra dữ liệu (NaN, vô hạn, mask ngoài [0,1])
        if np.isnan(image_resized).any() or np.isinf(image_resized).any():
            print(f"WARNING: Invalid image data in {volume_path}, slice {slice_idx}.")
            continue # Bỏ qua lát cắt này nếu dữ liệu không hợp lệ
        if np.isnan(mask_resized).any() or np.isinf(mask_resized).any():
            print(f"WARNING: Invalid mask data in {seg_path}, slice {slice_idx}.")
            continue # Bỏ qua lát cắt này
        if mask_resized.min() < 0 or mask_resized.max() > 1:
            print(f"Warning: Mask values out of [0,1] in {volume_path}, slice {slice_idx}.
Clipping performed.")
            mask_resized = np.clip(mask_resized, 0, 1)
```

- **Normalization (Min-Max Scaling):** Đưa cường độ pixel về [0, 1]. Đơn giản và hiệu quả cho nhiều mạng nơ-ron. Tuy nhiên, cần lưu ý rằng nó nhạy cảm với các giá trị ngoại lai (outliers) trong ảnh CT (ví dụ: vật liệu cản quang cao). Các phương pháp khác như Z-score normalization hoặc windowing (giới hạn khoảng Hounsfield Units - HU) có thể được cân nhắc.
- **Mask Binarization:** Chuyển đổi segmentation mask (có thể có giá trị [0, 1, 2] trong bộ dữ liệu LiTS gốc, với 1 là gan, 2 là tổn thương) thành mask nhị phân [0, 1] (chỉ gan) bằng ngưỡng 0.5. Điều này đơn giản hóa bài toán thành phân vùng gan.
- **Slice Selection:** Chỉ lấy 5 lát cắt trung tâm (slices_per_volume=5). Đây là một cách đơn giản hóa để giảm số lượng dữ liệu 2D, tập trung vào vùng gan thường lớn nhất. Nhược điểm là có thể bỏ lỡ thông tin ở các lát cắt rìa và không đại diện đầy đủ cho toàn bộ gan 3D.
- **Resizing:** cv2.resize được dùng để đưa tất cả các lát cắt về kích thước chuẩn (256x256). interpolation=cv2.INTER_LINEAR (song tuyến tính)

phù hợp cho ảnh cường độ, làm mịn ảnh khi thay đổi kích thước. interpolation=cv2.INTER_NEAREST (láng giềng gần nhất) là *bắt buộc* cho mask nhị phân để đảm bảo giá trị mask vẫn là 0 hoặc 1 mà không tạo ra các giá trị trung gian.

- **Storage (.npz):** np.savez_compressed lưu ảnh và mask đã xử lý vào file .npz, giúp tiết kiệm dung lượng và tăng tốc độ đọc dữ liệu so với việc đọc và xử lý file .nii mỗi lần.

```
# Lưu file dưới dạng npz nén
        filename = f"{base_filename}_slice-{slice_idx}.npz"
        filepath = os.path.join(output_dir, filename)
        np.savez_compressed(filepath, image=image_resized, mask=mask_resized)
        # print(f"Saved {filepath}") # Bỏ comment nếu muốn xem tiến trình
```

- **Data Validation:** Các kiểm tra np.isnan, np.isinf, và mask.min()/max() được thêm vào để đảm bảo tính toàn vẹn của dữ liệu trước khi lưu, giúp tránh lỗi trong quá trình huấn luyện.

- **Dataset Loading (LiverDataset vs PreprocessedLiverDataset):**

```
# file: LiTSpreprocessed.py

from torch.utils.data import Dataset, DataLoader
import os
import numpy as np
import torch

# ... các import và class khác ...

class PreprocessedLiverDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        """
        data_dir: folder chứa các file .npz đã được preprocess.
        transform: (tùy chọn) transform để áp dụng augmentation.
        """
        self.data_dir = data_dir
        # Lấy danh sách các file .npz trong thư mục
        self.file_list = sorted([f for f in os.listdir(data_dir) if f.endswith(".npz")])
        self.transform = transform
```

```
  def __len__(self):
    # Trả về tổng số file .npz (mỗi file là một lát cắt)
    return len(self.file_list)

  def __getitem__(self, idx):
    filepath = os.path.join(self.data_dir, self.file_list[idx])
    # Load dữ liệu từ file .npz
    data = np.load(filepath)
    image = data['image']   # shape: (1, H, W)
    mask = data['mask']     # shape: (1, H, W)

    # Kiểm tra dữ liệu cơ bản (đã có kiểm tra kỹ hơn trong preprocess.py)
    if np.isnan(image).any() or np.isinf(image).any():
      raise ValueError(f"Invalid image data in file {filepath}.")
    if np.isnan(mask).any() or np.isinf(mask).any():
      raise ValueError(f"Invalid mask data in file {filepath}.")
    # Đảm bảo mask vẫn trong khoảng [0, 1] sau khi load
    if mask.min() < 0 or mask.max() > 1:
      # print(f"Warning: Mask values out of [0,1] in file {filepath}. Clipping
performed.")
      mask = np.clip(mask, 0, 1)

    # Áp dụng augmentation nếu transform được cung cấp
    if self.transform is not None:
      image, mask = self.transform(image, mask)

    # Chuyển thành tensor kiểu float
    image_tensor = torch.from_numpy(image).float()
    mask_tensor = torch.from_numpy(mask).float()
    return image_tensor, mask_tensor
```

- LiverDataset (dùng trong LiTS_full.py): Thực hiện tất cả các bước (đọc .nii, chuẩn hóa, chọn lát cắt, resize) trong hàm __getitem__. Ưu điểm: Không cần bước tiền xử lý riêng, linh hoạt thay đổi tham số (số lát cắt, kích thước resize) mà không cần chạy lại preprocess. Nhược điểm: Chậm hơn đáng kể trong quá trình huấn luyện do phải thực hiện nhiều thao tác I/O và tính toán cho mỗi sample.

- PreprocessedLiverDataset (dùng trong LiTSpreprocessed.py): Chỉ cần đọc file .npz đã được xử lý sẵn. Ưu điểm: Nhanh hơn nhiều trong quá trình huấn luyện. Nhược điểm: Cần chạy preprocess.py trước, kém linh

hoạt nếu muốn thay đổi tham số tiền xử lý. Lỗi try except trong __getitem__. tăng cường sự ổn định khi tải dữ liệu.

- **Data Augmentation (AugmentResizeTransform):** Thực hiện on-the-fly trong quá trình huấn luyện để tăng cường dữ liệu.

```
# file: LiTSpreprocessed.py (Tương tự trong LiTS_full.py)

import cv2
import random
import numpy as np

class AugmentResizeTransform:
    def __init__(self, size=(256, 256), flip_prob=0.5, rotation_range=10):
        """
        size: tuple (width, height) để resize đến.
        flip_prob: xác suất lật ngang.
        rotation_range: biên độ góc xoay ngẫu nhiên (-rotation_range, rotation_range).
        """
        self.size = size
        self.flip_prob = flip_prob
        self.rotation_range = rotation_range

    def __call__(self, image, mask):
        # image, mask có kích thước (1, H, W) -> Lấy dữ liệu 2D
        image_2d = image[0]
        mask_2d = mask[0]

        # Random horizontal flip
        if random.random() < self.flip_prob:
            image_2d = np.fliplr(image_2d)
            mask_2d = np.fliplr(mask_2d)

        # Random rotation
        angle = random.uniform(-self.rotation_range, self.rotation_range)
        (h, w) = image_2d.shape[:2]
        center = (w // 2, h // 2)
        # Tính ma trận xoay
        rot_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
        # Áp dụng phép xoay affine, dùng nội suy phù hợp
        image_2d = cv2.warpAffine(image_2d, rot_matrix, (w, h),
flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_CONSTANT,
borderValue=0) # Thêm borderMode
        mask_2d = cv2.warpAffine(mask_2d, rot_matrix, (w, h),
flags=cv2.INTER_NEAREST, borderMode=cv2.BORDER_CONSTANT,
borderValue=0) # Thêm borderMode
```

```
        # Resize về kích thước target
        image_resized = cv2.resize(image_2d, self.size,
interpolation=cv2.INTER_LINEAR)
        mask_resized = cv2.resize(mask_2d, self.size,
interpolation=cv2.INTER_NEAREST)

        # Thêm lại kênh: (1, H, W)
        image_resized = np.expand_dims(image_resized, axis=0)
        mask_resized = np.expand_dims(mask_resized, axis=0)

        return image_resized, mask_resized
```

- **Random Horizontal Flip (np.fliplr):** Giúp mô hình bất biến với hướng trái-phải.
- **Random Rotation (cv2.getRotationMatrix2D, cv2.warpAffine):** Giúp mô hình chống chịu được sự thay đổi nhỏ về góc nhìn hoặc vị trí của bệnh nhân. Góc xoay nhỏ (rotation_range=10) là phù hợp vì ảnh CT thường có định hướng tương đối chuẩn. Sử dụng flags=cv2.INTER_LINEAR cho ảnh và flags=cv2.INTER_NEAREST cho mask.
- Resize cuối cùng trong transform này đảm bảo output luôn có kích thước (256, 256), ngay cả khi ảnh gốc trong .npz (nếu dùng PreprocessedLiverDataset) có kích thước khác (mặc dù trong kịch bản này chúng đều là 256x256).

3. Huấn luyện (Training)

- **Framework:** PyTorch, một thư viện học sâu phổ biến và mạnh mẽ.
- **Train/Validation Split:** Tỷ lệ 80/20 là phổ biến. random_state=42 đảm bảo việc chia là nhất quán qua các lần chạy, giúp tái lập kết quả. Tập validation rất quan trọng để đánh giá khả năng tổng quát hóa của mô hình trên dữ liệu chưa từng thấy và để điều khiển các cơ chế như learning rate scheduling và early stopping.
- **Loss Function (Combined Loss):**
    - Dice Loss: Đo lường độ trùng khớp (overlap) giữa dự đoán và nhãn, rất phù hợp cho segmentation. Công thức: 1 - (2 * |P ∩ G| + smooth) / (|P| + |G| + smooth). Hệ số smooth=1e-5 ngăn chia cho 0 khi cả dự đoán và nhãn đều rỗng.

- BCEWithLogitsLoss: Là sự kết hợp của Sigmoid layer và Binary Cross Entropy loss. Nó ổn định hơn về mặt số học so với việc áp dụng Sigmoid rồi tính BCELoss riêng lẻ. Đo lường sự khác biệt ở cấp độ pixel.
- Kết hợp 50/50 (0.5 * dice + 0.5 * bce): Cân bằng giữa việc tối ưu hóa độ trùng khớp vùng và độ chính xác pixel, thường mang lại kết quả tốt hơn so với chỉ dùng một trong hai.

```
# file: LiTSpreprocessed.py (Tương tự trong LiTS_full.py)

import torch
import torch.nn as nn

def dice_loss(pred, target, smooth=1e-5):
    pred = torch.sigmoid(pred) # Chuyển logits thành xác suất [0, 1]
    # Tính toán intersection và union trên các chiều không gian (H, W)
    intersection = (pred * target).sum(dim=[2, 3])
    union = pred.sum(dim=[2, 3]) + target.sum(dim=[2, 3])
    # Công thức Dice coefficient
    dice_coef = (2. * intersection + smooth) / (union + smooth)
    # Đảm bảo giá trị không vượt quá 1 (có thể xảy ra do làm tròn số học)
    dice_coef = torch.clamp(dice_coef, max=1.0)
    # Dice loss = 1 - Dice coefficient
    dice = 1 - dice_coef
    # Trả về giá trị loss trung bình trên batch
    return dice.mean()

def bce_loss(pred, target):
    # squeeze(1) để loại bỏ chiều kênh (từ Bx1xHxW thành BxHxW)
    pred = pred.squeeze(1)
    target = target.squeeze(1)
    # Sử dụng hàm tích hợp sẵn của PyTorch để ổn định số học
    return nn.functional.binary_cross_entropy_with_logits(pred, target)

def combined_loss(pred, target):
    # Kết hợp 50% Dice loss và 50% BCE loss
    return 0.5 * dice_loss(pred, target) + 0.5 * bce_loss(pred, target)
```

- **Optimizer (Adam):** Thuật toán tối ưu hóa dựa trên gradient bậc nhất và bậc hai, hiệu quả và thường được dùng làm mặc định. lr=1e-4 là một learning rate khởi đầu phổ biến.

```
optimizer = optim.Adam(model.parameters(), lr=lr)
```

- **Learning Rate Scheduler (ReduceLROnPlateau):** Tự động giảm LR (factor=0.5) nếu validation loss không cải thiện (mode='min') trong patience=2 epochs. Giúp mô hình hội tụ tốt hơn ở giai đoạn sau của quá trình huấn luyện khi gần đạt tối ưu cục bộ.

```
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min',
patience=3, factor=0.5, verbose=True) # Tăng patience, bật verbose
```

- **Early Stopping:** Ngăn chặn overfitting và tiết kiệm tài nguyên bằng cách dừng huấn luyện nếu validation loss không cải thiện ít nhất min_delta=1e-4 trong patience=5 epochs liên tiếp. best_loss lưu lại giá trị loss tốt nhất trên tập validation.

```
# file: LiTSpreprocessed.py (Tương tự trong LiTS_full.py)

class EarlyStopping:
    def __init__(self, patience=5, min_delta=1e-4):
        self.patience = patience # Số epoch chờ đợi nếu không cải thiện
        self.min_delta = min_delta # Mức cải thiện tối thiểu được coi là có ý nghĩa
        self.best_loss = None # Lưu loss tốt nhất từng thấy
        self.counter = 0 # Bộ đếm số epoch không cải thiện liên tiếp

    def step(self, loss):
        # Lần đầu tiên gọi
        if self.best_loss is None:
            self.best_loss = loss
            return False # Chưa dừng

        # Nếu loss hiện tại cải thiện đáng kể so với best_loss
        elif self.best_loss - loss > self.min_delta:
            self.best_loss = loss # Cập nhật best_loss
            self.counter = 0 # Reset bộ đếm
            return False # Chưa dừng

        # Nếu không cải thiện hoặc cải thiện không đáng kể
        else:
            self.counter += 1 # Tăng bộ đếm
            # Nếu bộ đếm vượt ngưỡng patience
            if self.counter >= self.patience:
                print(f"Early stopping triggered after {self.counter} epochs without
improvement.")
                return True # Dừng huấn luyện
            return False # Chưa dừng
```

- **Evaluation Metrics:**
    - **Dice Coefficient:** (2 * |P ∩ G|) / (|P| + |G|). Thước đo độ trùng khớp chuẩn cho segmentation, giá trị từ 0 đến 1 (càng cao càng tốt).
    - **Pixel Accuracy:** (Số pixel dự đoán đúng) / (Tổng số pixel). Đo tỷ lệ pixel được phân loại chính xác. Có thể bị ảnh hưởng bởi sự mất cân bằng lớp (ví dụ: nếu background chiếm đa số, accuracy cao không đồng nghĩa với segmentation tốt).
    - Các hàm evaluate và evaluate_loss tính toán các chỉ số này trên toàn bộ dataloader (train hoặc validation).

```
# file: LiTSpreprocessed.py (Tương tự trong LiTS_full.py)

import numpy as np
import torch

def evaluate(model, dataloader, device):
    model.eval() # Đặt model ở chế độ evaluation
    dice_scores = []
    pixel_accuracies = []
    with torch.no_grad(): # Tắt tính toán gradient
        for images, masks in dataloader:
            images = images.to(device)
            masks = masks.to(device)
            outputs = model(images) # Dự đoán
            preds = torch.sigmoid(outputs) # Chuyển sang xác suất
            preds = (preds > 0.5).float() # Nhị phân hóa mask dự đoán

            # Tính Dice coefficient cho batch hiện tại
            intersection = (preds * masks).sum(dim=[2,3])
            union = preds.sum(dim=[2,3]) + masks.sum(dim=[2,3])
            dice = (2. * intersection + 1e-5) / (union + 1e-5) # Thêm smooth để tránh chia
cho 0
            dice_scores.append(dice.mean().item()) # Lưu Dice trung bình của batch

            # Tính Pixel Accuracy cho batch hiện tại
            correct = (preds == masks).float().sum() # Số pixel dự đoán đúng
            total = torch.numel(preds) # Tổng số pixel
            pixel_accuracy = correct / total
            pixel_accuracies.append(pixel_accuracy.item())
```

```
# Tính trung bình Dice và Accuracy trên toàn bộ dataloader
avg_dice = np.mean(dice_scores)
avg_accuracy = np.mean(pixel_accuracies)
return avg_dice, avg_accuracy
```

- **Monitoring:**
    - In ra console: Train Loss, Valid Loss, Gap = Valid Loss - Train Loss (dấu hiệu overfitting nếu dương và lớn), LR hiện tại, các metrics (Dice, Acc) trên cả train và valid set sau mỗi epoch.
    - Vẽ đồ thị (matplotlib): Trực quan hóa sự thay đổi của Train/Valid Loss, Train/Valid Dice, Train/Valid Accuracy qua các epoch. Các đồ thị này rất quan trọng để phân tích quá trình học, tốc độ hội tụ, và mức độ overfitting.
- **Hardware & Parallelism:**
    - device = torch.device("cuda" if torch.cuda.is_available() else "cpu"): Tự động sử dụng GPU nếu có, giúp tăng tốc đáng kể.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

    - num_workers trong DataLoader: Tăng số lượng tiến trình con để tải dữ liệu song song, giúp giảm thời gian chờ đợi của GPU. Giá trị nên được điều chỉnh dựa trên số lõi CPU (checkcpu.py) và khả năng hệ thống. num_workers=0 (trong LiTSpreprocessed_nb_version) nghĩa là tải dữ liệu trong tiến trình chính, có thể chậm hơn.

```
train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True, num_workers=num_workers_to_use, pin_memory=True if device.type == 'cuda' else False)
```

    - pin_memory=True: Khi dùng GPU, thiết lập này giúp tăng tốc độ truyền dữ liệu từ CPU RAM sang GPU VRAM.
- **Gradient Clipping (torch.nn.utils.clip_grad_norm_):** Giới hạn norm của gradient ở một ngưỡng (max_norm=1.0). Giúp ngăn chặn hiện tượng "exploding gradients" có thể xảy ra, đặc biệt trong các mạng sâu hoặc khi dùng RNN/Transformer, làm ổn định quá trình huấn luyện.

```
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
```

vòng lặp xử lý từng batch trong một epoch

```
import torch

def train_one_epoch(model, dataloader, optimizer, device):
    model.train() # Đặt model ở chế độ training
    total_loss = 0.0
    for batch_idx, (images, masks) in enumerate(dataloader):
        images = images.to(device)
        masks = masks.to(device)

        # Kiểm tra dữ liệu hợp lệ (NaN, Inf)
        if torch.isnan(images).any() or torch.isinf(images).any():
            print(f"WARNING: Invalid image tensor in batch {batch_idx}, skipping
batch.")
            continue
        if torch.isnan(masks).any() or torch.isinf(masks).any():
            print(f"WARNING: Invalid mask tensor in batch {batch_idx}, skipping
batch.")
            continue

        optimizer.zero_grad() # Reset gradients
        outputs = model(images) # Forward pass
        loss = combined_loss(outputs, masks) # Tính loss

        # Kiểm tra loss hợp lệ
        if torch.isnan(loss) or torch.isinf(loss):
            print(f"WARNING: NaN or Inf loss encountered in batch {batch_idx},
skipping update.")
            continue
        if loss.item() < 0:
            print(f"WARNING: Negative loss encountered in batch {batch_idx}! Loss:
{loss.item():.6f}")
            # Có thể vẫn tiếp tục hoặc bỏ qua batch này tùy chiến lược

        loss.backward() # Backward pass (tính gradients)

        # Gradient Clipping để tránh exploding gradients
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

        optimizer.step() # Cập nhật weights
        total_loss += loss.item() # Cộng dồn loss
```

```
    # Trả về loss trung bình trên toàn bộ epoch
    # Chia cho số batch thực sự đã xử lý (quan trọng nếu có bỏ qua batch)
    num_batches_processed = batch_idx + 1 # Giả sử không có batch nào bị bỏ qua hoàn toàn
    if num_batches_processed > 0 :
        return total_loss / num_batches_processed
    else:
        return 0.0 # Trường hợp không xử lý được batch nào
```

III. THỰC NGHIỆM VÀ KẾT QUẢ

1. Cài đặt (Set up)

- Cần cài đặt đầy đủ các thư viện trong requirements.txt.
- Duy trì cấu trúc thư mục dự án là bắt buộc để các đường dẫn hoạt động chính xác.
- Số lõi CPU (checkcpu.py) nên được kiểm tra để tối ưu num_workers.

```
# file: checkcpu.py
import os num_cores = os.cpu_count()
print(f"Số lõi CPU: {num_cores}")
```

2. Quy trình Thực thi (Workflow)

- **Chuẩn bị dữ liệu**: Đặt dữ liệu .nii gốc vào taskliver_03/train/volumes và taskliver_03/train/segmentations. Đặt dữ liệu test vào taskliver_03/test.
- **(Tùy chọn, cho LiTSpreprocessed.py)** Chạy python preprocess.py. Quá trình này sẽ tạo các file .npz trong thư mục preprocess_data.
- **Huấn luyện:**

```
  for epoch in range(num_epochs):
      # Huấn luyện 1 epoch
      train_loss = train_one_epoch(model, train_loader, optimizer, device)
      # Đánh giá loss trên tập validation
      valid_loss = evaluate_loss(model, valid_loader, device)

      # Cập nhật learning rate dựa trên validation loss
      scheduler.step(valid_loss)
      current_lr = optimizer.param_groups[0]['lr']

      # Đánh giá metrics (Dice, Acc) trên cả train và validation set
      # Lưu ý: Tính metrics trên train set có thể tốn thời gian, cân nhắc chỉ tính trên validation
      avg_dice_train, avg_acc_train = evaluate(model, train_loader, device) # Có thể bỏ qua nếu muốn nhanh hơn
      avg_dice_valid, avg_acc_valid = evaluate(model, valid_loader, device)

      # Lưu lại các giá trị
      train_losses.append(train_loss)
      valid_losses.append(valid_loss)
      train_dice_list.append(avg_dice_train)
      train_acc_list.append(avg_acc_train)
      valid_dice_list.append(avg_dice_valid)
      valid_acc_list.append(avg_acc_valid)
```

```
        # Tính hiệu số loss để theo dõi overfitting
        loss_gap = valid_loss - train_loss

        print(f"Epoch [{epoch+1}/{num_epochs}], LR: {current_lr:.6f}")
        print(f"  Loss - Train: {train_loss:.4f}, Valid: {valid_loss:.4f}, Gap:
{loss_gap:.4f}")
        # Bỏ print train metrics nếu không tính
        # print(f"  Metrics - Train Dice: {avg_dice_train:.4f}, Train Acc:
{avg_acc_train:.4f}")
        print(f"  Metrics - Valid Dice: {avg_dice_valid:.4f}, Valid Acc:
{avg_acc_valid:.4f}")



        # Lưu model nếu validation Dice tốt hơn
        if avg_dice_valid > best_valid_dice:
            print(f"  Validation Dice improved ({best_valid_dice:.4f} -->
{avg_dice_valid:.4f}). Saving model...")
            best_valid_dice = avg_dice_valid
            os.makedirs("weights", exist_ok=True)
            save_path = "weights/pvt_liver_seg_preprocess_best.pth" # Lưu model tốt
nhất
            torch.save(model.state_dict(), save_path)

        # Kiểm tra điều kiện Early Stopping dựa trên validation loss
        if early_stopper.step(valid_loss):
            break # Dừng vòng lặp huấn luyện

    print("Training finished.")

    # Lưu mô hình cuối cùng (sau khi kết thúc hoặc early stopping)
    final_save_path = "weights/pvt_liver_seg_preprocess_final.pth"
    torch.save(model.state_dict(), final_save_path)
    print(f"Final model saved at {final_save_path}")
    print(f"Best validation Dice achieved: {best_valid_dice:.4f} (model saved at
weights/pvt_liver_seg_preprocess_best.pth)")
```

- Chạy python LiTSpreprocessed.py để huấn luyện trên dữ liệu .npz. Model cuối cùng sẽ được lưu tại weights/pvt_liver_seg_preprocess.pth.
- HOẶC Chạy python LiTS_full.py để huấn luyện trực tiếp từ dữ liệu .nii. Model cuối cùng sẽ được lưu tại weights/pvt_liver_seg_final.pth. (Lưu ý tên file khác nhau).

- Theo dõi output trên console và các file đồ thị (loss_plot.png, dice_plot.png, pixel_acc_plot.png) được tạo ra.

- **Dự đoán:** Chạy python inference.py. Script này sẽ:
  - Load model từ weights/pvt_liver_seg_preprocess.pth (Lưu ý: hiện tại nó chỉ load model này, cần sửa đổi nếu muốn dùng model từ LiTS_full.py).

```
model.load_state_dict(torch.load(model_path, map_location=device))
model.eval()
```

  - Xử lý từng file .nii trong taskliver_03/test.
  - Dự đoán mask cho lát cắt trung tâm.
  - Lưu ảnh gốc kèm mask dự đoán vào thư mục plots dưới dạng file .png.

```
# 2. Folder chứa dữ liệu test (.nii)
test_root = "taskliver_03/test"
if not os.path.isdir(test_root):
    print(f"Error: Test data directory not found at {test_root}")
    return

# Lấy danh sách file volume test
volume_files = sorted([
    f for f in os.listdir(test_root)
    if f.startswith("test-volume-") and f.endswith(".nii")
])

if not volume_files:
    print(f"No test volume files found in {test_root}")
    return

# Tạo thư mục chứa kết quả plot nếu chưa có
output_plot_dir = "plots"
os.makedirs(output_plot_dir, exist_ok=True)

print(f"Found {len(volume_files)} test volumes. Starting inference...")

for vol_file in volume_files:
    print(f"  Processing {vol_file}...")
    volume_path = os.path.join(test_root, vol_file)

    try:
        # Load volume NIfTI
        volume_nib = nib.load(volume_path)
```

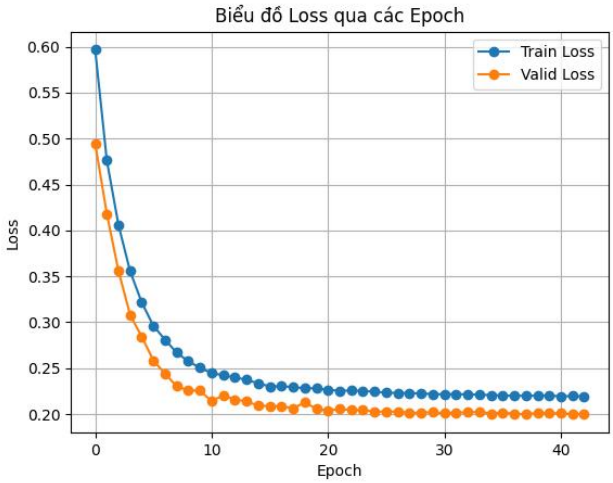
```
        volume_data = volume_nib.get_fdata().astype(np.float32)

        # Lấy lát cắt trung tâm (ví dụ)
        # Lưu ý: Có thể cần chiến lược khác để xử lý toàn bộ volume
        if volume_data.ndim < 3 or volume_data.shape[2] == 0:
            print(f"    Warning: Volume {vol_file} has invalid dimensions or 0 slices.
Skipping.")
            continue
        mid_slice_idx = volume_data.shape[2] // 2
        image_2d = volume_data[:, :, mid_slice_idx]

        # Tiền xử lý ảnh 2D giống như lúc huấn luyện:
        # 1. Chuẩn hóa về [0, 1] (Quan trọng nếu model được huấn luyện với dữ liệu
chuẩn hóa)
        img_min, img_max = image_2d.min(), image_2d.max()
        if img_max - img_min > 0:
            image_normalized = (image_2d - img_min) / (img_max - img_min)
        else:
            image_normalized = np.zeros_like(image_2d)

        # 2. Resize về (256, 256)
        image_resized = cv2.resize(image_normalized, (256, 256),
interpolation=cv2.INTER_LINEAR)

        # 3. Chuyển thành tensor, thêm chiều batch và kênh
        # Input shape: (H, W) -> (1, 1, H, W)
        image_tensor =
torch.from_numpy(image_resized).unsqueeze(0).unsqueeze(0).to(device)

        # Thực hiện dự đoán với model
        with torch.no_grad():
            pred_logits = model(image_tensor)
            pred_sigmoid = torch.sigmoid(pred_logits) # Chuyển logits thành xác suất
            # Nhị phân hóa mask dự đoán với ngưỡng 0.5
            pred_mask = (pred_sigmoid > 0.5).float().cpu().numpy()[0, 0] # Lấy kết quả
2D numpy array

        # Trực quan hóa kết quả
        plt.figure(figsize=(12, 6)) # Kích thước figure

        # Ảnh gốc (đã resize và chuẩn hóa)
        plt.subplot(1, 2, 1)
        plt.imshow(image_resized, cmap='gray', aspect='equal') # aspect='equal' để tỉ
lệ đúng
        plt.title(f"Input Slice {mid_slice_idx}")
        plt.axis("off") # Tắt trục tọa độ
```

```
            # Ảnh gốc phủ mask dự đoán
            plt.subplot(1, 2, 2)
            plt.imshow(image_resized, cmap='gray', aspect='equal')
            # Dùng 'jet' colormap và alpha=0.5 để mask trong suốt
            plt.imshow(pred_mask, alpha=0.5, cmap='jet')
            plt.title("Predicted Liver Mask")
            plt.axis("off")

            # Lưu hình ảnh kết quả
            save_filename =
f"pred_{os.path.splitext(vol_file)[0]}_slice{mid_slice_idx}.png"
            save_path = os.path.join(output_plot_dir, save_filename)
            plt.savefig(save_path, bbox_inches='tight', dpi=150) # Lưu với độ phân giải
cao hơn
            plt.close() # Đóng figure hiện tại để giải phóng bộ nhớ

            # print(f"    Saved prediction plot to {save_path}")

        except Exception as e:
            print(f"    Error processing {vol_file}: {e}")
            # Đảm bảo đóng plot nếu có lỗi xảy ra giữa chừng
            plt.close()

    print("Inference finished.")
```

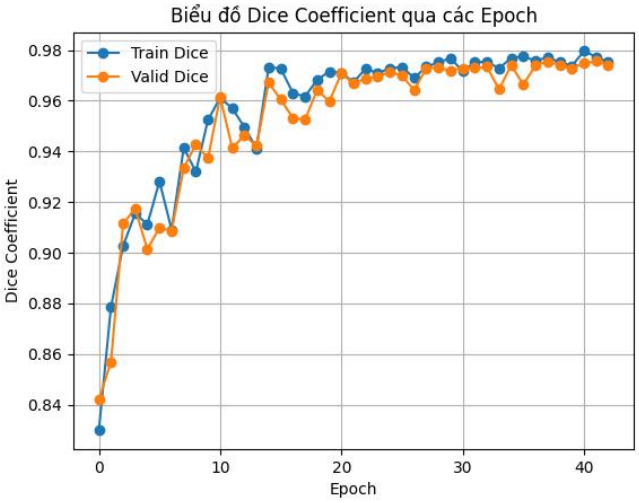
3. **Đánh giá (Evaluation)**
   - **Định lượng (Trong quá trình huấn luyện):** Dice coefficient và Pixel Accuracy được tính toán và báo cáo sau mỗi epoch trên tập train và validation. Các đồ thị cung cấp cái nhìn trực quan về xu hướng hiệu năng.
   - **Định tính (Sau huấn luyện):** inference.py tạo ra các hình ảnh trực quan hóa kết quả dự đoán trên lát cắt trung tâm của dữ liệu test, cho phép đánh giá bằng mắt thường chất lượng của segmentation (ví dụ: độ mịn của đường viền, có bỏ sót vùng nào không, có dự đoán nhầm sang cơ quan khác không).
   - **Hạn chế:** Chưa có đánh giá định lượng tự động trên toàn bộ tập test sau khi huấn luyện xong.
4. **Phân tích (Analysis)**
   - **PVT Encoder:** Việc sử dụng PVT cho thấy tiềm năng của kiến trúc Transformer trong việc nắm bắt các mối quan hệ không gian phức tạp và ngữ

cảnh rộng trong ảnh y tế, nhờ vào cơ chế self-attention và các tầng đặc trưng đa tỷ lệ.

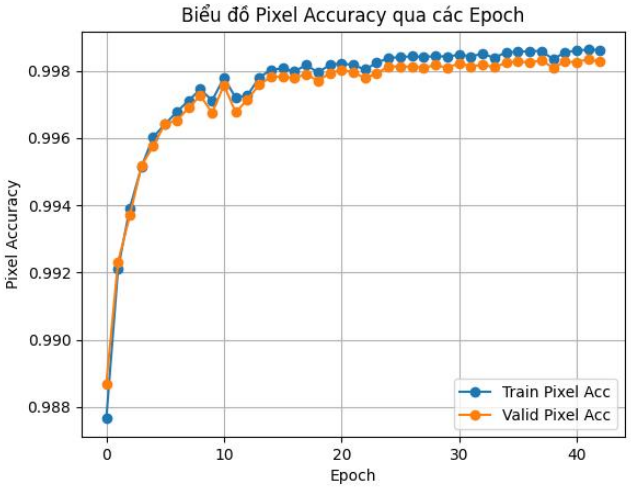
- **U-Net Decoder & Skip Connections:** Sự kết hợp này rất quan trọng để tạo ra segmentation mask chi tiết. Phân tích đồ thị có thể cho thấy sự hội tụ nhanh chóng nhờ pretrained weights và hiệu quả của các kỹ thuật regularization (scheduler, early stopping).
- **Preprocessed vs. Full Data:** Dự kiến LiTSpreprocessed.py sẽ có thời gian huấn luyện mỗi epoch ngắn hơn đáng kể do giảm tải I/O và tiền xử lý. Hiệu năng cuối cùng (Dice, Acc) có thể tương đương nếu các tham số tiền xử lý giống nhau, nhưng LiTS_full.py có thể linh hoạt hơn nếu cần thử nghiệm các kiểu tiền xử lý khác nhau mà không cần tạo lại file .npz.
- **Hyperparameters:** batch_size=2 là khá nhỏ, có thể do giới hạn bộ nhớ GPU. Điều này có thể làm cho gradient bị nhiễu hơn, nhưng việc sử dụng BatchNorm và tối ưu hóa Adam giúp giảm thiểu vấn đề này. lr=1e-4 và các tham số scheduler/early stopping là các giá trị khởi đầu hợp lý cần được tinh chỉnh thêm.
- **Đồ thị:** Một quá trình huấn luyện tốt sẽ thấy cả train/valid loss giảm, train/valid metrics tăng. Khoảng cách (Gap) giữa train và valid metrics nên nhỏ và ổn định. Nếu Gap tăng lớn, đặc biệt là khi valid loss bắt đầu tăng trở lại, đó là dấu hiệu rõ ràng của overfitting, và early stopping sẽ có ích.



Hình 3.1 Biểu đồ Loss qua các Epoch



Hình 3.2 Biểu đồ Dice Coefficient qua các Epoch



Hình 3.3 Biểu đồ Pixel Accuracy qua các Epoch

IV. THẢO LUẬN

1. Điểm mạnh

- **Kiến trúc Hiện đại:** Sử dụng PVTv2, một backbone Transformer hiệu quả.
- **Kỹ thuật Huấn luyện Tốt:** Áp dụng loss kết hợp, scheduler, early stopping, gradient clipping.
- **Tính linh hoạt:** Cung cấp hai workflow huấn luyện (dữ liệu tiền xử lý và dữ liệu gốc).
- **Mã nguồn có cấu trúc:** Tách biệt rõ ràng các thành phần, dễ hiểu và bảo trì.
- **Tái lập:** Sử dụng random_state cho split dữ liệu.

2. Hạn chế và Hướng phát triển

- **Hạn chế 2D:** Xử lý lát cắt độc lập bỏ lỡ thông tin 3D quan trọng cho sự liền mạch và chính xác về mặt giải phẫu. **Hướng phát triển:** Xây dựng mô hình 3D (ví dụ: 3D U-Net, V-Net) hoặc kiến trúc 2.5D (sử dụng nhiều lát cắt liền kề làm đầu vào). Tích hợp PVT vào kiến trúc 3D.
- **Inference Hạn chế:** Chỉ dự đoán và visualize lát cắt trung tâm của tập test. **Hướng phát triển:** Viết lại inference.py để lặp qua tất cả các lát cắt liên quan của một volume, thực hiện dự đoán, và ghép nối lại thành một segmentation volume 3D (.nii).
- **Thiếu Đánh giá Test Định lượng:** Không tự động tính toán metrics trên tập test. **Hướng phát triển:** Bổ sung vào inference.py (sau khi đã có dự đoán 3D) việc tải ground truth segmentation của tập test (nếu có) và tính toán các chỉ số 3D phổ biến như Volumetric Dice, Average Symmetric Surface Distance (ASSD), Hausdorff Distance (95%).
- **Tối ưu Hyperparameter:** Các giá trị hiện tại (LR, batch size, patience, factor,...) là các giá trị khởi đầu. **Hướng phát triển:** Sử dụng các kỹ thuật tìm kiếm siêu tham số tự động như Optuna, Ray Tune để tìm bộ tham số tối ưu hơn.
- **Post-processing:** Kết quả dự đoán thô có thể chứa các lỗ nhỏ hoặc đường viền không mịn. **Hướng phát triển:** Áp dụng các kỹ thuật hậu xử lý như lấp lỗ (hole filling), chọn thành phần liên thông lớn nhất, hoặc sử dụng Conditional Random Fields (CRFs) để làm mịn và cải thiện tính nhất quán của mask.

- **Mở rộng Bài toán:** Mô hình hiện tại chỉ phân vùng gan. **Hướng phát triển:** Mở rộng để phân vùng cả tổn thương gan (liver lesion segmentation), đây là bài toán phức tạp hơn và có giá trị lâm sàng cao. Có thể dùng kiến trúc multi-class hoặc hai giai đoạn (phân vùng gan -> phân vùng tổn thương trong gan).
- **So sánh Mô hình:** Chưa so sánh với các phương pháp khác. **Hướng phát triển:** Huấn luyện và đánh giá các backbone CNN phổ biến (ResNet, EfficientNet) hoặc các kiến trúc segmentation khác (DeepLabV3+, UNet++) trên cùng bộ dữ liệu và quy trình để so sánh hiệu năng.

V. KẾT LUẬN

Dự án đã thành công trong việc xây dựng và đánh giá một pipeline học sâu sử dụng Pyramid Vision Transformer (PVT) kết hợp với kiến trúc U-Net để phân vùng gan 2D trên ảnh CT. Mô hình thể hiện khả năng học các đặc trưng đa tỷ lệ và phục hồi chi tiết không gian hiệu quả. Việc triển khai hai quy trình huấn luyện song song, áp dụng các kỹ thuật huấn luyện tiên tiến, và cung cấp mã nguồn có cấu trúc là những đóng góp quan trọng. Mặc dù mô hình hiện tại giới hạn ở xử lý 2D và cần cải thiện phần đánh giá trên tập test, dự án này cung cấp một nền tảng vững chắc và đầy hứa hẹn. Các kết quả và phân tích mở ra nhiều hướng phát triển tiềm năng, bao gồm việc chuyển sang mô hình 3D, tối ưu hóa siêu tham số, và mở rộng sang các bài toán phân vùng phức tạp hơn trong lĩnh vực chẩn đoán hình ảnh y tế.