Trường Đại học Khoa học tự nhiên
Đại học Quốc gia Thành phố Hồ Chí Minh
Khoa Công nghệ thông tin

AMAS-IT

Báo cáo lab project 2 - Color processing

Lớp: 22CLC04

Thực hiện:
Ngũ Kiệt Hùng - 22127134

Giảng viên hướng dẫn:
Thầy Trần Hà Sơn
Thầy Nguyễn Ngọc Toàn

Tháng 7, 2024

Mục lục

1 Giới thiệu chủ đề

Việc xử lý ảnh thông qua phương pháp biến đổi ma trận biểu diễn điểm ảnh đã và vẫn đang là một trong những vấn đề trọng điểm trong cả các lĩnh vực ứng dụng thực tiễn và nghiên cứu khoa học. Đối với các tác vụ thường ngày, xử lý ảnh thường được áp dụng để chỉnh sửa hoặc tạo ra các sản phẩm nghệ thuật. Mặt khác, trong nghiên cứu khoa học, ta có thể thấy sự gia tăng nhanh chóng của các công nghệ, thuật toán xử lý ảnh trong thời gian gần đây do sự phát triển của phần cứng, nghiên cứu về giải thuật trí tuệ nhân tạo. Các tác vụ liên quan đến khoa học dữ liệu đòi hỏi ta phải có một lượng dữ liệu đầu vào tương đối lớn để huấn luyện các mô hình, giải quyết các bài toán tối ưu. Việc chuẩn bị tập dữ liệu đủ để sấp xỉ các nghiệm gần đúng của các bài toán trên rất khó, đặc biệt là khi không gian dữ liệu là các bức ảnh được biểu diễn dưới dạng ma trận 2, 3 chiều.

Nhằm giải quyết bài toán trên, giới nghiên cứu đã đưa ra nhiều giải pháp. Một trong những cách thức đơn giản và hiệu quả nhất, chính là chỉnh sửa các tập dữ liệu đã có sẵn thông qua một số phép ánh xạ điểm ảnh, như thay đổi đổi cường độ chỉ số các dữ liệu, quay các thành phần của điểm dữ liệu quanh một trục, masking, ... Vì thế đòi hỏi ta phải có các chương trình tự động hoá những quy trình chuẩn bị tập dữ liệu lớn.

Bài thực hành này có vai trò cài đặt cũng như áp dụng các kỹ thuật lập trình, tính toán ma trận nhằm tối ưu hóa các phép toán trên ma trận điểm ảnh thông qua chỉ một vài tham số cơ bản cho mỗi chức năng. Đồng thời, bài thực hành sẽ khám phá một số các phương pháp, thuật toán đơn giản nhằm tối ưu hóa tốc độ và độ chính xác của kết quả.

2 Ý tưởng tiếp cận vấn đề

Hầu hết các chức năng cơ bản của xử lý ảnh đã được khám phá và nghiên cứu. Các ứng dụng thông dụng mà ta sử dụng trên điện thoại thông minh như các ứng dụng mạng xã hội, Adobe Suite, ... đều có khả năng xử lý ảnh với đa dạng các chức năng. Bài toán này chỉ tập trung vào việc tự động hoá các chức năng ấy thông qua một số thủ tục ánh xạ tuyến tính ma trận đầu vào bằng thư viện NumPy.

Trước hết, nhằm phục vụ cho việc hiểu rõ cũng như áp dụng các phương pháp tối ưu cho vấn đề, bài thực hành sẽ đi sơ qua về một số các cơ sở của các thuật toán.

2.1 Cơ sở, phân tích toán

Khi nói đến bài toán xử lý điểm ảnh, một điểm dữ liệu, tức một bức ảnh, thông thường sẽ là một ma trận 2^+ chiều, với 2 chiều đầu tiên thể hiện vị trí của các điểm ảnh trong bức ảnh, các chiều còn lại thường được sử dụng để thể hiện các thành phần của điểm ảnh, VD: với hệ màu RGB, chiều còn lại sẽ là 1 vector $\mathbb{R}^3$ chứa các thành phần đỏ, xanh lá, xanh dương tạo thành điểm ảnh (pixel) đó. Bài thực hành này sẽ chủ yếu làm việc trên ma trận điểm ảnh $\mathbb{R}^{\mathrm{W}\times\mathrm{H}\times 3}$.

Một trong những đặc tính được sử dụng nhiều nhất chính là tính phân phối của các kênh màu. Thông thường, ta, con người, rất dễ nhận thấy sự khác biệt hoặc không bình thường trong các bức ảnh. Một trong những nguyên do lớn nhất có thể đề cập đến chính vì sự phân bố không đồng đều của các màu sắc. Vì thế, bài thực hành này sẽ tập trung nhiều vào việc khai thác các thông tin ta có về mật độ phân của cường độ sáng của các điểm ảnh. Cụ thể hơn, ta mong muốn có thể tìm được $\mathrm{CDF}_{in}(R,G,B)$ với R, G, B là tập các cường độ các kênh màu có trong bức ảnh.

Tuỳ vào mục đích giải quyết của bài toán mà hàm tích luỹ mật độ CDF_{in} có thể được xác định khác nhau. Trong bài toán này, hàm mật độ chính mà ta sẽ quan tâm đến là $\mathrm{prob}\{\frac{r_i+g_i+b_i}{3} \le n_i\}$, với n_i, $0 \le i \le L$ thể hiện số điểm ảnh có cường độ i và L là cường độ sáng tối đa.

2.2 Xử lí nâng độ sáng cho ảnh (brightness correction)

Có thể đoán, việc tăng độ sáng cho một bức ảnh chỉ đơn giản là nhân các thành phần của mỗi điển ảnh với một trọng số nào đó, lưu ý rằng có thể các thành phần sau khi nhân có thể vượt quá giá trị tối đa của kênh màu, vì thế ta phải chặn tại giá trị tối đa.

$$rgb_{new} = \alpha \times rgb_{old}, \quad \alpha \in \mathbb{R}$$

Tất nhiên, một số điểm ảnh sẽ có thành phần hội tụ về các điểm chặn trên/dưới của miền giá trị thành phần trước những thành phần khác, khiến tính tương đối về tỉ lệ giữa các thành phần của mỗi điểm ảnh bị thay đổi, dẫn đến mất màu sắc của nó. Vấn đề này được nói rõ hơn ở phần xử lý độ tương phản.

2.3 Xử lý tăng độ tương phản cho ảnh (contrast correction)

Đối với bài toán tăng độ tương phản, có nhiều cách ta có thể tiếp cận. Một phương pháp đơn giản là xét tỉ lệ giữa các vùng sáng nhất và các vùng tối nhất, từ đó có thể trả về một tỉ lệ giữa 2 thành phần min, max của bức ảnh. Bằng cách "kéo giãn" bức ảnh qua việc nội suy cường độ sáng của mỗi điểm ảnh với tỉ lệ vừa nhận được phía trên, ta có thể giảm cường độ sáng của mỗi điểm ảnh tương đối so với 2 điểm min max của bức ảnh, khiến khác biệt của chúng trở nên nhỏ hơn, i.e. giảm độ tương phải, hoặc ngược lại.

Tuy nhiên, việc scale các điểm dữ liệu một cách "ngây ngô" như vậy rất dễ dẫn đến tình trạng mất đi tính cân bằng trong phân bố của kênh màu. Bằng chứng rõ nhất chính là Hue của bức ảnh sẽ thay đổi nếu ta tăng hoặc giảm độ tương phản quá mức, Hoặc sự tăng/giảm quá đà của cường độ điểm ảnh trong ảnh trắng đen.

Một trong các phương pháp phổ biến nhằm giải quyết vấn đề này chính là thông qua phương pháp cân bằng histogram phân bố màu [1] của các kênh màu và chuyển cơ sở hệ màu nhằm bảo toàn các thành phần màu của điểm ảnh (vì ta chỉ quan tâm đến luminance, độ sáng của điểm ảnh). Đây là phương pháp chính mà bài thực hành sẽ thực hiện, vì độ tin cậy trong chất lượng ảnh cũng như dễ dàng tính toán trong dạng ma trận. Từ các điểm ảnh, ta có thể xây dựng hàm phân phối tích luỹ

$$\mathrm{CDF}_{in}(r_i, g_i, b_i) = \mathrm{prob}\{\mathrm{r} + \mathrm{g} + \mathrm{b} \leq 3n_i\} = \sum_{\mathrm{r+g+b} \leq 3\mathrm{n_i}} p_{in}(r, g, b)$$

Với r_i, g_i, b_i lần lượt là số lượng điểm ảnh tại kênh r, g, b có cường độ i, $n_i = \frac{r_i + g_i + b_i}{3}$

Từ đây, ta mong muốn một ma trận điểm ảnh có cùng số lượng các kênh màu như ảnh hiện tại, tính trên thang đo cường độ sáng (RGB max 255), tức $CDF_{out}(r_i, g_i, b_i) = CDF_{in}(r_i, g_i, b_i)$ $0 \leq r_i, g_i, b_i \leq L$. Vì ta mong muốn histogram của ma trận ảnh output sẽ được giãn đều nhất có thể, nên $CDF_{out}(r_i, g_i, b_i) = n_{out}/L$. Vậy ta có thể thu được cường độ sáng của ma trận ảnh đầu ra: $n_{out} = \mathrm{L} \times \mathrm{CDF}_{\mathrm{in}}(r_i, g_i, b_i)$. Với giá trị cường độ mới của mỗi điểm ảnh, ta có thể sử dụng phương pháp biến đổi luminance shifting [2] bảo tồn màu chủ đạo của điểm ảnh.

2.4 Xử lý lật ảnh (flipping)

Việc lật ảnh cũng tương đối đơn giản, xét ma trận điểm ảnh A, nếu ta cần lật ảnh trên trục hoành (X-axis), ta chỉ việc hoán đổi vị trí của các điểm ảnh qua trục hoành đó, i.e. $a_{i,j} = a_{W-i,j}$, $1 \leq i \leq W$, $1 \leq j \leq H$. Tương tự với trục tung (Y-axis).

2.5 Xử lý chuyển đổi ảnh về grayscale/sepia

Ở mỗi điểm ảnh, ta có thể xem cường độ sáng của nó theo 3 cách: thành phần có lượng lớn nhất hoặc trung bình cộng của các thành phần hoặc tích theo trọng số của từng thành phần. Đối với một bài thực hành này, phương pháp tích theo trọng số vector biến đổi TU-R 601-2 luma được sử dụng. Từ cường độ sáng này, ta có thể điều chỉnh các điểm ảnh sao cho các thành phần đều có lượng bằng nhau, bằng với chỉ số này:

$$r_{new}(x,y) = g_{new}(x,y) = \mathrm{b}_{\mathrm{new}}(\mathrm{x},\mathrm{y}) = \mathrm{t}_1 \times r_{old}(x,y) + t_2 \times g_{old}(x,y) + t_3 \times \mathrm{b}_{\mathrm{old}}(\mathrm{x},\mathrm{y}),$$

$$1 \leq x \leq W,\ 1 \leq y \leq H,\ T = [t1 = 0.393 \quad \mathrm{t2} = 0.769 \quad \mathrm{t3} = 0.189]$$

Tương tự, biến đổi sepia là tích của từng thành phần của điểm ảnh với ma trận biến đổi sepia. Tuy nhiên, vì sepia không phải thuần trắng đen, thế nên từng thành phần của điểm ảnh phải biến đổi phù hợp.

$$\mathrm{rgb}_{\mathrm{new}} = \mathrm{Sep} \times \mathrm{rgb}_{\mathrm{old}}$$

$$Sep = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix}$$

2.6 Xử lý làm mờ và làm sắc nét ảnh

Việc cung cấp cho các mô hình học máy những dữ liệu đầu vào đáng tin cậy và dễ dàng nhận diện các features là một bài toán đã được nghiên cứu lâu dài trong khoa học dữ liệu,

vì quá trình dự đoán phụ thuộc rất nhiều vào các trọng số huấn luyện, mà nếu bị ảnh hưởng bởi nhiễu, có thể dễ dàng dẫn đến hiện tượng "ảo tưởng" của các mô hình.

Bài toán tối ưu mô hình dự đoán dữ liệu đầu vào là ảnh cũng không ngoại lệ. Việc nâng các điểm nổi bật của ảnh lên được thực hiện thông qua việc tách các features ấy ra thông qua một bộ lọc cho những điểm ảnh có sự thay đổi đáng kể về kì vọng giá trị điểm ảnh (tức các cạnh, các chi tiết phân biệt độ sâu, góc, …), và sau đó thực hiện củng cố các features trên ảnh gốc thông qua phép luminance shifting như đã nói ở phần tương phản.

Bài toán tách features có nhiều hướng để tiếp cận. Bài thực hành này sẽ chủ yếu khám phá việc phương pháp unsharp masking (UM) [3]. Mà bước đầu tiên trong đó sẽ là tìm ma trận điểm ảnh được làm "trơn" thêm thông qua giải thuật Gaussian filtering [4].

Việc sử dụng Gaussian Filtering thông thường đòi hỏi ta phải tính tích chập (convolution) của một ma trận kernel có dạng $\mathbb{R}^{M\times M}$ cho mỗi điểm ảnh; việc này tốn rất nhiều chi phí tính toán. Thay vào đó, bài thực hành áp dụng [5] nhằm sấp xỉ Gaussian Smoothing mà chi phí tính toán được tối ưu về các tác vụ truy xuất dữ liệu.

Một cách khái quát, với một ma trận điểm ảnh 2 chiều, ta có thể tính một ma trận tổng giá trị tích luỹ (Summed Area Table or Integral Image) bằng việc tính tổng tích luỹ theo hàng của ma trận điểm ảnh đầu vào, sau đó tính tổng tích luỹ bằng cột. Ta sẽ có được ma trận $S^{W\times H\times C}$, với mỗi toạ độ là tổng tích luỹ các giá trị điểm ảnh phía trên và bên trái của toạ độ đang xét. Với ma trận này, cho trước 4 toạ độ tạo thành hình vuông a, b, c, d, ta có thể tính tổng giá trị các điểm ảnh trong hình chữ nhật đó:

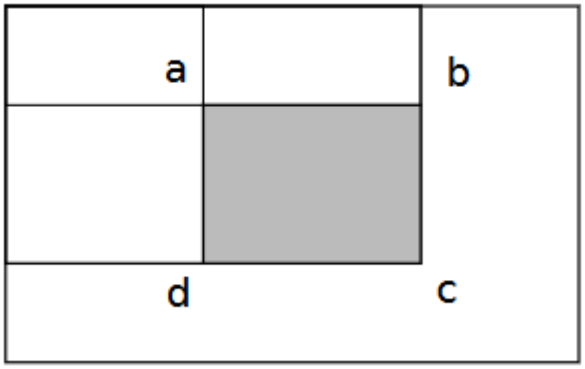

Figure 1 Tính tổng giá trị các điểm ảnh (Kovesi, P.)

$$\sum abcd = \mathrm{S}(x_c, y_c) - \mathrm{S}(x_b, y_b) - \mathrm{S}(x_d, y_d) + \mathrm{S}(x_a, y_a) \quad \text{(eq0)}$$

Nếu ta lại lấy giá trị tổng này chia cho số lượng điểm ảnh nằm trong hình chữ nhật đó, ta sẽ có trung bình các điểm ảnh, nói cách khác, ta có một bộ lọc theo trung bình cộng với điểm trung bình nằm giữa hình chữ nhật/vuông đó.

Trở lại với bài toán làm trơn điểm ảnh, nếu cho ta trước σ của hàm Gaussian, và số lần áp dụng bộ lọc trung bình cộng như trên n_{pass}, ta có thể sấp xỉ được độ dài của kernel lý tưởng (eq1)

$$w_{ideal} = \sqrt{\frac{12\sigma^2}{n_{pass}} + 1}$$

Nhưng vì w_{ideal} thường là số thực, ta có thể dùng 2 bộ lọc trung bình khác nhau, với độ dài là số lẻ và $w_l < w_{ideal} \leq w_u$ sau đó thực hiện filter trên 2 kernel width w_l m lần, width w_u $n - m$ lần để được kết quả gần sấp xỉ với σ. (eq2)

$$\lfloor m \rfloor = \frac{12\sigma^2 - n_{pass}w_l^2 - 4n_{pass}w_l - 3n_{pass}}{-4w_l - 4}$$

Từ đây, ta đã có giải thuật làm mờ một bức ảnh, và cũng từ cách thức đã nói đến phía trên của việc nâng độ sắc nét của ảnh, ta có công thức: (eq3)

$$\text{sharpened} = \text{original} + (\text{original} - \text{blurred}) \times \text{amount} \qquad [6]$$

2.7 Cắt ảnh theo kích thước

Việc cắt ảnh là một trong các công cụ quan trọng giúp các data scientist mở rộng dataset và phân rõ các dữ liệu huấn luyện cho các mô hình học máy. Việc cắt ảnh là tầm thường, với độ phức tạp chủ yếu rơi vào việc truy xuất dữ liệu. Tuy nhiên, thông thường các mô hình học máy sẽ được huấn luyện trên một định dạng toạ độ ảnh nhất định, i.e. W, H là hằng số. Vì thế, ta phải kéo giãn ảnh. Việc kéo giãn ảnh lên các định dạng lớn hơn ban đầu sẽ khiến nó xuất hiện các "lỗ", những điểm ảnh trống do việc kéo giãn.

Một phương pháp khá dễ cài đặt cho vấn đề này là ta có thể lấy điểm màu từ bức ảnh ban đầu mà gần với toạ độ của bức ảnh mới này nhất. Tuy nhiên, việc chọn cách lấp lỗ như vậy rất dễ khiến bức ảnh trở nên đứt khúc, khiến các mảng của bức ảnh trở nên không tự nhiên liền mạch. Một phương pháp khác mà bài thực hành sử dụng chính là Bilinear Filtering. Trong đó, ta sẽ nội suy giá trị những điểm ảnh bị mất trên 2 toạ độ x, y của ảnh lớn hơn bằng giá trị các điểm ảnh tại 4 điểm $Q_{11}, Q_{12}, Q_{21}, Q_{22}$, $Q_{ij} = \left(x^{(i)}, y^{(j)}\right)$ tạo thành hình

vuông với toạ độ điểm ta đang cần tìm làm trung tâm. Sau đó, sử dụng đẳng thức phía dưới để tính giá trị của điểm ảnh [7]:

$$\frac{1}{(x^{(2)}-x^{(1)})\times(y^{(2)}-y^{(1)})}\begin{bmatrix}x^{(2)}-x & x-x^{(1)}\end{bmatrix}\begin{bmatrix}f(Q_{11}) & f(Q_{12})\\ f(Q_{21}) & f(Q_{22})\end{bmatrix}\begin{bmatrix}y^{(2)}-y\\ y-y^{(1)}\end{bmatrix}=f(x,y)$$

$$f:\mathbb{N}^2\to\mathbb{N}^3$$

Điểm thú vị của phương pháp Bilinear Filtering nằm ở việc nó có thể được đưa về dạng ánh xạ tuyến tính giữa tập điểm ảnh ở ảnh cũ và tập điểm ảnh ở ảnh mới, giúp ta tối ưu hoá tốc độ tính toán bằng NumPy và hỗ trợ tính toán vector từ phần cứng.

2.8 Cắt ảnh theo khung (masking)

Cắt ảnh theo khung có thể hiểu là ta nhân theo phần tử ma trận điểm ảnh ban đầu với một ma trận nhị phân cùng kích thước, có các thành phần điểm ảnh là bằng 1 hoặc 0 hết với mỗi điểm ảnh của ma trận nhị phân đó.

Với hướng tiếp cận như vậy, bài toán trở nên tầm thường. Ta chỉ việc tìm ma trận masking ứng với khung ta cần. Với hình tròn lấy tâm ở giữa ảnh, ta chỉ cần xét khoảng cách theo toạ độ của các điểm ảnh so với tâm và chỉ lấy những điểm ảnh có khoảng cách mong muốn. Với khung hai ellipse chéo nhau, ta thực hiện tương tự, nhưng với biểu thức ellipse:

$$\frac{\left((x-h)\times cos(R°)+(y-k)\times sin(R°)\right)^2}{a^2}+\frac{\left((x-h)\times sin(R°)-(y-k)\times cos(R°)\right)^2}{b^2}\leq 1 \qquad \text{(eq4)}$$

Với $h,\ k$ là toạ độ của điểm trung tâm ảnh, R, a, b là góc xoay so với trục hoành của ellipse; nửa chiều dài và nửa chiều rộng của ellipse. Với phương trình trên, ta có thể cho 2 ellipse có cùng chiều dài, chiều rộng và quay mỗi ellipse đối xứng với nhau qua trục tung và có tổng số đó góc là 90° để lấy ma trận nhị phân khung.

2.9 Phóng to, thu nhỏ (resizing)

Tương tự như phần cắt ảnh theo kích thước, việc truy suất thông tin của ảnh nhỏ hơn là một bài toàn tầm thường. Đối với việc thu nhỏ, ta chỉ cần lấy các điểm ảnh theo mỗi bước s tương ứng với tỉ lệ thu nhỏ so với ban đầu, thay vì lấy tuần tự tất cả điểm ảnh. Nhưng đối với việc phóng to ảnh, ta trở lại với bài toán các lỗ trống điểm ảnh xuất hiện. Với việc phóng to tỉ lệ lớn, rất có thể sẽ xảy ra tình trạng thuật toán lấp lỗ bằng điểm ảnh gần nhất (nearest-neighbor) có sai sót lớn so với tính đúng đắn dưới góc nhìn của con người.

Thế nên, tương tự như bài toán cắt ảnh theo kích thước, bài thực hành sẽ sử dụng phương pháp Bilinear Filtering nhằm cải thiện chất lượng của ảnh.

3 Cài đặt thuật toán

Đáp ứng với yêu cầu, bài thực hành này chỉ sử dụng các thư viện để tính toán trên dãy số và ma trận (NumPy), nhập xuất hình ảnh (Pillow) và matplotlib dùng để hiển thị các kết quả của thuật toán; cùng một số các module built-in của Python3.

Chi tiết cài đặt của các hàm đã được chú thích rõ ràng trong mã nguồn, nên để tránh khiến cho báo cáo trở nên dày đặc không cần thiết, ta chỉ nêu một số điểm nổi bật.

3.1 Nhập xuất hình ảnh

Về việc nhập xuất hình ảnh, có nhiều phương pháp để nhập xuất bằng thư viện built-in của Python, hoặc bằng các thư viện ngoài. Ở đây, bài thực hành sử dụng Pillow nhằm đơn giản hóa mã nguồn cũng như cung cấp một giao diện cho phép ta chuyển đổi các dữ liệu hình ảnh RGB sang một ma trận trong NumPy một cách dễ dàng.

Lưu ý ở đây rằng, vì các thuật toán sử dụng nhiều phép toán luỹ thừa, nhân trọng số, có thể dẫn đến các số cần độ chính xác cao khi biểu diễn floating-point, cũng như hỗ trợ việc chuẩn hóa giá trị RGB về miền [0,1] nhằm biểu diễn trên matplotlib, ta phải khai báo kiểu dữ liệu của ma trận là *float64*.

3.2 Hiển thị hình ảnh

Việc hiển thị hình ảnh cũng khá đơn giản, Pillow cũng đã hỗ trợ một số các hàm như Image.show() nhằm bỏ đi các bước nhọc nhằn để hiển thị hình ảnh. Nhưng để có thể hỗ trợ tốt cho đa thiết bị cũng như hỗ trợ Jupyter Notebook, bài thực hành sẽ sử dụng matplotlib để hiển thị các bảng, hình ảnh. Như đã *lưu ý* phía trên, matplotlib yêu cầu dữ liệu đầu vào cho hiển thị ảnh phải được chuẩn hóa từ khoảng [0,255] về [0,1], nên để đơn giản tổng quát hóa giao diện đầu vào, hàm sẽ tự động chia các thành phần vô hướng của ma trận cho 255.

3.3 Lưu trữ hình ảnh

Hàm lưu trữ cũng như san phẳng ma trận 2 chiều cũng khá đơn giản, chỉ việc sử dụng những thủ tục có sẵn của NumPy và Pillow. Lưu ý rằng hình ảnh nên được chuyển qua dạng lưu trữ "RGB" (mặc định ban đầu là dạng tham chiếu điểm ảnh) để quá trình lưu trữ diễn ra suông sẻ.

3.4 Cấu trúc cài đặt chương trình

Các hàm xử lý chính đều đã được cung cấp thông tin liên quan đến dữ liệu đầu vào và đầu ra, cũng như vì việc chuyển đổi, tối ưu các phép tính ma trận có thể trở nên rất rắc rối, nên báo cáo sẽ chỉ nói sơ qua về cấu trúc chung của chương trình.

Các hàm xử lý sẽ thường nhận ma trận NumPy 3D làm biến đầu tiên, và các biến tiếp theo là tham số cho các quá trình xử lý. Thông thường, người dùng sẽ không trực tiếp gọi các hàm này trừ khi cần sự thay đổi chính xác. Thay vào đó, các hàm kết thúc bằng _stub đóng vai trò làm bàn đạp cho phép chương trình đơn giản hoá các yêu cầu đầu vào cho các tính năng xử lý, và trả về một bộ gồm kết quả phép biến đổi và các tham số đầu vào mà người dùng đã nhập.

3.5 Điều chỉnh độ sáng

Input: img_2d: numpy.ndarray((WxHxC)), $A \in \mathbb{R}^{W \times H \times C}$

factor: float, $\alpha \in \mathbb{R}$

Output: numpy.ndarray((WxHxC)), $A' \in \mathbb{R}^{W \times H \times C}$

Cài đặt: Ma trận điểm ảnh đầu vào được nhân với một số thực $(1 + \alpha)$ với α là tỉ lệ tăng/giảm độ sáng. $A' = \alpha A$

3.6 Điều chỉnh độ tương phản

Input: img_2d: numpy.ndarray((WxHxC)), $A \in \mathbb{R}^{W \times H \times C}$

L: integer, $L \in \mathbb{N}$

intensity: float, $int \in [0, 1]$

Output: numpy.ndarray((WxHxC)), $A' \in \mathbb{R}^{W \times H \times C}$

Cài đặt:

Ma trận điểm ảnh đầu vào được đưa qua chức năng numpy.sum() theo chiều thứ 3, và lấy trung bình của mỗi điểm ảnh để nhận về một ma trận bảng tìm kiếm (lookup matrix) n_in_lut, $n_{in} = \left[\left[n_{ij} \right] \right] \in \mathbb{R}^{W \times H}$, $n_{ij} \in \mathbb{N}$, sau đó, ta xây dựng PDF_{in} của ma trận điểm ảnh bằng cách đếm số lượng các n_{ij} khác nhau, và nhân số lượng với tỉ lệ $1/(W \times H)$.

Tiếp theo, ta tính CDF_{in} từ PDF_{in}:

$$CDF_{in}(n_i) = p_{in}(n_i) + CDF_{in}(n_{i-1}), \qquad CDF_{in}(0) = 0$$

Từ đây, ta có thể xây dựng ma trận tỉ lệ thay đổi cường độ sáng so với ma trận ảnh đầu ra.

$$f_{map} = \left[\left[f_{x,y} \right]\right]^{W\times H} = \frac{[\,[\,255\,]\,]^{W\times H} - \left[\left[CDF_{in}\big(n_{in}(x,y)\big)\right]\right]^{W\times H} \times L}{[\,[\,255\,]\,]^{W\times H} - \left[\left[CDF_{in}\big(n_{in}(x,y)\big)\right]\right]^{W\times H}},$$

$$f_{map} \in \mathbb{R}^{W\times H}$$

Cài đặt chi tiết trong mã nguồn có thể khác vì ta phải xét các trường hợp L lớn khiến phần tử vô hướng của điểm ảnh bị tràn miền giá trị.

Tiếp đến, vì hệ số tỉ lệ ta vừa thu được nếu nhân trực tiếp với ma trận ban đầu sẽ xảy ra hiện tượng mất cân bằng màu chủ đạo, ta sẽ thực hiện:

1) Chuyển điểm ảnh từ hệ RGB $a = (a_1, \dots, a_C),\ a \in A(x,y)$ về hệ CMY $\tilde{y} = (y_1, \dots, y_C),\quad y_k = 255 - a_k,\ 1 \le k \le C$. Thu được $Y \in \mathbb{R}^{W\times H\times C}$
2) Nhân theo phần tử ma trận hệ số tỉ lệ thay đổi với ma trận điểm ảnh hệ CMY $Y'(x,y) = f_{map}(x,y)Y(x,y)$
3) Chuyển ma trận hệ CMY về hệ RGB $A' = 255 - Y'$
4) Nội suy giữa ma trận đầu vào và ma trận kết quả bằng nội suy tuyến tính thông qua biến intensity.

3.7 Lật ảnh quanh trục

Input: img_2d: numpy.ndarray((WxHxC)), $A \in \mathbb{R}^{W\times H\times C}$

axis: int, $axis = 0, 1$

Output: numpy.ndarray((WxHxC)), $A' \in \mathbb{R}^{W\times H\times C}$

Cài đặt: Để thuận lợi, ta sử dụng hàm numpy.flip(img_2d, axis) của numpy để trả về kết quả.

3.8 Chuyển ảnh về hệ trắng đen/sepia

Input: img_2d: numpy.ndarray((WxHxC)), $A \in \mathbb{R}^{W \times H \times C}$

grayscale_mode: int, $mode = 0, 1$

Output: numpy.ndarray((WxHxC)), $A' \in \mathbb{R}^{W \times H \times C}$

Cài đặt:

Để thuận tiện, chương trình lưu sẵn các hệ số của ma trận biến đổi TU-R 601-2 vào mã nguồn.

Đối với hệ trắng đen, ta chỉ việc nhân ma trận đầu vào với vector biến đổi để lấy được cường độ sáng của mỗi điểm ảnh $Lm(x, y) = A(x, y)^T T$ (lưu ý ký hiệu T phía trên $A(x, y)$ là phép chuyển vị vector). Với mỗi phần tử trong Lm, ta lặp lại nó $C - 1$ lần nữa để thăng hạng lên bằng ma trận đầu vào, và trả về kết quả.

Đối với hệ sepia, ta trực tiếp nhân từng điểm ảnh với ma trận biến đổi:

$$A'(x, y) = Sep \times A(x, y)$$

3.9 Làm mờ ảnh

Input: img_2d: numpy.ndarray((WxHxC)), $A \in \mathbb{R}^{W \times H \times C}$

std_dev: float, $\sigma \in \mathbb{R}$, phương sai của hàm Gaussian dự kiến

n_pass: int, $n_{pass} \in \mathbb{N}$, số lần chạy giải thuật

intensity: float, $int \in [0,1]$

Output: numpy.ndarray((WxHxC)), $A' \in \mathbb{R}^{W \times H \times C}$

Cài đặt:

Trước hết, ta phải xác định được chiều dài lý tưởng w_{ideal} cho kernel như đã nói đến phía trên bằng (eq1), và xác định số lần chạy lý tưởng cho w_l bằng (eq2). Tiếp đến, với m và w_l, ta thực hiện áp dụng bộ lọc trung bình cộng thông qua thủ tục iter_pass(), nhận số lần chạy, kernel width và chiều ma trận đầu ra.

Trong thủ tục iter_pass(), đầu tiên ta sẽ tính toạ độ các điểm a, b, c, d cách đều trung tâm là mỗi điểm ảnh trong ma trận đầu ra mong muốn. Để thuận tiện, ta sẽ lưu 4 ma trận

dạng $\mathbb{R}^{W\times H}$ với mỗi phần tử là toạ độ cách đều theo $\frac{w_l}{2}$ của các phần tử ma trận đầu ra (x, y). Sau đó, sử dụng công thức tính tổng (eq0) để tính giá trị trung bình của các điểm ảnh xung quanh (x, y). Và lưu kết quả cuối cùng làm điểm ảnh đầu ra.

Với w_l, ta thực hiện bộ lọc trên m lần, sau đó chuyển qua w_u và thực hiện bộ lọc trên $n - m$ lần nữa để thu được kết quả cuối cùng.

3.10 Làm nét ảnh

Input: img_2d: numpy.ndarray((WxHxC)), $A \in \mathbb{R}^{W\times H\times C}$

amount: float, $a \in \mathbb{R}$, mức độ làm nét

intensity: float, $int \in [0,1]$

Output: numpy.ndarray((WxHxC)), $A' \in \mathbb{R}^{W\times H\times C}$

Cài đặt:

Tương tự, vì ta đã có (eq3), việc tìm ảnh sắc nét qua phương pháp Unsharp Masking chỉ đơn giản là thủ tục tìm ảnh được làm trơn qua chức năng làm mờ phía trên (với std_dev=1, n_pass=3, intensity=1) để trích xuất các feature cạnh của ảnh, sau đó nhân với hệ số a và cộng ma trận kết quả vào ma trận ban đầu để có ma trận đầu ra.

$$A' = \big(A - gaussian\_blur(A, 1, 3)\big)\alpha + A$$

3.11 Cắt ảnh theo kích thước

Input: img_2d: numpy.ndarray((WxHxC)), $A \in \mathbb{R}^{W\times H\times C}$

radius: int, $r \in \mathbb{N}$

Output: numpy.ndarray((WxHxC)), $A' \in \mathbb{R}^{W\times H\times C}$

Cài đặt:

Cài đặt của chức năng này tương đối đơn giản, ta chỉ việc truy suất dữ liệu của ma trận ban đầu thông qua slicing ndarray, sau đó thực hiện bilinear interpolating và phóng to ảnh kết quả.

Thủ tục cho Bilinear Interpolating tương tự như kỹ thuật tạo các ma trận offsets như ở phần làm mờ ảnh, kết quả cuối cùng là tổng theo trọng số của 4 điểm nội suy xunh quanh.

3.12 Cắt ảnh theo mask tròn/ellipse

Input: img_2d: numpy.ndarray((WxHxC)), $A \in \mathbb{R}^{W \times H \times C}$

[radius: int, $r \in \mathbb{N}$ if circular masking]

[a: int, $a \in \mathbb{N}$ if ellipse masking]

[b: int, $b \in \mathbb{N}$ if ellipse masking]

Output: numpy.ndarray((WxHxC)), $A' \in \mathbb{R}^{W \times H \times C}$

Cài đặt:

Đầu tiên, ta tìm trung tâm của ảnh (x_c, y_c), từ đây ta tạo một ma trận khoảng cách theo chiều của ma trận ban đầu $D \in \mathbb{R}^{W \times H}$ và tính khoảng cách Euclid của các phần tử so với toạ độ trung tâm (x_c, y_c). Sau đó, ta thực hiện so sánh các phần tử của ma trận khoảng cách với r đối với mask tròn, (eq4) với mask ellipse, để thu được ma trận mask nhị phân. Cuối cùng ta nhân theo phần tử ma trận nhị phân vừa nhận được với ma trận điểm ảnh đầu vào.

3.13 Phóng to và Thu nhỏ ảnh

Input: img_2d: numpy.ndarray((WxHxC)), $A \in \mathbb{R}^{W \times H \times C}$

factor: float, $a \in \mathbb{R}$, tỉ lệ phóng to/thu nhỏ

Output: numpy.ndarray((factor*WxfactorHxC)), $A' \in \mathbb{R}^{aW \times aH \times C}$

Cài đặt:

Trước hết, hàm sẽ xét xem $a \leq 1$ hay không. Nếu thoả, hàm sẽ thực hiện thu nhỏ với tỉ lệ $1/a$ bằng cách nhảy bước khi lấy các điểm ảnh của ma trận đầu vào.

Mặt khác, nếu $a > 1$, hàm sẽ thực hiện gọi đến thủ tục Bilinear Interpolating nhằm nâng chiều dài, chiều cao của ma trận đầu vào và lấp các pixel trống.

3.14 Hàm main đầu vào

Hàm main đóng vai trò là giao diện về một thủ tục nhập, xử lý và xuất ảnh của chương trình. Các thông số đầu vào đòi hỏi người dùng phải nhập đường dẫn của tập tin ảnh, chọn chức năng thực hiện, chọn các tham số cho tính năng xử lý. Khi đã xử lý xong, ảnh sẽ được in ra qua matplotlib, đồng thời chương trình cũng sẽ cho phép người dùng lưu ảnh PNG.

4 Thực nghiệm

Tất cả các cài đặt thuật toán đều được thực hiện trong Python thuần, cùng sự kết hợp của một số thư viện nhập xuất và tính toán khác. Nhằm thể hiện rõ được mối tương quan giữa các thông số đầu vào cũng như tính hiểu quả của các phương pháp chọn điểm trung tâm khởi đầu, các thực nghiệm *test* đều được tính toán và lưu lại. Các tính toán đều có kết quả quang nghiệm cũng như các số liệu về tổng sai số và tổng thời gian chạy tính bằng giây.

4.1 Mức độ hoàn thiện

STT	Chức năng/Hàm	Mức độ hoàn thành
1	Thay đổi độ sáng	100%
2	Thay đổi độ tương phản	100%
3.1	Lật ảnh ngang	100%
3.2	Lật ảnh dọc	100%
4.1	RGB thành ảnh xám	100%
4.2	RGB thành ảnh sepia	100%
5.1	Làm mờ ảnh	100%
5.2	Làm sắc nét ảnh	100%
6	Cắt ảnh theo kích thước	100%
7.1	Cắt ảnh theo khung tròn	100%
7.2	Cắt ảnh theo khung ellipse	100%
8	Hàm main	100%
9	Phóng to/Thu nhỏ	100%

4.2 Dataset

Dataset được sử dụng là bức ảnh mẫu Lena được lưu trữ dưới dạng RGB gồm các màu tự nhiên, được chọn sao cho các dải màu có sự lặp lại nhưng cũng có sự khác biệt đáng kể trong dải tương phản của nó.

Figure 2 Ảnh mẫu Lena

4.3 Kết quả thực nghiệm

Figure 3 Tăng độ sáng, factor = 0.3

Figure 4 Tăng độ tương phản, factor = 0.2

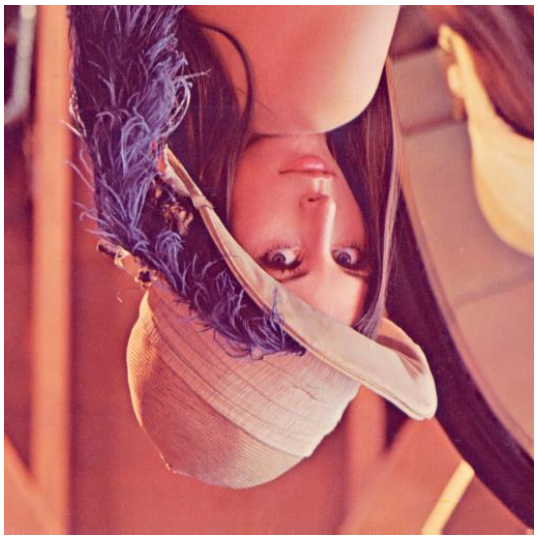

Figure 5 Lật ảnh theo chiều ngang

Figure 6 Lật ảnh theo chiều dọc

Figure 7 Ảnh hệ trắng đen

Figure 8 Ảnh hệ sepia

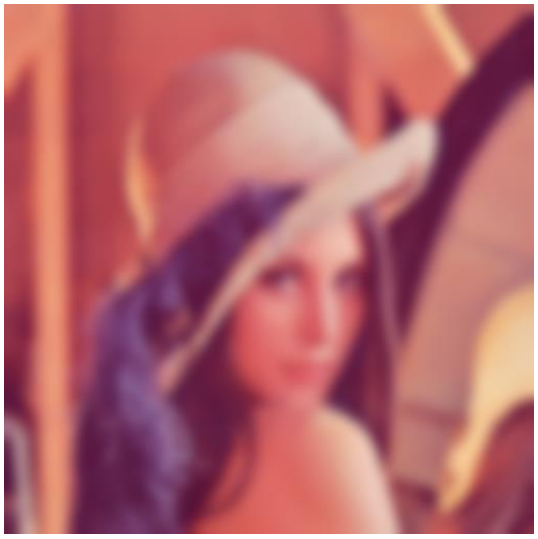

Figure 9 Ảnh bị làm mờ, phương sai hàm Gaussian = 6

Figure 10 Làm nét ảnh, độ làm nét = 1-0.7 = 0.3

Figure 11 Cắt ảnh theo kích thước, radius = 64

Figure 12 Cắt ảnh theo khung tròn

Figure 13 Cắt ảnh theo khung ellipse

Figure 14 Phóng to ảnh, factor = x2

Figure 15 Thu nhỏ ảnh, factor = 0.5

4.4 Nhận xét kết quả

Các kết quả đều được trả về trong thời gian phù hợp, với tổng thời gian chạy cho tất cả chức năng trong 1 lượt là $\pm 0.5s + 2s$, trên hệ máy có nhân Ryzen 7 5800H. Các kết quả trả về đều có chất lượng trong kì vọng và thoả mãn yêu cầu cơ bản của việc xử lý, biến đổ ảnh hàng loạt.

5 Kết luận

Bài thực hành đã cài đặt và thực nghiệm trên một tập dữ liệu đầu vào là một bức ảnh cùng các điểm dữ liệu là một vector chứa ba giá trị R, G, B đại diện cho điểm ảnh. Bài thực hành áp dụng các phương pháp biến đổi theo các hàm ánh xạ tuyến tính nhằm nâng cao chất lượng cũng như biến đổi theo nhu cầu các dữ liệu ảnh.

Cùng với đó, bài thực hành cũng khám phá một vài cải tiến, cách thức tối ưu thời gian chạy bằng cách đánh đổi bộ nhớ, và áp dụng một số thủ thuật trong các bài báo khoa học nhằm tăng chất lượng của các tính năng xử lý. Bài thực hành sử dụng các kỹ thuật lập trình cũng như ma trận hóa các dữ liệu nhằm tận dụng tối đa sức mạnh của các thư viện tính toán đại số tuyến tính (NumPy), kết hợp với các thư viện nhập xuất nhằm đơn giản hóa quá trình chạy chương trình.

Lời cảm ơn

Xin cảm ơn các tài liệu liên quan đến NumPy từ thầy Nguyễn Ngọc Toàn cũng như chi tiết các phương pháp thao tác, biến đổi ma trận từ thầy Trần Hà Sơn.

6 Tham khảo

[1] j.-h. Han, S. Yang and B.-U. Lee, "A Novel 3-D Color Histogram Equalization Method," *IEEE Transactions on Image Processing,* vol. 2, no. 20, pp. 506-512, 2011.

[2] S. K. Naik and C. A. Murthy, "Hue-Preserving Color Image Enhancement Without Gamut Problem," *IEEE TRANSACTIONS ON IMAGE PROCESSING,* vol. 12, no. 20, pp. 1591-1598, 2003.

[3] R. Fisher, S. Perkins, A. Walker and E. Wolfart, "Unsharp Filter," 2003. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/unsharp.htm. [Accessed July 2024].

[4] Wikipedia, "Gaussian Blur," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Gaussian_blur. [Accessed July 2024].

[5] P. Kovesi, "Fast Almost-Gaussian Filtering," IEEE, Sydney, 2010.

[6] Wikipedia, "Unsharp Masking," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Unsharp_masking. [Accessed July 2024].

[7] Wikipedia, "Bilinear Interpolation," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Bilinear_interpolation. [Accessed July 2024].