TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM

KHOA ĐIỆN - ĐIỆN TỬ

MÔN HỌC: THỰC TẬP XỬ LÝ ẢNH Y SINH

BÁO CÁO CUỐI KỲ

ĐỀ TÀI: IMAGE COMPARISON

GVHD: T.S Nguyễn Mạnh Hùng

SVTH:

Họ và tên	MSSV
Kiêm Thị Đào Hoa	19129017
MailorKham Kheng Kham	19129L11
Võ Vĩnh Trường	19129057
Phạm Thị Thảo Vân	19129066

Thành phố Hồ Chí Minh, Tháng 04 năm 2022

LỜI CẢM ƠN

Trong quá trình thực hiện đề tài, nhóm chúng tôi đã nhận được nhiều sự giúp đỡ, đóng góp ý kiến và chỉ bảo tận tình của Thầy.

Đặc biệt, chúng tôi xin gửi lời cảm ơn chân thành đến Thầy Nguyễn Mạnh Hùng đã trực tiếp quan tâm giúp đỡ, khuyến khích tinh thần học hỏi cũng như tạo mọi điều kiện để chúng tôi có thể hoàn thành tốt đề tài này.

Với điều kiện thời gian cũng như kiến thức còn hạn hẹp, trong quá trình nghiên cứu và thực hiện không thể tránh được những thiếu sót. Chúng tôi rất mong nhận được sự chỉ bảo, đóng góp ý kiến của quý thầy cô cùng các bạn để có thể bổ sung và phát triển thêm về đề tài.

MỤC LỤC

DANH SÁCH HÌNH ẢNH

1. LÝ THUYẾT

Ở chương này, ta cần sử dụng một lượng lớn dữ liệu về tim. Với tập dữ liệu của TS. Nguyễn Mạnh Hùng cung cấp, ta sẽ học được những kiến thức cơ bản về ***registration***, lấy mẫu lại (***resampling)*** và so sánh hình ảnh (***image comparison***). Sau đó, ta sẽ sử dụng các phép đo trích xuất để đánh giá tệp ảnh.

1.1 Biến đổi trong miền thời gian (Spatial transformations)

1.1.1. Các phép biến đổi trong miền thời gian

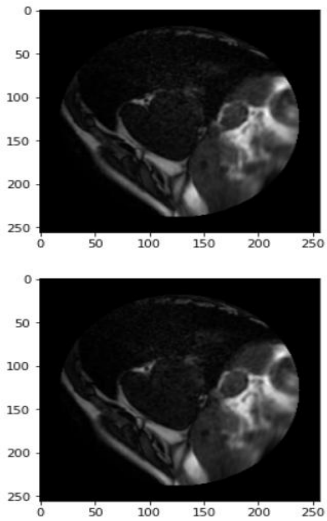
Ta đã thấy có nhiều cách để có thể đánh giá một hình ảnh, vì lý do cần trích xuất đặc trưng của hình ảnh cũng như là phục vụ cho việc quan sát nên cần sử dụng những phương pháp ấy.

1.1.2. Tập dữ liệu SDC4201

- Trong bài báo cáo này sử dụng tập dữ liệu về hình ảnh của tim bao gồm 30 ảnh lát cắt của phổi

1.1.3. Sự thay đổi về tập ảnh (Significant variability)

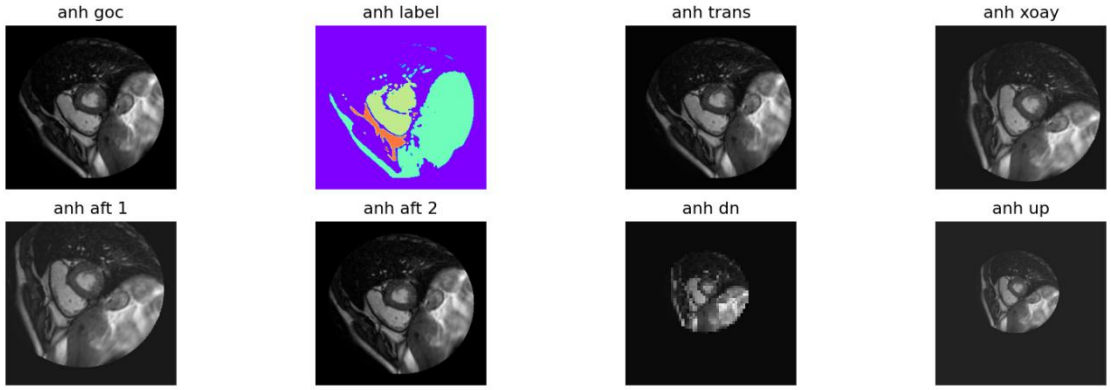
Với một tập dữ liệu hình ảnh lớn thì sẽ có sự khác biệt về thang cường độ, hướng đối tượng và vị trí đối tượng trong cửa sổ hình ảnh.



Hình 1. 1. Ví dụ một số hình ảnh trong tập dữ liệu.

1.1.4. Sự ghi ảnh (Registration)

Một cách để giải quyết vấn đề này là ghi hình ảnh vào một vị trí và hệ tọa độ được xác định trước. Ví dụ: Ta có thể làm cho tất cả các hình ảnh thẳng hàng với một hình ảnh mẫu hoặc một biểu đồ. Quá trình sắp xếp 2 hình ảnh với nhau được gọi là "registration". Sự ghi ảnh yêu cầu thực hiện nhiều chuyển đổi đối với một hình ảnh, chẳng hạn như dịch chuyển, xoay và thu nhỏ hình ảnh đó.



Hình 1. 2. Ví dụ về Registration.

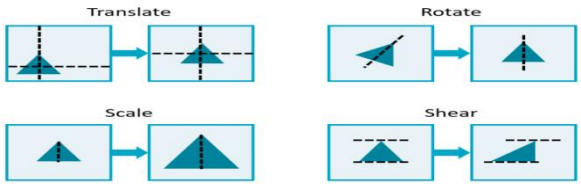
1.1.5. Các phép biến đổi Affine bảo toàn các điểm, đường thẳng và mặt phẳng (Affine transformations preserve points, lines, and planes)

Phép biến đổi Affine biến đổi một hình ảnh trong khi vẫn bảo toàn tất cả các điểm, đường thẳng và mặt phẳng. Dưới đây là bốn ví dụ về phép biến đổi affine:

- *Phép dịch (Translation)* là sự dịch chuyển của một hình ảnh dọc theo một trục. Ví dụ, nó có thể được sử dụng để căn giữa một đối tượng.

- Mặt khác, *phép quay (Rotation)* sẽ biến hình ảnh dọc theo một mặt phẳng.

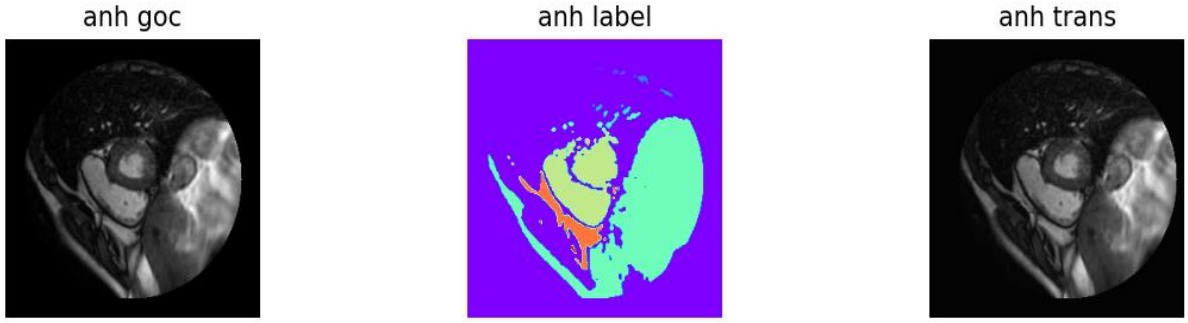
- *Chia tỷ lệ (Scaling)* làm tăng hoặc giảm kích thước của hình ảnh và hoạt động cắt sẽ dịch chuyển các đầu của trục ra xa nhau.



Hình 1. 3. Các phép biến đổi affine.

1.1.6. Phép dịch (Translation)

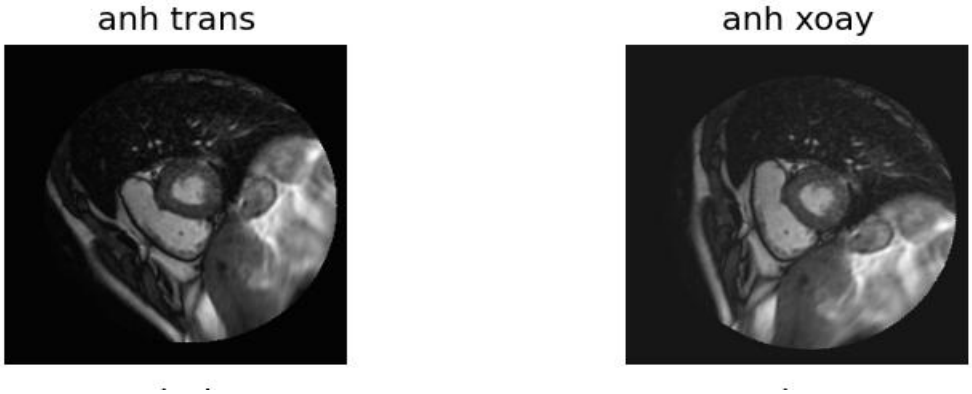
Hãy xem cách chúng ta có thể thực hiện một số biến đổi này trong *SciPy*. Ở đây chúng ta có một quả tim nằm ngoài trung tâm và chúng ta muốn di chuyển đến trung tâm của hình ảnh. Đầu tiên, chúng ta sẽ load hình ảnh đầu vào bằng thư viện *ImageIO.* Sau đó, chúng ta sẽ lọc ảnh và tiếp tục gán label (để thấy quả tim). Tiếp theo, tính toán sự khác biệt giữa khối tâm hiện tại của phần đầu và khối tâm mục tiêu, cho cả hàng và cột. Cuối cùng, chúng ta gọi hàm *shift()* của *SciPy*, truyền vào hình ảnh và số lượng pixel chúng ta cần để di chuyển dọc theo trục thứ nhất và thứ hai.



Hình 1. 4. Ví dụ minh họa về phép dịch.

1.1.7. Phép xoay (Rotation)

Các phép xoay có thể được thực hiện bằng cách sử dụng hàm *rotate()*. Góc quay được xác định bằng độ, với các số dương chỉ lên từ phương ngang và các số âm hướng xuống. Trong không gian hai chiều, chúng ta luôn quay dọc theo mặt phẳng xy, nhưng trong không gian ba chiều, có ba mặt phẳng quay mà chúng ta có thể sử dụng.



Hình 1. 5. Ví dụ minh họa về phép xoay.

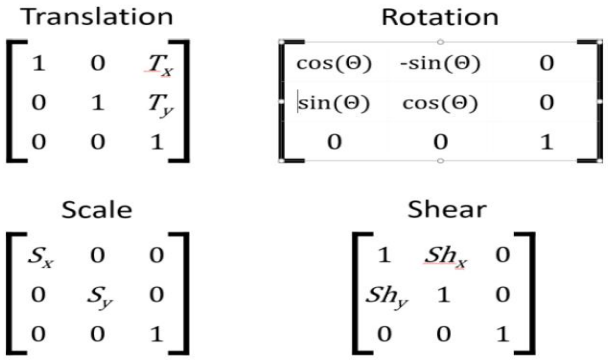
1.1.8. Xoay hình ảnh (Image rotation)

Một lưu ý với hàm *rotate()* là mặc định bảo toàn tất cả dữ liệu gốc. Điều này có nghĩa là hình ảnh được xoay của bạn thực sự có thể lớn hơn hình ảnh gốc của bạn. Để giữ nguyên hình dạng hình ảnh ban đầu của bạn, đặt "reshape = False" vào lệnh gọi hàm.

Hình 1. 6. Ví dụ minh họa về phép xoay 1 ảnh (sử dụng tập dữ liệu khác)

1.1.9. Ma trận biến đổi (Transformation matrix)

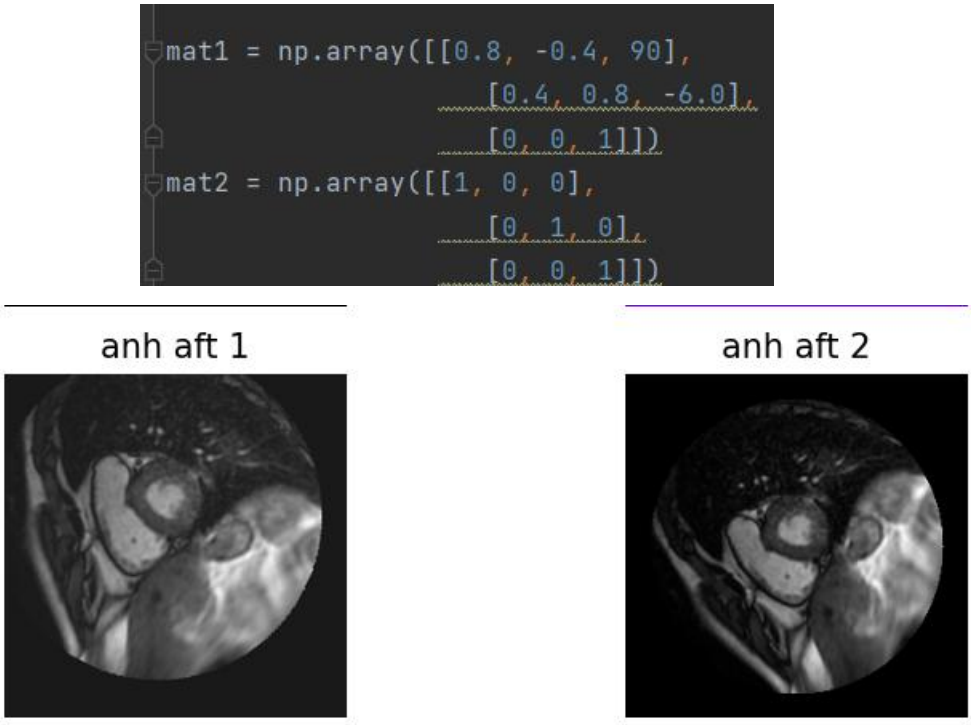
Đối với các *registrations* phức tạp, tính toán một ma trận chuyển đổi giữa hình ảnh gốc và ảnh mong muốn có thể hữu ích. Về cơ bản, các phần tử của ma trận chuyển đổi mã hóa các lệnh cho các hoạt động mà chúng ta đã nói ở trên: dịch, quay, chia tỷ lệ và kéo (translation, rotation, scaling, and shearing). Chúng ta không nói đến các phương pháp tính toán các ma trận này, nhưng có thể xem chúng được sử dụng như thế nào để đơn giản hóa quá trình registration.



Hình 1. 7. Ma trận biến đổi.

1.1.10. Áp dụng ma trận biến đổi (Applying a transformation matrix)

Đầu tiên, chúng ta tạo ma trận biến đổi. Đầu tiên chúng ta hãy sử dụng ma trận nhận dạng, có các ma trận dọc theo đường chéo và các số 0 bên ngoài nó. Chúng ta có thể áp dụng nó bằng cách truyền hình ảnh và ma trận vào hàm *affine_transform()*. Hình ảnh kết quả giống với hình ảnh gốc. Tiếp theo, hãy thao tác với các giá trị ma trận mã hóa dịch chuyển và thay đổi tỷ lệ. Khi chúng ta áp dụng ma trận mới này và vẽ biểu đồ kết quả, ta có thể thấy rằng hình ảnh đã được căn giữa và lớn hơn.



Hình 1. 8. Ví dụ áp dụng ma trận biến đổi (Sử dụng tập dữ liệu khác

1.2. Lấy mẫu lại và nội suy (Resampling and interpolation)

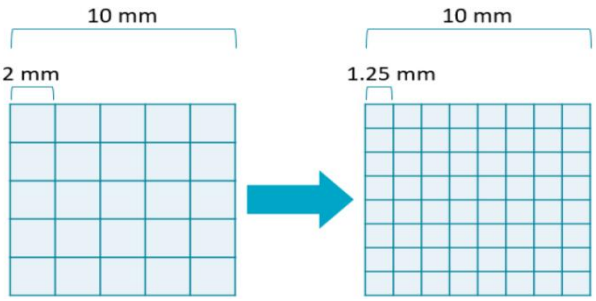
1.2.1. Lấy mẫu lại và nội suy

Khi so sánh các hình ảnh, sự khác biệt về hình dạng mảng và tỷ lệ lấy mẫu có thể gây trở ngại cho việc phân tích. Lấy mẫu lại *(resampling)* là một cách để giải quyết vấn đề này.

1.2.2. Lấy mẫu lại thay đổi hình dạng mảng (Resampling changes the

array shape)

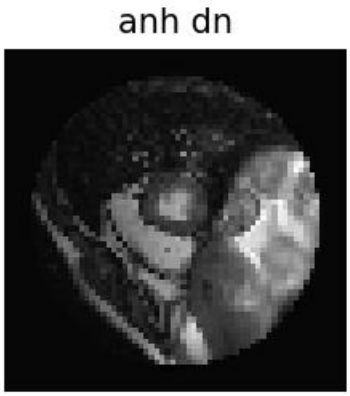
Lấy mẫu lại là quá trình thay đổi dữ liệu thành một mảng khác. Nó khác với cắt xén ở chỗ ảnh xem không thay đổi. Thay vào đó, số lượng pixel được tăng hoặc giảm tùy vào resampling, do đó thay đổi hình dạng của mảng.



Hình 1. 9. Minh họa Resampling

1.2.3. Downsampling

Một ứng dụng hữu ích là "downsampling", trong đó thông tin được hợp nhất trên nhiều pixel để giảm kích thước hình ảnh. Ta sẽ giảm một nửa kích thước này bằng cách sử dụng hàm *zoom()* của *SciPy*. *zoom()* điều chỉnh số lượng phần tử dọc theo mỗi trục theo một hệ số nhất định. Chúng ta sẽ đặt hệ số thu phóng là 0,5. Điều này làm giảm một nửa số phần tử trên mỗi trục để nó là 128 x 128. Việc vẽ đồ thị cho thấy hình ảnh hiện có ít chi tiết hơn trước, nhưng nó cũng chiếm một nửa dung lượng bộ nhớ.

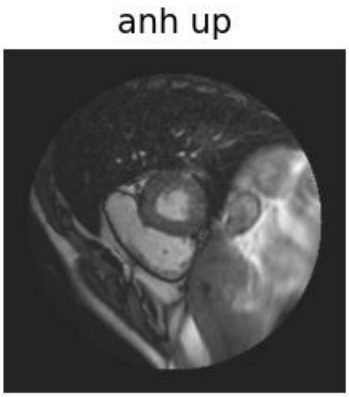


Hình 1. 10. Ví dụ về Downsampling.

1.2.4. Upsampling

Cũng có thể lấy mẫu lên các lưới dày đặc hơn. Nhưng lưu ý rằng điều này không

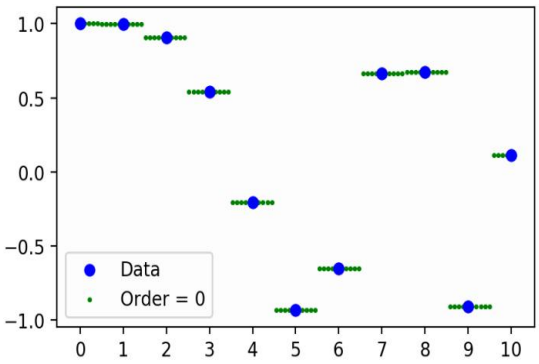
thực sự làm tăng độ phân giải. Mặc dù ta đưa nhiều pixel hơn vào hình ảnh, nhưng sẽ không thêm thông tin mới mà trước đó chưa có. Một ứng dụng hữu ích của *upsampling* là làm cho tốc độ lấy mẫu dọc theo các kích thước khác nhau bằng nhau, chẳng hạn, để tạo voxels hình khối. Để lấy mẫu một hình ảnh, chúng ta gọi hàm *zoom()* và chỉ định hệ số thu phóng lớn hơn.



Hình 1. 11. Ví dụ về Upsampling.

1.2.5. Phép nội suy (Interpolation)

Việc lấy mẫu lại thực sự tạo ra một hình ảnh hoàn toàn mới dựa trên hình ảnh cũ. Và trong hầu hết các trường hợp, việc điền vào hình ảnh mới này yêu cầu ước tính dữ liệu ban đầu không có ở đó. Quá trình ước lượng này được gọi là nội suy. Ví dụ, ở đây chúng tôi có một tập dữ liệu 10 điểm đơn giản. Nếu chúng ta muốn lấy mẫu nó để được 100 điểm, ta phải ước tính các giá trị nên có giữa mỗi điểm ban đầu này.

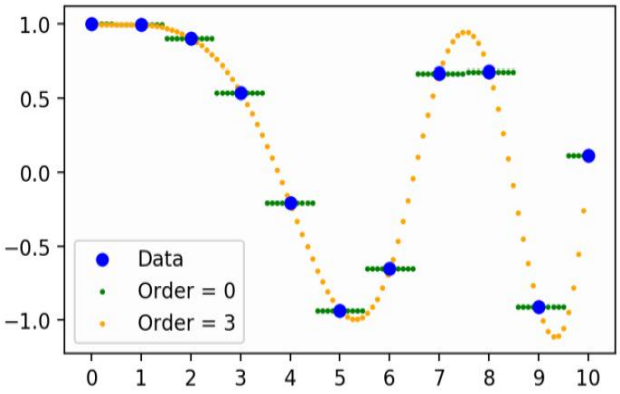


Hình 1. 12. Đồ thị kết quả sử dụng phép nội suy.

Một cách tiếp cận là sử dụng giá trị gốc gần nhất cho mỗi điểm mới. Đây là phép

nội suy bậc 0, vì không mô hình hóa bất kỳ mối quan hệ nào giữa các giá trị ban đầu.

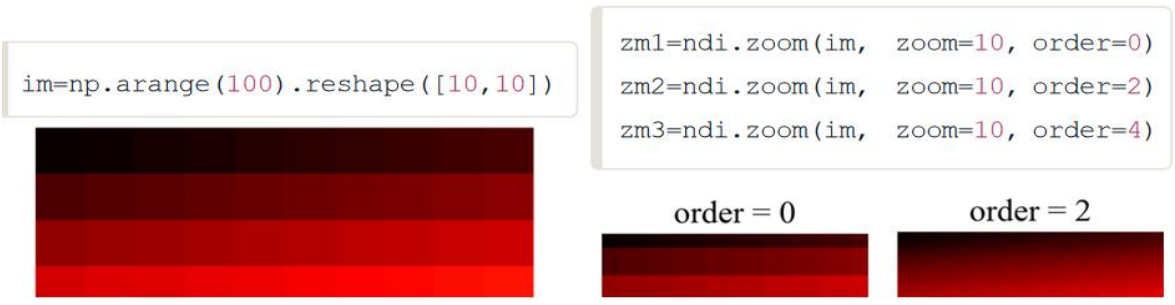
Để ước tính bậc cao hơn, *SciPy* sử dụng phép nội suy B-spline, sử dụng một tập hợp các hàm để mô hình hóa không gian giữa các điểm. Thứ tự kiểm soát mức độ phức tạp của các hàm này: bậc 1 là tuyến tính, bậc 2 là phương trình bậc hai, v.v. Trong ví dụ này, ta có thể thấy rằng phép nội suy khối tạo ra một đường cong mượt mà giữa các điểm.



Hình 1. 13. Đồ thị kết quả sử dụng phép nội suy B-spline.

1.2.6. Nội suy 2 chiều (Interpolation in 2D)

Ở đây ta tạo một hình ảnh 10 x 10 có các giá trị tăng dần. Nếu lấy mẫu lại trên ma trận 100 x 100, việc thay đổi bậc sẽ ảnh hưởng đến "độ mịn" của hình ảnh thu được. Với bậc 0, trả về hình ảnh ban đầu trên một ma trận mới. Tuy nhiên, ở các bậc cao hơn, chúng ta có thể thấy gradient thay đổi mượt mà hơn dọc theo mỗi trục. Sự khác biệt cơ bản giữa phép nội suy bậc thấp và bậc cao nằm ở thời gian tính toán: phép nội suy bậc 5 cho một khối 3D, có thể mất rất nhiều thời gian để tính toán.

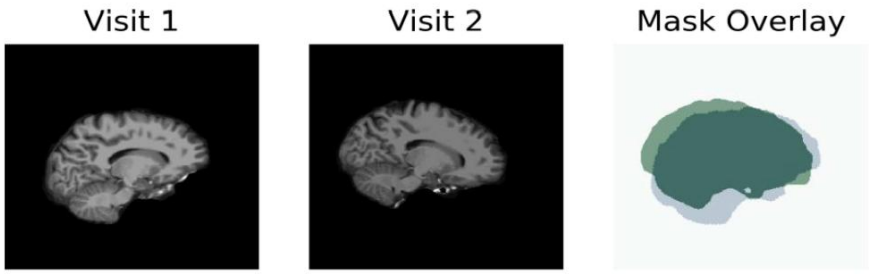


Hình 1. 14. Ví dụ về phép nội suy 2D.

1.3. So sánh hình ảnh (Comparing images)

1.3.1. So sánh hình ảnh

Ở phần này ta sẽ tìm hiểu làm thế nào để biết hình ảnh được ghi tốt? Làm thế nào để so sánh một hình ảnh được phân đoạn tự động với một hình ảnh được gắn nhãn thủ công? Để so sánh trực tiếp hai mảng, ta phải tạo ra các thước đo về độ giống nhau của hình ảnh.



Hình 1. 15. Minh họa về comparing images.

Phía trên là hai phần não của cùng một người, được chụp vào những thời điểm khác nhau. Có thể thấy rằng chúng tương tự nhau nhưng không thẳng hàng với nhau. Để định lượng chính xác mức độ không hoàn hảo của sự liên kết này, chúng ta cần áp dụng một hàm để so sánh hai hình ảnh.

1.3.2. Các chỉ số tóm tắt (Summary metrics)

Ở cấp độ pixel, có thể có hàng nghìn điểm so sánh giữa hai hình ảnh. Việc cần làm của ta là tổng hợp tất cả những so sánh này thành một con số duy nhất. Cũng như các lĩnh vực khác của khoa học dữ liệu, chẳng hạn như machine learning, có nhiều cách để đánh giá dữ liệu. Có các hàm mục tiêu *(cost function)*, chẳng hạn như sai số tuyệt đối trung bình hoặc sai số bình phương trung bình, cần được giảm thiểu. Các chức năng khách quan, chẳng hạn như là thương số của (tổng số giao điểm / tổng số các pixel) được cho là được tối đa hóa.

1.3.3. Sai số tuyệt đối trung bình (Mean absolute error)

Áp dụng với hai hình ảnh ở trên và tính toán độ tương đồng bằng cách sử dụng

hàm mục tiêu: sai số tuyệt đối trung bình.

- Đầu tiên, đọc các hình ảnh. Sau đó, tìm lỗi tại mỗi pixel bằng cách lấy *im1* trừ *im2*.

- Tiếp theo, lấy giá trị tuyệt đối của sai số, bởi vì ta quan tâm đến việc các hình ảnh có khác nhau theo bất kỳ cách nào hay không, chứ không phải liệu hình ảnh có lớn hơn hình ảnh kia hay không.

- Cuối cùng, lấy giá trị trung bình của toàn bộ hình ảnh lỗi để có được một thước đo tóm tắt duy nhất về mức độ tương tự.

Mục tiêu là không đạt được sai số tuyệt đối trung bình bằng 0. Trên thực tế, điều đó có nghĩa là các hình ảnh giống hệt nhau, nhưng không phải vậy. Thay vào đó, muốn giảm thiểu hàm mục tiêu bằng cách thay đổi một hoặc cả hai hình ảnh. Thay đổi và xoay hình ảnh đầu tiên, sau đó tính toán lại. Chúng ta tính toán sai số tuyệt đối trung bình theo cùng một cách như trước, sử dụng hình ảnh mới. Mục tiêu mới thấp hơn mục tiêu trước đó, cho thấy rằng cả hai được tương đồng tốt hơn so với trước đó.

Hình 1. 16. Ví dụ về Mean absolute error.

1.3.4. Thương số của tổng số giao điểm / tổng số các pixel (Intersection of the union)

Một vấn đề đối với phương pháp sai số tuyệt đối trung bình là các mô có giá trị cường độ cao sẽ gây ra sai số nhiều hơn các loại khác. Một cách khắc phục là so sánh các

mặt nạ hình ảnh thương số của (tổng số giao điểm / tổng số các pixel) (*Intersection of the union*) là một biện pháp đặc biệt phù hợp với điều này.

- Đầu tiên, chúng ta tạo mặt nạ hình ảnh bằng cách chọn các pixel lớn hơn 0.

- Thứ hai, chúng ta lấy giao điểm của hai mặt nạ: nghĩa là, các pixel xuất hiện trong cả hai mặt nạ.

- Tiếp theo, tìm hợp: các pixel xuất hiện trong một trong hai mặt nạ.

- Cuối cùng, chúng ta chia tổng của giao điểm cho tổng của phần giao.

Kết quả là một số từ 0 đến 1, với 0 đại diện cho không có các pixel chung với nhau giữa 2 ảnh mặt nạ và 1 đại diện cho sự tương đồng hoàn hảo.

```
def intersection_of_union(im1, im2):
    i = np.logical_and(im1, im2)
    u = np.logical_or(im1, im2)
    return i.sum() / u.sum()
print('Intersection of the union is:',intersection_of_union(im_bin1, im_bin2))
```

Hình 1. 17. Ví dụ về Intersection of the union.

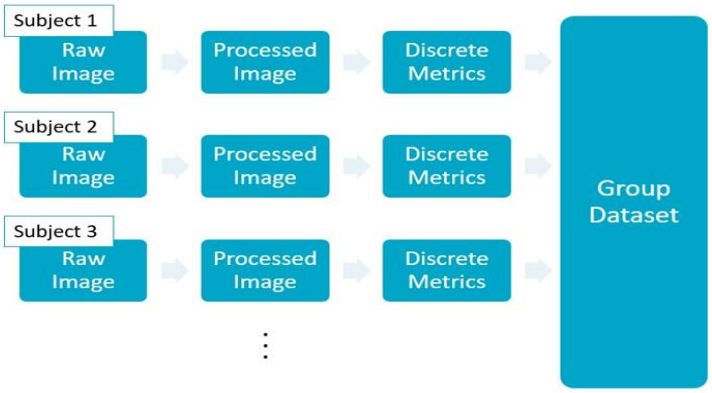
1.4. Chuẩn hóa các phép đo

1.4.1. Chuẩn hóa các phép đo

Để làm tròn khóa học, chúng ta hãy thảo luận về tầm quan trọng của việc tính toán nhiễu khi so sánh các hình ảnh.

1.4.2. Quy trình phân tích

Được thể hiện ở đây là một quy trình làm việc nhiều chủ đề chung, trong đó mỗi chủ đề có một hình ảnh thô mà chúng tôi xử lý, đo lường và lưu trữ trong một tập dữ liệu nhóm chung. Trong mô hình này, mỗi hình ảnh được xử lý độc lập, nhưng bạn đang trích xuất các chỉ số giống nhau từ mỗi chủ đề.


Subject 1
Raw Image
Processed Image
Discrete Metrics
Subject 2
Raw Image
Processed Image
Discrete Metrics
Subject 3
Raw Image
Processed Image
Discrete Metrics
Group Dataset

2. DOCUMENT

2.1. Center_of_mass

scipy.ndimage.center_of_mass(input, labels=None, index=None): Tính khối lượng tâm các giá trị của một mảng tại các nhãn.

Tham số:

Input (ndarray): Dữ liệu để tính toán khối lượng tâm. Khối lượng có thể tích cực hoặc tiêu cực.

Labels (ndarray, optional): Nhãn cho các đối tượng trong đầu vào, như được tạo bởi ndimage.label. Chỉ được sử dụng với chỉ mục. Kích thước phải giống với đầu vào.

Index (int hoặc chuỗi ints, optional): Các nhãn dùng để tính khối lượng tâm. Nếu không được chỉ định, tất cả các nhãn lớn hơn 0 sẽ được sử dụng. Chỉ được sử dụng với nhãn.

Kết quả trả về:

center_of_mass (tuple hoặc danh sách các tuple): Tọa độ của khối tâm.

2.2. Shift

scipy.ndimage.shift(input, shift, output=None, order=3, mode='constant', cval=0.0, prefilter=True): Mảng được dịch chuyển bằng cách sử dụng nội suy spline của thứ tự được yêu cầu. Các điểm bên ngoài ranh giới của đầu vào được điền theo chế độ đã cho.

Tham số:

Input (array_like): Ngõ vào là một mảng

Shift (float hoặc chuỗi): Sự dịch chuyển dọc theo các trục. Nếu một float, shift là giống nhau cho mỗi trục. Nếu một chuỗi, shift phải chứa một giá trị cho mỗi trục.

Output (mảng, dtype hoặc tùy chọn): Mảng để đặt đầu ra hoặc kiểu của mảng được trả về. Theo mặc định, một mảng có cùng loại với đầu vào sẽ được tạo.

Oder (int, tùy chọn): Thứ tự của nội suy spline, mặc định là 3. Thứ tự phải nằm trong phạm vi 0 – 5.

Mode {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}, optional: Tham số chế độ xác định cách mảng đầu vào được mở rộng ra ngoài ranh giới của nó. Mặc định là 'hằng số'. Hành vi cho mỗi giá trị hợp lệ như sau (xem các ô bổ sung và chi tiết về chế độ ranh giới):

+ ‘Reflect’: ‘Phản xạ’ (d c b a | a b c d | d c b a): Đầu vào được mở rộng bằng cách phản ánh về cạnh của pixel cuối cùng. Chế độ này đôi khi cũng được gọi là đối xứng nửa mẫu.

+ ‘Grid-mirror’: ‘Gương lưới’: Đây là một từ đồng nghĩa với 'phản ánh'.

+ ‘Constant’: ‘Hằng số’ (k k k k | a b c d | k k k k): Đầu vào được mở rộng bằng cách điền vào tất cả các giá trị ngoài cạnh với cùng một giá trị không đổi, được xác định bởi tham số cval. Không có phép nội suy nào được thực hiện ngoài các cạnh của đầu vào.

+ ‘Grid-constant’: ‘Hằng số lưới’ (k k k k | a b c d | k k k k): Đầu vào được mở rộng bằng cách điền vào tất cả các giá trị ngoài cạnh với cùng một giá trị không đổi, được xác định bởi tham số cval. Nội suy cũng xảy ra đối với các mẫu nằm ngoài phạm vi đầu vào.

+ ‘Nearest’: ‘Gần nhất’ (a a a a | a b c d | d d d d): Đầu vào được mở rộng bằng cách sao chép pixel cuối cùng.

+ ‘Mirror’ (d c b | a b c d | c b a): Đầu vào được mở rộng bằng cách phản ánh về tâm của pixel cuối cùng. Chế độ này đôi khi cũng được gọi là đối xứng toàn bộ mẫu.

+ ‘Grid-wrap’: ‘Lưới quấn’ (a b c d | a b c d | a b c d): Đầu vào được mở rộng bằng cách quấn quanh cạnh đối diện.

+ ‘Wrap’: ‘Quấn’ (d b c d | a b c d | b c a b): Đầu vào được mở rộng bằng cách quấn quanh cạnh đối diện, nhưng theo cách sao cho điểm cuối cùng và điểm ban đầu trùng nhau chính xác. Trong trường hợp này, người ta không xác định rõ mẫu nào sẽ

được chọn tại điểm trùng lắp.

Cval (vô hướng, tùy chọn): Giá trị để lấp đầy các cạnh trước đây của đầu vào nếu chế độ là 'không đổi'. Mặc định là 0,0.

Prefilter (bool, tùy chọn): Xác định xem mảng đầu vào có được lọc trước bằng spline_filter trước khi nội suy hay không. Giá trị mặc định là True, sẽ tạo một mảng float64 tạm thời gồm các giá trị được lọc nếu thứ tự> 1. Nếu đặt giá trị này thành Sai, đầu ra sẽ hơi mờ nếu thứ tự> 1, trừ khi đầu vào được lọc trước, tức là nó là kết quả của việc gọi spline_filter trên đầu vào ban đầu.

Kết quả trả về

Shift (ndarray): Đầu vào đã thay đổi.

2.3. Rotate

scipy.ndimage.rotate(input, angle, axes=(1, 0), reshape=True, output=None, order=3, mode='constant', cval=0.0, prefilter=True): Xoay một mảng. Mảng được quay trong mặt phẳng được xác định bởi hai trục được cung cấp bởi tham số trục bằng cách sử dụng phép nội suy spline của thứ tự được yêu cầu.

Tham số:

Input (array_like): Ngõ vào là một mảng

Angle (float): Góc quay tính bằng độ.

Axes (tuple of 2 ints, optional): Hai trục xác định mặt phẳng quay. Mặc định là hai trục đầu tiên.

Reshape (bool, optional): Nếu định dạng lại là True, hình dạng đầu ra được điều chỉnh để mảng đầu vào được chứa hoàn toàn trong đầu ra. Mặc định là True.

Output (mảng, dtype hoặc tùy chọn): Mảng để đặt đầu ra hoặc kiểu của mảng được trả về. Theo mặc định, một mảng có cùng loại với đầu vào sẽ được tạo.

Oder (int, tùy chọn): Thứ tự của nội suy spline, mặc định là 3. Thứ tự phải nằm

trong phạm vi 0 – 5.

Mode {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}, optional: Tham số chế độ xác định cách mảng đầu vào được mở rộng ra ngoài ranh giới của nó. Mặc định là 'hằng số'. Hành vi cho mỗi giá trị hợp lệ như sau (xem các ô bổ sung và chi tiết về chế độ ranh giới):

+ ‘Reflect’: ‘Phản xạ’ (d c b a | a b c d | d c b a): Đầu vào được mở rộng bằng cách phản ánh về cạnh của pixel cuối cùng. Chế độ này đôi khi cũng được gọi là đối xứng nửa mẫu.

+ ‘Grid-mirror’: ‘Gương lưới’: Đây là một từ đồng nghĩa với 'phản ánh'.

+ ‘Constant’: ‘Hằng số’ (k k k k | a b c d | k k k k): Đầu vào được mở rộng bằng cách điền vào tất cả các giá trị ngoài cạnh với cùng một giá trị không đổi, được xác định bởi tham số cval. Không có phép nội suy nào được thực hiện ngoài các cạnh của đầu vào.

+ ‘Grid-constant’: ‘Hằng số lưới’ (k k k k | a b c d | k k k k): Đầu vào được mở rộng bằng cách điền vào tất cả các giá trị ngoài cạnh với cùng một giá trị không đổi, được xác định bởi tham số cval. Nội suy cũng xảy ra đối với các mẫu nằm ngoài phạm vi đầu vào.

+ ‘Nearest’: ‘Gần nhất’ (a a a a | a b c d | d d d d): Đầu vào được mở rộng bằng cách sao chép pixel cuối cùng.

+ ‘Mirror’ (d c b | a b c d | c b a): Đầu vào được mở rộng bằng cách phản ánh về tâm của pixel cuối cùng. Chế độ này đôi khi cũng được gọi là đối xứng toàn bộ mẫu.

+ ‘Grid-wrap’: ‘Lưới quấn’ (a b c d | a b c d | a b c d): Đầu vào được mở rộng bằng cách quấn quanh cạnh đối diện.

+ ‘Wrap’: ‘Quấn’ (d b c d | a b c d | b c a b): Đầu vào được mở rộng bằng cách quấn quanh cạnh đối diện, nhưng theo cách sao cho điểm cuối cùng và điểm ban đầu trùng nhau chính xác. Trong trường hợp này, người ta không xác định rõ mẫu nào sẽ được chọn tại điểm trùng lắp.

Cval (vô hướng, tùy chọn): Giá trị để lấp đầy các cạnh trước đây của đầu vào nếu chế độ là 'không đổi'. Mặc định là 0,0.

Prefilter (bool, tùy chọn): Xác định xem mảng đầu vào có được lọc trước bằng spline_filter trước khi nội suy hay không. Giá trị mặc định là True, sẽ tạo một mảng float64 tạm thời gồm các giá trị được lọc nếu thứ tự >1. Nếu đặt giá trị này thành Sai, đầu ra sẽ hơi mờ nếu thứ tự >1, trừ khi đầu vào được lọc trước, tức là nó là kết quả của việc gọi spline_filter trên đầu vào ban đầu.

Kết quả trả về

Rotate (ndarray): Đầu vào đã bị xoay.

2.4. Affine_transform

scipy.ndimage.affine_transform(input, matrix, offset=0.0, output_shape=None, output=None, order=3, mode='constant', cval=0.0, prefilter=True): Áp dụng một phép biến đổi affine. Cho một vectơ chỉ số pixel của hình ảnh đầu ra 0, giá trị pixel được xác định từ hình ảnh đầu vào ở vị trí np.dot (matrix, 0) + offset. Thao tác này thực hiện lấy mẫu lại ‘pull’ (hoặc ‘back’), chuyển đổi không gian đầu ra thành đầu vào để định vị dữ liệu. Các phép biến đổi liên kết thường được mô tả theo hướng ‘đẩy’ (hoặc ‘tiến’), biến đổi đầu vào thành đầu ra. Nếu bạn có ma trận cho phép chuyển đổi ‘push’, hãy sử dụng nghịch đảo của nó (numpy.linalg.inv) trong hàm này.

Tham số:

Input (array_like): Ngõ vào là một mảng

Matrix (ndarray): Ma trận chuyển đổi tọa độ nghịch đảo, ánh xạ tọa độ đầu ra thành tọa độ đầu vào. Nếu ndim là số thứ nguyên của đầu vào, thì ma trận đã cho phải có một trong các hình dạng sau:

(ndim, ndim): ma trận biến đổi tuyến tính cho mỗi tọa độ đầu ra.

(ndim,): giả sử rằng ma trận biến đổi 2-D là đường chéo, với đường chéo xác định bởi giá trị đã cho. Sau đó, một thuật toán hiệu quả hơn được sử dụng để khai thác khả

năng phân tách của vấn đề.

(ndim + 1, ndim + 1): giả sử rằng phép biến đổi được chỉ định bằng cách sử dụng tọa độ thuần nhất [1]. Trong trường hợp này, bất kỳ giá trị nào được chuyển sang offset đều bị bỏ qua.

(ndim, ndim + 1): như trên, nhưng hàng dưới cùng của ma trận biến đổi thuần nhất luôn là [0, 0, ..., 1], và có thể bị bỏ qua.

Offset (float, chuỗi hoặc tùy chọn): Phần bù vào mảng mà phép biến đổi được áp dụng. Nếu một float, offset là giống nhau cho mỗi trục. Nếu một chuỗi, phần bù phải chứa một giá trị cho mỗi trục.

output_shape (tuple of ints, optional): Dạng tuple

Output (mảng, dtype hoặc tùy chọn): Mảng để đặt đầu ra hoặc kiểu của mảng được trả về. Theo mặc định, một mảng có cùng loại với đầu vào sẽ được tạo.

Oder (int, tùy chọn): Thứ tự của nội suy spline, mặc định là 3. Thứ tự phải nằm trong phạm vi 0 – 5.

Mode {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}, optional: Tham số chế độ xác định cách mảng đầu vào được mở rộng ra ngoài ranh giới của nó. Mặc định là 'hằng số'. Hành vi cho mỗi giá trị hợp lệ như sau (xem các ô bổ sung và chi tiết về chế độ ranh giới):

+ ‘Reflect’: ‘Phản xạ’ (d c b a | a b c d | d c b a): Đầu vào được mở rộng bằng cách phản ánh về cạnh của pixel cuối cùng. Chế độ này đôi khi cũng được gọi là đối xứng nửa mẫu.

+ ‘Grid-mirror’: ‘Gương lưới’: Đây là một từ đồng nghĩa với 'phản ánh'.

+ ‘Constant’: ‘Hằng số’ (k k k k | a b c d | k k k k): Đầu vào được mở rộng bằng cách điền vào tất cả các giá trị ngoài cạnh với cùng một giá trị không đổi, được xác định bởi tham số cval. Không có phép nội suy nào được thực hiện ngoài các cạnh của đầu vào.

+ ‘Grid-constant’: ‘Hằng số lưới’ (k k k k | a b c d | k k k k): Đầu vào được mở

rộng bằng cách điền vào tất cả các giá trị ngoài cạnh với cùng một giá trị không đổi, được xác định bởi tham số cval. Nội suy cũng xảy ra đối với các mẫu nằm ngoài phạm vi đầu vào.

+ 'Nearest': 'Gần nhất' (a a a a | a b c d | d d d d): Đầu vào được mở rộng bằng cách sao chép pixel cuối cùng.

+ 'Mirror' (d c b | a b c d | c b a): Đầu vào được mở rộng bằng cách phản ánh về tâm của pixel cuối cùng. Chế độ này đôi khi cũng được gọi là đối xứng toàn bộ mẫu.

+ 'Grid-wrap': 'Lưới quấn' (a b c d | a b c d | a b c d): Đầu vào được mở rộng bằng cách quấn quanh cạnh đối diện.

+ 'Wrap': 'Quấn' (d b c d | a b c d | b c a b): Đầu vào được mở rộng bằng cách quấn quanh cạnh đối diện, nhưng theo cách sao cho điểm cuối cùng và điểm ban đầu trùng nhau chính xác. Trong trường hợp này, người ta không xác định rõ mẫu nào sẽ được chọn tại điểm trùng lắp.

Cval (vô hướng, tùy chọn): Giá trị để lấp đầy các cạnh trước đây của đầu vào nếu chế độ là 'không đổi'. Mặc định là 0,0.

Prefilter (bool, tùy chọn): Xác định xem mảng đầu vào có được lọc trước bằng spline_filter trước khi nội suy hay không. Giá trị mặc định là True, sẽ tạo một mảng float64 tạm thời gồm các giá trị được lọc nếu thứ tự> 1. Nếu đặt giá trị này thành Sai, đầu ra sẽ hơi mờ nếu thứ tự> 1, trừ khi đầu vào được lọc trước, tức là nó là kết quả của việc gọi spline_filter trên đầu vào ban đầu.

Kết quả trả về

affine_transform (ndarray): Đầu vào đã bị biến đổi.

2.5. Zoom

scipy.ndimage.zoom(input, zoom, output=None, order=3, mode='constant', cval=0. 0, prefilter=True, *, grid_mode=False): Thu phóng một mảng. Mảng được thu phóng bằng cách sử dụng nội suy spline của thứ tự được yêu cầu.

Tham số:

Input (array_like): Ngõ vào là một mảng

Zoom (float hoặc chuỗi): Hệ số thu phóng dọc theo các trục. Nếu một số thực, thu phóng sẽ giống nhau cho mỗi trục. Nếu là một chuỗi, thu phóng phải chứa một giá trị cho mỗi trục.

Output (mảng, dtype hoặc tùy chọn): Mảng để đặt đầu ra hoặc kiểu của mảng được trả về. Theo mặc định, một mảng có cùng loại với đầu vào sẽ được tạo.

Oder (int, tùy chọn): Thứ tự của nội suy spline, mặc định là 3. Thứ tự phải nằm trong phạm vi 0 – 5.

Mode {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}, optional: Tham số chế độ xác định cách mảng đầu vào được mở rộng ra ngoài ranh giới của nó. Mặc định là 'hằng số'. Hành vi cho mỗi giá trị hợp lệ như sau (xem các ô bổ sung và chi tiết về chế độ ranh giới):

+ ‘Reflect’: ‘Phản xạ’ (d c b a | a b c d | d c b a): Đầu vào được mở rộng bằng cách phản ánh về cạnh của pixel cuối cùng. Chế độ này đôi khi cũng được gọi là đối xứng nửa mẫu.

+ ‘Grid-mirror’: ‘Gương lưới’: Đây là một từ đồng nghĩa với 'phản ánh'.

+ ‘Constant’: ‘Hằng số’ (k k k k | a b c d | k k k k): Đầu vào được mở rộng bằng cách điền vào tất cả các giá trị ngoài cạnh với cùng một giá trị không đổi, được xác định bởi tham số cval. Không có phép nội suy nào được thực hiện ngoài các cạnh của đầu vào.

+ ‘Grid-constant’: ‘Hằng số lưới’ (k k k k | a b c d | k k k k): Đầu vào được mở rộng bằng cách điền vào tất cả các giá trị ngoài cạnh với cùng một giá trị không đổi, được xác định bởi tham số cval. Nội suy cũng xảy ra đối với các mẫu nằm ngoài phạm vi đầu vào.

+ ‘Nearest’: ‘Gần nhất’ (a a a a | a b c d | d d d d): Đầu vào được mở rộng bằng cách sao chép pixel cuối cùng.

+ ‘Mirror’ (d c b | a b c d | c b a): Đầu vào được mở rộng bằng cách phản ánh về tâm của pixel cuối cùng. Chế độ này đôi khi cũng được gọi là đối xứng toàn bộ mẫu.

+ ‘Grid-wrap’: ‘Lưới quấn’ (a b c d | a b c d | a b c d): Đầu vào được mở rộng bằng cách quấn quanh cạnh đối diện.

+ ‘Wrap’: ‘Quấn’ (d b c d | a b c d | b c a b): Đầu vào được mở rộng bằng cách quấn quanh cạnh đối diện, nhưng theo cách sao cho điểm cuối cùng và điểm ban đầu trùng nhau chính xác. Trong trường hợp này, người ta không xác định rõ mẫu nào sẽ được chọn tại điểm trùng lắp.

Cval (vô hướng, tùy chọn): Giá trị để lấp đầy các cạnh trước đây của đầu vào nếu chế độ là 'không đổi'. Mặc định là 0,0.

Prefilter (bool, tùy chọn): Xác định xem mảng đầu vào có được lọc trước bằng spline_filter trước khi nội suy hay không. Giá trị mặc định là True, sẽ tạo một mảng float64 tạm thời gồm các giá trị được lọc nếu thứ tự> 1. Nếu đặt giá trị này thành Sai, đầu ra sẽ hơi mờ nếu thứ tự> 1, trừ khi đầu vào được lọc trước, tức là nó là kết quả của việc gọi spline_filter trên đầu vào ban đầu.

grid_mode (bool, optional): Nếu False, khoảng cách từ các tâm pixel sẽ được thu phóng. Nếu không, khoảng cách bao gồm toàn bộ pixel sẽ được sử dụng. Ví dụ: tín hiệu 1d có độ dài 5 được coi là có độ dài 4 khi chế độ lưới là Sai, nhưng độ dài 5 khi chế độ lưới là True. Điểm bắt đầu của mũi tên trong sơ đồ trên tương ứng với vị trí tọa độ 0 trong mỗi chế độ.

Kết quả trả về

Zoom (ndarray): Đầu vào đã thu phóng.

2.6. Gaussian_filter

scipy.ndimage.gaussian_filter(input, sigma, order=0, output=None, mode='reflect', cval=0.0, truncate=4.0): Bộ lọc Gaussian đa chiều.

Tham số:

Input (array_like): Ngõ vào là một mảng

Sigma (scalar hoặc chuỗi scalar): Độ lệch chuẩn cho hạt nhân Gaussian. Độ lệch chuẩn của bộ lọc Gaussian được cung cấp cho mỗi trục dưới dạng một chuỗi hoặc một số duy nhất, trong trường hợp đó, nó bằng nhau cho tất cả các trục.

Output (mảng, dtype hoặc tùy chọn): Mảng để đặt đầu ra hoặc kiểu của mảng được trả về. Theo mặc định, một mảng có cùng loại với đầu vào sẽ được tạo.

Oder (int, chuỗi ints hoặc tùy chọn): Thứ tự của bộ lọc dọc theo mỗi trục được cung cấp dưới dạng một chuỗi các số nguyên hoặc một số duy nhất. Thứ tự 0 tương ứng với tích chập với hạt nhân Gaussian. Một thứ tự dương tương ứng với tích chập với đạo hàm của một Gaussian.

Mode {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}, optional: Tham số chế độ xác định cách mảng đầu vào được mở rộng ra ngoài ranh giới của nó. Mặc định là 'hằng số'. Hành vi cho mỗi giá trị hợp lệ như sau (xem các ô bổ sung và chi tiết về chế độ ranh giới):

+ ‘Reflect’: ‘Phản xạ’ (d c b a | a b c d | d c b a): Đầu vào được mở rộng bằng cách phản ánh về cạnh của pixel cuối cùng. Chế độ này đôi khi cũng được gọi là đối xứng nửa mẫu.

+ ‘Grid-mirror’: ‘Gương lưới’: Đây là một từ đồng nghĩa với ‘phản ánh’.

+ ‘Constant’: ‘Hằng số’ (k k k k | a b c d | k k k k): Đầu vào được mở rộng bằng cách điền vào tất cả các giá trị ngoài cạnh với cùng một giá trị không đổi, được xác định bởi tham số cval. Không có phép nội suy nào được thực hiện ngoài các cạnh của đầu vào.

+ ‘Grid-constant’: ‘Hằng số lưới’ (k k k k | a b c d | k k k k): Đầu vào được mở rộng bằng cách điền vào tất cả các giá trị ngoài cạnh với cùng một giá trị không đổi, được xác định bởi tham số cval. Nội suy cũng xảy ra đối với các mẫu nằm ngoài phạm vi đầu vào.

+ ‘Nearest’: ‘Gần nhất’ (a a a a | a b c d | d d d d): Đầu vào được mở rộng bằng

cách sao chép pixel cuối cùng.

+ 'Mirror' (d c b | a b c d | c b a): Đầu vào được mở rộng bằng cách phản ánh về tâm của pixel cuối cùng. Chế độ này đôi khi cũng được gọi là đối xứng toàn bộ mẫu.

+ 'Grid-wrap': 'Lưới quấn' (a b c d | a b c d | a b c d): Đầu vào được mở rộng bằng cách quấn quanh cạnh đối diện.

+ 'Wrap': 'Quấn' (d b c d | a b c d | b c a b): Đầu vào được mở rộng bằng cách quấn quanh cạnh đối diện, nhưng theo cách sao cho điểm cuối cùng và điểm ban đầu trùng nhau chính xác. Trong trường hợp này, người ta không xác định rõ mẫu nào sẽ được chọn tại điểm trùng lắp.

Cval (vô hướng, tùy chọn): Giá trị để lấp đầy các cạnh trước đây của đầu vào nếu chế độ là 'không đổi'. Mặc định là 0,0.

Truncate (float): Cắt bớt bộ lọc ở nhiều độ lệch chuẩn này. Mặc định là 4.0.

Kết quả trả về

Gaussian_filter (ndarray): Trả về mảng có cùng hình dạng với đầu vào.

2.7. Range

numpy.arange([start,]stop, [step,]dtype=None, *, like=None): Trả về các giá trị cách đều nhau trong một khoảng thời gian nhất định. Các giá trị được tạo trong khoảng thời gian nửa mở [bắt đầu, dừng) (nói cách khác, khoảng bao gồm bắt đầu nhưng không bao gồm dừng). Đối với các đối số số nguyên, hàm tương đương với hàm dải ô tích hợp sẵn trong Python, nhưng trả về một ndarray chứ không phải là một danh sách. Khi sử dụng bước không phải số nguyên, chẳng hạn như 0,1, thường tốt hơn là sử dụng numpy.linspace. Xem phần cảnh báo bên dưới để biết thêm thông tin.

Tham số:

Start (integer hoặc real, tùy chọn): Khoảng bắt đầu. Khoảng bao gồm giá trị này. Giá trị bắt đầu mặc định là 0.

Stop (integer hoặc thực): Kết thúc khoảng thời gian. Khoảng không bao gồm giá trị này, ngoại trừ một số trường hợp khi bước không phải là số nguyên và việc làm tròn dấu phẩy động ảnh hưởng đến độ dài của bước ra.

Step (integer hoặc real, tùy chọn): Khoảng cách giữa các giá trị. Đối với bất kỳ đầu ra nào, đây là khoảng cách giữa hai giá trị liền kề, out [i + 1] - out [i]. Kích thước bước mặc định là 1. Nếu bước được chỉ định làm đối số vị trí, thì bắt đầu cũng phải được cung cấp.

Dtype (dtype): Kiểu của mảng đầu ra. Nếu dtype không được đưa ra, hãy suy ra kiểu dữ liệu từ các đối số đầu vào khác.

Like (array_like): Đối tượng tham chiếu để cho phép tạo mảng không phải là mảng NumPy. Nếu một mảng giống như được truyền vào tương tự hỗ trợ giao thức __array_ Chức năng__, kết quả sẽ được xác định bởi nó. Trong trường hợp này, nó đảm bảo việc tạo một đối tượng mảng tương thích với đối tượng được truyền vào thông qua đối số này.

Kết quả trả về:

Arrange (ndarray): Mảng các giá trị cách đều nhau. Đối với các đối số dấu phẩy động, độ dài của kết quả là ceil ((dừng - bắt đầu) / bước). Do tràn dấu phẩy động, quy tắc này có thể dẫn đến phần tử cuối cùng lớn hơn điểm dừng.

2.8. Reshape

numpy.reshape(a, newshape, order='C'): Cung cấp một hình dạng mới cho một mảng mà không thay đổi dữ liệu của nó.

Tham số:

A (array_like): Mảng được định hình lại.

Newshape (int hoặc tuple): Hình dạng mới phải tương thích với hình dạng ban đầu. Nếu là một số nguyên, thì kết quả sẽ là một mảng 1-D có độ dài đó. Một kích thước hình dạng có thể là -1. Trong trường hợp này, giá trị được suy ra từ độ dài của mảng và các

kích thước còn lại.

Order {‘C’, ‘F’, ‘A’} (tùy chọn): Đọc các phần tử của một bằng cách sử dụng thứ tự chỉ mục này và đặt các phần tử vào mảng được định dạng lại bằng cách sử dụng thứ tự chỉ mục này. ‘C’ có nghĩa là đọc / ghi các phần tử bằng cách sử dụng thứ tự chỉ mục giống C, với chỉ số trục cuối cùng thay đổi nhanh nhất, quay lại chỉ số trục đầu tiên thay đổi chậm nhất. ‘F’ có nghĩa là đọc / ghi các phần tử bằng cách sử dụng thứ tự chỉ mục giống Fortran, với chỉ mục đầu tiên thay đổi nhanh nhất và chỉ mục cuối cùng thay đổi chậm nhất. Lưu ý rằng các tùy chọn ‘C’ và ‘F’ không tính đến bố cục bộ nhớ của mảng bên dưới và chỉ tham chiếu đến thứ tự lập chỉ mục. ‘A’ có nghĩa là đọc / ghi các phần tử theo thứ tự chỉ mục giống Fortran nếu a là Fortran liền kề trong bộ nhớ, ngược lại là thứ tự giống C.

Kết quả trả về:

reshaped_array (ndarray): Đây sẽ là một đối tượng xem mới nếu có thể; nếu không, nó sẽ là một bản sao. Lưu ý rằng không có gì đảm bảo về bố cục bộ nhớ (C- hoặc Fortran- liền kề) của mảng được trả về.

2.9. Mean

numpy.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>, *, where=<no value>): Tính toán trung bình cộng dọc theo trục xác định. Trả về giá trị trung bình của các phần tử mảng. Giá trị trung bình được lấy trên mảng phẳng theo mặc định, nếu không thì trên trục được chỉ định. Các giá trị trung gian và trả về float64 được sử dụng cho các đầu vào số nguyên.

Tham số:

A (array_like): Mảng chứa các số có giá trị trung bình được mong muốn. Nếu a không phải là một mảng, một chuyển đổi sẽ được cố gắng thực hiện.

Axis (None hoặc int hoặc nhiều int, tùy chọn): Trục hoặc các trục dọc theo đó các phương tiện được tính toán. Mặc định là tính giá trị trung bình của mảng phẳng. Nếu đây

là một bộ int, một giá trị trung bình được thực hiện trên nhiều trục, thay vì một trục hoặc tất cả các trục như trước đây.

Dtype (data-type, tùy chọn): Nhập để sử dụng trong tính toán giá trị trung bình. Đối với đầu vào số nguyên, mặc định là float64; đối với đầu vào dấu chấm động, nó giống với kiểu đầu vào.

Out (ndarray, tùy chọn): Mảng đầu ra thay thế để đặt kết quả. Mặc định là Không có; nếu được cung cấp, nó phải có cùng hình dạng với đầu ra dự kiến, nhưng kiểu sẽ được đúc nếu cần thiết. Xem phần Xác định loại đầu ra để biết thêm chi tiết.

Keepdims (bool, tùy chọn): Nếu điều này được đặt thành True, các trục được giảm xuống sẽ được giữ lại trong kết quả là các kích thước có kích thước bằng một. Với tùy chọn này, kết quả sẽ phát chính xác so với mảng đầu vào. Nếu giá trị mặc định được chuyển, thì keepdims sẽ không được chuyển qua phương thức trung bình của các lớp con của ndarray, tuy nhiên, bất kỳ giá trị không mặc định nào cũng sẽ như vậy. Nếu phương thức của lớp con không triển khai keepdims thì bất kỳ ngoại lệ nào sẽ được đưa ra.

Where (array_like of bool, tùy chọn): Các yếu tố cần đưa vào giá trị trung bình. Xem giảm để biết chi tiết.

Kết quả trả về:

M (ndarray, dtype): Nếu out = None, trả về một mảng mới chứa các giá trị trung bình, nếu không, một tham chiếu đến mảng đầu ra sẽ được trả về.

2.10. Abs

numpy.absolute(x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj]) = <ufunc 'absolute'>: Tính giá trị tuyệt đối element-wise. np.abs là cách viết tắt của hàm này.

Tham số:

X (array_like): Đầu vào mảng.

Out (ndarray, Không có, hoặc bộ ndarray và Không có, tùy chọn): Vị trí lưu trữ kết quả. Nếu được cung cấp, nó phải có hình dạng mà các đầu vào phát tới. Nếu không được cung cấp hoặc Không có, một mảng mới được cấp phát sẽ được trả về. Một bộ (chỉ có thể làm đối số từ khóa) phải có độ dài bằng số đầu ra.

Where (array_like, tùy chọn): Điều kiện này được phát qua đầu vào. Tại các vị trí mà điều kiện là True, mảng out sẽ được đặt thành kết quả ufunc. Ở những nơi khác, mảng out sẽ giữ nguyên giá trị ban đầu của nó. Lưu ý rằng nếu một mảng out chưa khởi tạo được tạo thông qua mặc định out = None, các vị trí bên trong nó mà điều kiện là False sẽ vẫn chưa được khởi tạo.

Kết quả trả về:

Absolute: Một ndarray chứa giá trị tuyệt đối của mỗi phần tử trong x. Đối với đầu vào phức tạp, a+ib, giá trị tuyệt đối là $\sqrt{a^2+b^2}$. Đây là một đại lượng vô hướng nếu x là một đại lượng vô hướng.

2.11. Where

np.where(condition, x, y): Trả về các phần tử được chọn từ x hoặc y tùy thuộc vào điều kiện.

Tham số:

condition (array_like, bool): Trong đó True, mang lại x, nếu không thì mang lại y.

x, y (array_like): Giá trị để lựa chọn. x, y và điều kiện cần được phát sóng thành một số hình dạng.

Kết quả trả về:

Out (ndarray): Một mảng có các phần tử từ x trong đó điều kiện là Đúng và các phần tử từ y ở nơi khác.

2.12. Logical_and

numpy.logical_and(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj]) = <ufunc 'logical_and'>: Tính giá trị của x1 và x2.

Tham số:

x1, x2 (array_like): Các mảng đầu vào. Nếu x1.shape! = X2.shape, chúng phải được truyền phát thành một hình dạng chung (trở thành hình dạng của đầu ra).

Out (ndarray, hoặc bộ ndarray và Không có, tùy chọn): Vị trí lưu trữ kết quả. Nếu được cung cấp, nó phải có hình dạng mà các đầu vào phát tới. Nếu không được cung cấp hoặc Không có, một mảng mới được cấp phát sẽ được trả về. Một bộ (chỉ có thể làm đối số từ khóa) phải có độ dài bằng số đầu ra.

Where (array_like, tùy chọn): Điều kiện này được phát qua đầu vào. Tại các vị trí mà điều kiện là True, mảng out sẽ được đặt thành kết quả ufunc. Ở những nơi khác, mảng out sẽ giữ nguyên giá trị ban đầu của nó. Lưu ý rằng nếu một mảng out chưa khởi tạo được tạo thông qua mặc định out = None, các vị trí bên trong nó mà điều kiện là False sẽ vẫn chưa được khởi tạo.

Kết quả trả về:

Y (ndarray hoặc bool): Kết quả Boolean của phép toán logic AND được áp dụng cho các phần tử của x1 và x2; hình dạng được xác định bằng cách phát sóng. Đây là một vô hướng nếu cả x1 và x2 đều là vô hướng.

2.13. Logical_or

numpy.logical_or(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj]) = <ufunc 'logical_or'>: Tính giá trị của x1 OR x2.

Tham số:

x1, x2 (array_like): OR logic được áp dụng cho các phần tử của x1 và x2. Nếu x1.shape! = X2.shape, chúng phải được truyền phát thành một hình dạng chung (trở

thành hình dạng của đầu ra).

Out (ndarray hoặc bộ ndarray và Không có, tùy chọn): Vị trí lưu trữ kết quả. Nếu được cung cấp, nó phải có hình dạng mà các đầu vào phát tới. Nếu không được cung cấp hoặc Không có, một mảng mới được cấp phát sẽ được trả về. Một bộ (chỉ có thể làm đối số từ khóa) phải có độ dài bằng số đầu ra.

Where (array_like, tùy chọn): Điều kiện này được phát qua đầu vào. Tại các vị trí mà điều kiện là True, mảng out sẽ được đặt thành kết quả ufunc. Ở những nơi khác, mảng out sẽ giữ nguyên giá trị ban đầu của nó. Lưu ý rằng nếu một mảng out chưa khởi tạo được tạo thông qua mặc định out = None, các vị trí bên trong nó mà điều kiện là False sẽ vẫn chưa được khởi tạo.

Kết quả trả về:

Y (ndarray hoặc bool): Kết quả Boolean của phép toán OR logic được áp dụng cho các phần tử của x1 và x2; hình dạng được xác định bằng cách phát sóng. Đây là một vô hướng nếu cả x1 và x2 đều là vô hướng.

2.14. Label

scipy.ndimage.label(input, structure=None, output=None): Gắn nhãn các tính năng trong một mảng.

Thông số:

Input (array_like): Một đối tượng giống như mảng được gắn nhãn. Mọi giá trị khác 0 trong đầu vào đều được tính là đối tượng xử lý và giá trị 0 được coi là nền.

Structure (array_like): Một phần tử cấu trúc xác định các kết nối tính năng. cấu trúc phải đối xứng. Nếu không có phần tử cấu trúc nào được cung cấp, một phần tử được tạo tự động có cấu trúc mặc định là:

0 1 0

1 1 0

0 1 0

Output (None, data-type, array_like): Nếu đầu ra là một kiểu dữ liệu, nó chỉ định kiểu

của mảng tính năng được gắn nhãn kết quả. Nếu đầu ra là một đối tượng giống mảng, thì đầu ra sẽ được cập nhật với các tính năng được gắn nhãn từ hàm này. Hàm này có thể hoạt động tại chỗ, bằng cách chuyển đầu ra = đầu vào.

Kết quả trả về:

Label (ndarray or int): Một mảng số nguyên trong đó mỗi tính năng duy nhất trong đầu vào có một nhãn duy nhất trong mảng được trả về.

Num_features (int): Có bao nhiêu giá trị đã được tìm thấy. Nếu kết quả đầu ra là none, hàm này trả về một bộ giá trị (label_array , num_features). Nếu đầu ra là một ndarray, thì nó sẽ được cập nhật các giá trị trong label_array và chỉ num_features được hàm này trả về.

3. EXERCISE

3.1. Tính sự khác biệt (err) giữa 2 ảnh?

- Bước 1: Chọn ảnh làm tập dữ liệu

Hình 3. 1. a. Ảnh bên trái: Ảnh Ref – b. Ảnh bên phải: Ảnh Test

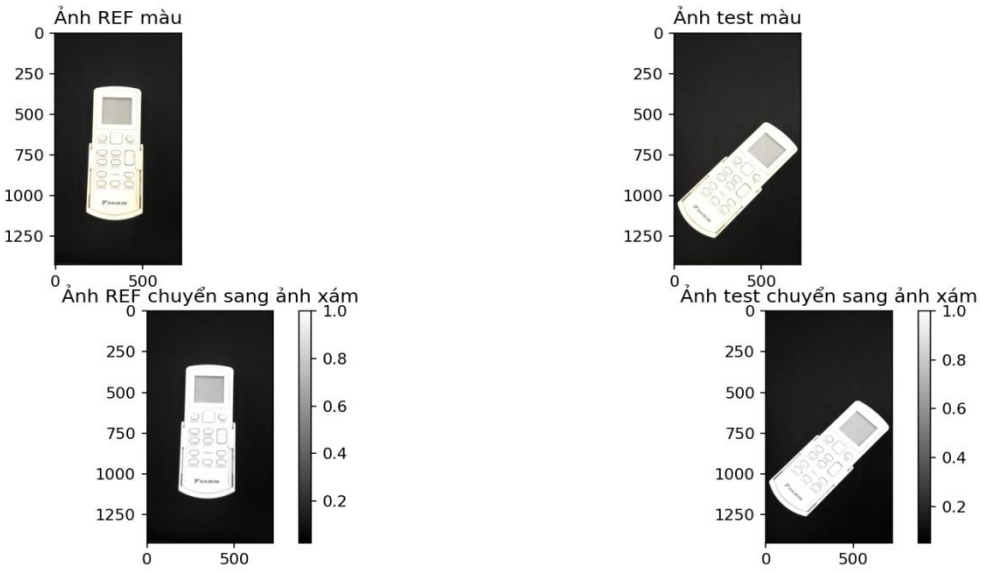
- Bước 2: Khai báo thư viện

```
import imageio
import numpy as np
import scipy.ndimage as ndi
import matplotlib.pyplot as plt
from skimage import color
from skimage import io
```

Code lấy các thư viện sử dụng

- Bước 3: Đọc dữ liệu và chuyển thành ảnh Gray

```
im1_goc = imageio.imread('D:\HK2_2021\TT_XLA\CUOI KY\image1.jpg')
im2_goc = imageio.imread('D:\HK2_2021\TT_XLA\CUOI KY\image2.jpg')
im1 = color.rgb2gray(io.imread('D:\HK2_2021\TT_XLA\CUOI KY\image1.jpg'))
im2 = color.rgb2gray(io.imread('D:\HK2_2021\TT_XLA\CUOI KY\image2.jpg'))
```

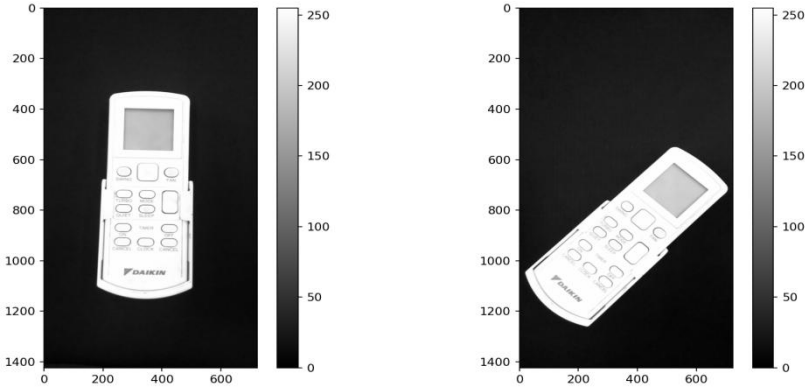


Hình 3. 2. Ảnh gốc và ảnh chuyển sang ảnh

- Bước 4: Thay đổi cường độ mức xám của 2 ảnh:

```
image1=np.zeros(im1.shape, dtype='uint8')
image2=np.zeros(im2.shape, dtype='uint8')
print(image1.shape, image2.shape)
min1=im1.min()
max1=im1.max()
min2=im2.min()
max2=im2.max()
print("Hệ số:", min1, max1, min2, max2)
for i in range(im1.shape[0]):
    for j in range(im1.shape[1]):
        r=im1[i,j]
        image1[i,j]=r*(255/(1-min1))+((255*min1)/(min1-1))
for i in range(im2.shape[0]):
    for j in range(im2.shape[1]):
        r=im2[i,j]
        image2[i,j]=r*(255/(1-min2))+((255*min2)/(min2-1))
```

Code thay đổi mức độ xám

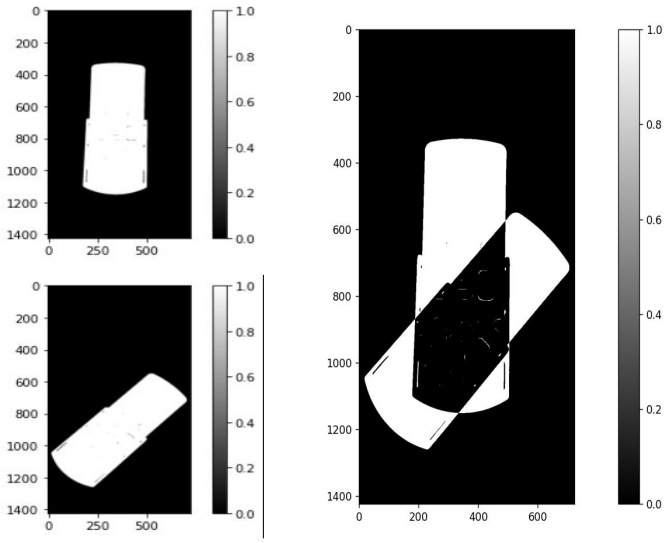


Hình 3. 3. Ảnh thay đổi mức xám từ 0 – 255

- Bước 5: Chuyển đổi thành ảnh Binary và tính giá trị err và ảnh khác biệt:

```
# a: Tinh ERR RAW
# Convert binary
def convert_bin(q):
    mask=q>100
    return mask
im_bin1= convert_bin(image1)
im_bin2= convert_bin(image2)

# Calculate absolute image difference
abs_err = np.abs(im_bin1.astype('int') - im_bin2.astype('int'))


# Calculate mean absolute error
mean_abs_err = np.mean(abs_err)
```

Code chuyển đổi ảnh bin và tính giá trị Err và ảnh khác biệt



Hình 3. 4. a. 2 ảnh đã chuyển đổi sang Binary – b. Ảnh khác biệt

3.2. Dịch tâm ảnh Test theo tâm của ảnh Ref. Tính lại err?

- Bước 1: Tìm tâm (ảnh Ref) và dịch

```
# b: Tim tam va dich
mask1 = np.where(image1>100, 1, 0)
# Label the objects in "mask"
labels1, nlabels1 = ndi.label(mask1)
print('Num. Labels:', nlabels1)
# Find image center of mass
com1 = ndi.center_of_mass(labels1, labels1, index=1)
```

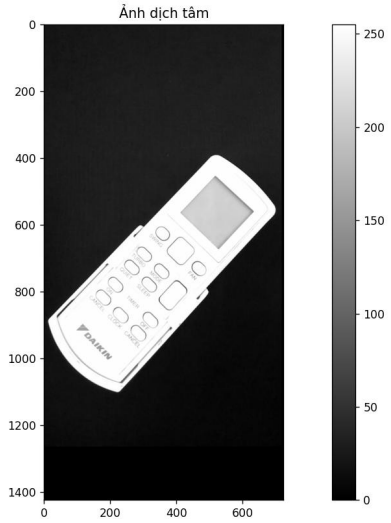
Code tìm tâm ảnh Ref

```
mask2 = np.where(image2>100, 1, 0)
# Label the objects in "mask"
labels2, nlabels2 = ndi.label(mask2)
print('Num. Labels:', nlabels2)
# Find image center of mass
com2 = ndi.center_of_mass(labels2, labels2, index=1)

# Calculate amount of shift needed
d02 = com1[0]-com2[0]
d12 = com1[1]-com2[1]
print(com2[0], com2[1], d02, d12)
```

Code tìm khoảng cách để dịch

```
im_trans = ndi.shift(image2, shift=(d02, d12))
mask = np.where(im_trans>100, 1, 0)
labels, nlabels = ndi.label(mask)
print('Num. Labels:', nlabels)
com = ndi.center_of_mass(labels, labels, index=1)

print(com[0], com[1])
```

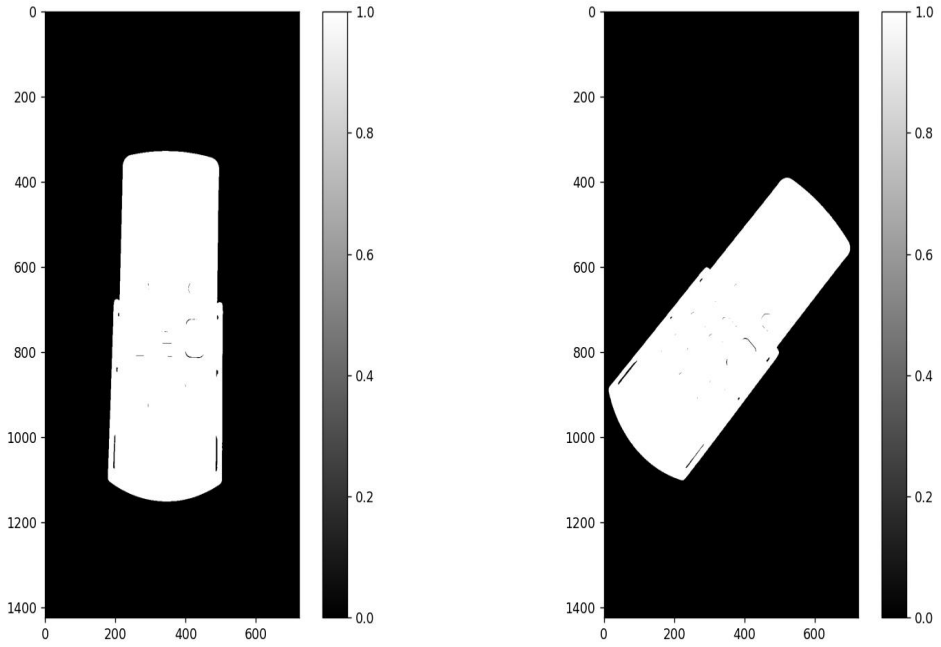
Code dịch ảnh và kiểm tra tọa độ so với ảnh Ref



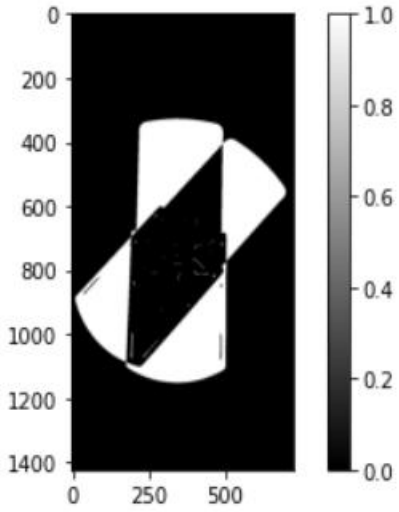
Hình 3. 5. Ảnh dịch tâm của ảnh gốc

```
im_bin2_l2= convert_bin(im_trans)

# Calculate absolute image difference
abs_err_l2 = np.absolute(im_bin1.astype('int')- im_bin2_l2.astype('int'))

# Calculate mean absolute error
mean_abs_err_l2 = np.mean(np.abs(im_bin1.astype('int') - im_bin2_l2.astype('int')))
print('MAE:', mean_abs_err_l2)
```

Code tính lại giá trị Err và ảnh khác biệt



Hình 3. 6. Ảnh (Binary) của ảnh Test đã dịch tâm theo ảnh gốc



Hình 3. 7. Ảnh binary về sự khác biệt của 2 ảnh

3.3. Xoay ảnh theo chiều kim đồng hồ mỗi lần 1 độ. Tìm err nhỏ nhất và index của err đó?

- Bước 1: Tạo vòng lặp for dịch ảnh 360 độ (1 vòng tròn 360 độ). Lưu giá trị đã tính vào list trống đã tạo sẵn

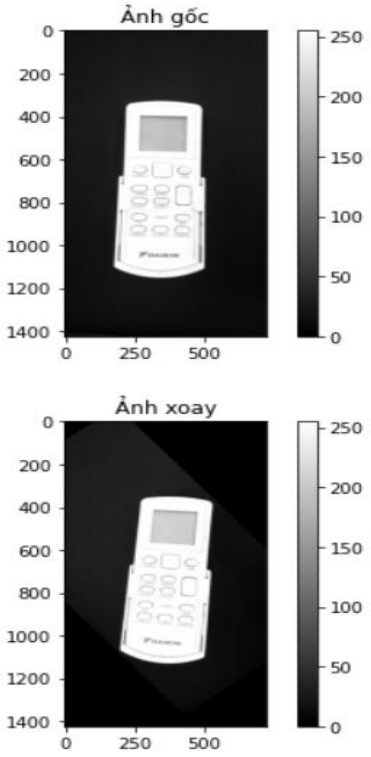
```
mae=[]
for i in range(360):
    im_ro = ndi.rotate(im_trans, angle=-i, reshape=False)
    im_bin2_new = convert_bin(im_ro)
    mean_abs_err_l2 = np.mean(np.abs(im_bin1.astype('int') - im_bin2_new.astype('int')))
    mae.append(mean_abs_err_l2)
    print('MAE:',[i, mean_abs_err_l2])
```

Code xoay ảnh 360 độ

- Bước 2: Thể hiện ảnh sau khi xoay và ảnh Ref

```
im_ro = ndi.rotate(im_trans, angle=-321, reshape=False)
plt.figure()
plt.imshow(image1, cmap='gray')
plt.colorbar()
plt.title("Ảnh gốc")
plt.figure()
plt.imshow(im_ro, cmap='gray')
plt.colorbar()
plt.title("Ảnh xoay")
```

Code xoay ảnh



Hình 3. 8. Ảnh xoay sau khi tìm ra giá trị MAE (mean abs err) nhỏ nhất