 A building with a triangular shape

Description automatically generated with medium confidence A white surface with a triangle

Description automatically generated 

(a) (b) (c) (d)

***Hình 1.10***: Một ví dụ về tình toán các đạo hàm hình ảnh bằng cách sử dụng các bộ lọc phái sinh Sobel. (A) ảnh gốc theo thang độ xám, (b) đạo hàm x, (c) đạo hàm y, (d) độ lớn độ dốc.

Trong ví dụ này việc tính toán đạo hàm theo phương x, y và độ lớn của vectơ gradient bằng bộ lọc Sobel. Đối số thứ hai chọn đạo hàm x hoặc y và thứ ba lưu trữ đầu ra. Hình 1.10 cho thấy 1 hình ảnh với các đạo hàm được tính toán bằng bộ lọc Sobel. Trong hai hình ảnh kết quả đạo hàm, các đạo hàm dương được hiển thị với các pixel sáng và các đạo hàm âm là tối. Vùng màu xám có giá trị gần bằng không

Sử dụng phương pháp này có nhược điểm là các đạo hàm được lấy theo tỷ lệ được xác định bởi độ phân giải hình ảnh. Điều này có thể được khắc phục bằng phương pháp làm mờ hình ảnh bởi nhiều và tính toán các đạo hàm ở bất kỳ tỷ lệ nào, có thể sử dụng bộ lọc Gaussian,

**(1.8)**

Trong đó là đạo hàm x và y của G, hàm Gaussian có độ lệch chuẩn σ.

Hàm tests.gaussian\_filter() đã sử dụng để làm mờ trước đó cũng có thể lấy thêm các độ số để tính toán các đạo hàm Gaussian thay thế. Để kiểm chứng điều này trên một hình ảnh, chỉ cần thực hiện:

|  |
| --- |
| sigma = 5 #standard deviation  imx = zeros(im.shape) filters.gaussian\_filter(im, (sigma, sigma), (0, 1), imx)  imy = zeros(im.shape) filters.gaussian\_filter(im, (sigma, sigma), (1, 0), imy) |

Đối số thứ ba chỉ định thứ tự các đạo hàm sẽ sử dụng theo mỗi hướng bằng cách sử dụng độ lệch chuẩn được xác định bởi đối số thứ hai.

Thứ hai. Hình 1.11 cho thấy các đạo hàm và cường độ gradient cho các tỷ lệ khác nhau. So sánh điều này với độ mờ ở cùng tỷ lệ trong Hình 1.9.

A collage of a building

Description automatically generated

1. (b) (c) (d)

***Hình 1.11:*** Một ví dụ về tính toán các đạo hàm hình ảnh bằng các đạo hàm Gaussian. (trên cùng) x-đạo hàm, (giữa) y-đạo hàm và (dưới) độ lớn gradient. (a) ảnh gốc trong thang độ xám, (b) bộ lọc đạo hàm Gaussian với σ = 2, (c) với σ = 5, (d) với σ = 10.

**Hình thái học đếm các đối tượng**

Hình thái học là một framework và tập hợp các phương pháp xử lý hình ảnh để đo lường và phân tích các hình dạng cơ bản. Hình thái thường được áp dụng cho hình ảnh nhị phân nhưng cũng có thể được sử dụng với thang độ xám. Hình ảnh nhị phân là một hình ảnh trong đó mỗi pixel chỉ có hai giá trị, thường là 0 và 1. Hình ảnh nhị phân thường là kết quả của việc lấy ngưỡng một hình ảnh, và do vậy định đếm các đối tượng hoặc đo kích thước của chúng.

Các toán tử hình thái được trình bày trong scipy.ndimage với module morphology. Các chức năng đếm và đo lường cho hình ảnh nhị phân nằm trong các phép đo mở hình scipy.ndimage. Phân tích một ví dụ đơn giản về cách sử dụng chúng dưới đây.

Xem xét hình ảnh nhị phân trong Hình 1.12a, Để đếm các đối tượng trong hình ảnh đó, sử dụng đoạn chương trình sau:

|  |
| --- |
| from scipy.ndimage import measurements, morphology  # load image and threshold to make sure it is binary  im = array(Image.open("houses.png").convert("L"))  im = 1 \* (im > 128)  labels, nbr\_objects = measurements.label(im)  print("Number of objects:", nbr\_objects) |

Đoạn chương trình này tải hình ảnh và lấy ngưỡng để có hình ảnh nhị phân. Nhân với 1 chuyển đổi mảng boolean thành một ảnh nhị phân. Sau đó, hàm label() tìm các đối tượng riêng lẻ và gán nhãn số nguyên cho pixel theo từng đối tượng thuộc vùng pixel đó. Hình 1.12b hiển thị mảng của các nhãn. Các giá trị mức xám chỉ ra địa chỉ đối tượng. Như có thể thấy, có một số kết nối nhỏ giữa các đối tượng. Sử dụng toán tử mở ảnh (opening) nhị phân có thể loại bỏ những kết nối nhỏ này.

|  |
| --- |
| # morphology: opening to separate objects better  im\_open = morphology.binary\_opening(im, ones((9, 5)), iterations=2)  labels\_open, nbr\_objects\_open = measurements.label(im\_open)  print("Number of objects:", nbr\_objects\_open) |

Để sửa lỗi, hàm binary\_opening() dùng để xác định cấu trúc phần tử dùng trong phép toán hình thái học. Trong trường hợp này, đã sử dụng 9 pixel (4 ở trên, pixel tại chính vị trí của nó và 4 ở dưới) theo hướng y và 5 theo hướng x. Có thể dùng câu lệnh này để định nghĩa cấu trúc bất kỳ dùng trong hình thái học. Tham số iterations xác định số lần áp dụng phép toán trên hình ảnh. Có thể thử nghiệm và xem số lượng đối tượng thay đổi như thế nào trên các ví dụ cụ thể. Hình ảnh sau khi được mở (opening) và hình ảnh nhận tương ứng được hiển thị trong Hình 1.12c và Hình 1.12d.

A group of images of different shapes

Description automatically generated

***Hình 1.12:*** Một ví dụ về hình thái học. Dùng phép toán mở nhị phân để tách các đối tượng và sau đó là đếm chúng. (a) Hình ảnh nhị phân gốc, (b) Nhãn tương ứng với giá trị gốc, mức màu xám biểu thị chỉ số đối tượng, (c) Hình ảnh nhị phân sau khi mở, (d) Nhãn tương ứng với hình ảnh đã mở.

**1.4.3. Các module SciPy hữu ích khác**  
SciPy đi kèm với một số mô hình hữu ích cho đầu vào và đầu ra. Hai trong số đó là io và misc.

Đọc và ghi các tệp .mat: Nếu có một số dữ liệu offline hoặc một số dữ liệu online được lưu trữ ở định dạng tệp Matlab dưới .mat, có thể đọc dữ liệu này bằng module scipy.io. Đây là cách thực hiện:

|  |
| --- |
| data = scipy.io.loadmat('test.mat') |

Bây giờ dữ liệu đối tượng chứa một từ điển với các khóa tương ứng với tên biến được lưu trong tệp .mat. Các biến là trong một định dạng mảng. Việc lưu vào tệp tin .mat cũng đơn giản không kém. Chỉ cần tạo một từ điển với tất cả các biến muốn lưu và sử dụng savemat().

|  |
| --- |
| data = {}  data['x'] = x  scipy.io.savemat('test.mat', data) |

Điều này sẽ lưu mảng x để nó có tên "x" khi đọc vào Matlab. Thông tin thêm về Scipy.io có thể được tìm thấy trong tài liệu trực tuyến: http://docs.scipy.org/doc/scipy/reference/io.html.

**Lưu mảng dưới dạng hình ảnh:** vì chúng ta đang thao tác các giải thuật trên đối tượng hình ảnh và thực hiện tính toán bằng cách sử dụng các đối tượng trong mảng, rất hữu ích khi có thể lưu chúng trực tiếp dưới dạng tệp hình ảnh. Nhiều hình ảnh trong cuốn sách này được tạo ra như sau:

Hàm imsave() có sẵn thông qua module scipy.misc. để lưu một mảng im vào tập tin, chỉ cần làm

|  |
| --- |
| import scipy.misc  scipy.misc.imsave('test.jpg', img) |

Module scipy.misc cũng chữa sẵn dữ liệu nổi tiếng có tên “Lena”.

|  |
| --- |
| Lena = scipy.misc.lena() |

Việc này sẽ cho kết quả là một phiên bản mảng mức xám có kích thước 512x512 của hình ảnh.

**1.5. VÍ DỤ NÂNG CAO: GIẢM NHIỄU HÌNH ẢNH**

Kết thúc chương này với một ví dụ rất hữu ích về khử nhiễu hình ảnh. Giảm nhiễu hình ảnh. Giảm nhiễu hình ảnh là quá trình loại bỏ nhiễu hình ảnh đồng thời cố gắng giữ lại các chi tiết và cấu trúc. Mô hình khử nhiễu Rudin-Osher-Fatemi (ROF)[28] được giới thiệu ở phần này. Loại bỏ nhiễu khỏi hình ảnh rất quan trọng đối với nhiều ứng dụng khác nhau. Mô hình ROF có đặc điểm thú vị là nó tìm thấy một phiên bản hình ảnh mượt hơn trong khi vẫn giữ lại được các cạnh và cấu trúc của hình ảnh

Kiến thức toán học của mô hình ROF và các kỹ thuật liên quan nằm ngoài nội dung của cuốn sách này. Ở đây chỉ giới thiệu nội dungngắn gọn (đơn giản hóa) trước khi trình bày cách triển khai bộ giải ROF dựa trên thuật toán của Chambolle [5].

Tổng biến thiên cường độ pixel của hình ảnh thang độ xám 1 được định nghĩa là tổng độ lớn của vectơ gradient. Trong miền liên tục được mô tả như sau: (1.9)

Trong miền rời rạc tổng biến thiên cường độ pixel là:

(1.10)

trong đó tổng được lấy trên toàn bộ tọa độ ảnh x = [x, y].

Trong cách trình bày dựa theo Chambolle của ROF, mục tiêu là tìm ra hình ảnh đã được khử nhiễu U để tối thiểu hàm mục tiêu sau:

(1.11)

trong đó khoảng cách || I - U || đo lường sự khác biệt giữa ảnh U và ảnh gốc I. Điều này có nghĩa thực chất là mô hình tìm kiếm các hình ảnh “phẳng” nhưng cho phép “nhảy” tại các cạnh giữa các vùng.

Trong ví dụ này, đã sử dụng hàm roll(), như tên gọi, “rolls” các giá trị của một mảng theo chu kỳ quanh một trục. Điều này rất thuận tiện cho việc tính toán sự khác biệt của điểm lân cận. trong ví dụ này cũng đã sử dụng lệnh linalg.norm() để đo sự khác biệt giữa hai mảng (trong trường hợp này là ma trận hình ảnh U và Uold). Lưu hàm denoise() trong tệp rof.py.