**TRANG 21-26**

Nhiều phần tử có thể được truy cập bằng cách cắt mảng. Dưới đây là một số ví dụ về hình ảnh thang độ xám:

|  |
| --- |
| In[i, :] = In[j, :] # Set the values of row i with values from row j  In[:, i] = 100 # Set all values in column i to 100  In[:100, :50].sum() # The sum of the values in the first 100 rows and 50 columns  In[50:100, 50:100] # Rows 50-100, columns 50-100 (100th not included)  In[i, :].mean() # Average of row i  In[:, -1] # Last column  In[-2, :] # Second to last row |

Lưu ý, ví dụ chỉ có một chỉ số. Nếu chỉ sử dụng một chỉ số, nó được hiểu là chỉ số hàng. Cũng lưu ý các ví dụ cuối cùng. Chỉ số âm sẽ tính những phần tử từ cuối cùng tính lên. Đây là công cụ thường xuyên sử dụng cắt mảng để truy cập các giá trị pixel và đó là một khái niệm quan trọng để hiểu giá trị phần tử trong mảng.

Có nhiều toán tử và cách để sử dụng mảng. Chúng sẽ được giới thiệu trong suốt cuốn sách này. Xem tài liệu trực tuyến hoặc cuốn sách [24] để biết thêm một số giải thích chi tiết.

**1.3.2. Biến đổi cấp độ xám**

Sau khi đọc hình ảnh cho mảng NumPy, có thể thực hiện bất kỳ thao tác toán học nào muốn thực hiện trên chúng. Một ví dụ đơn giản về điều này là biến đổi các mức độ xám của hình ảnh. Lấy bất kỳ hàm f nào ánh xạ khoảng 0...255 (hoặc 0 ... 1 sau khi đã chuẩn hóa hình ảnh) với chính nó (có nghĩa là đầu ra có cùng khoảng với đầu vào). Dưới đây là một số ví dụ.

|  |
| --- |
| from PIL import Image  import numpy as np  # Đọc ảnh và chuyển đổi sang ảnh xám  im = np.array(Image.open('empire.jpg').convert('L'))  # Đảo ngược ảnh (invert image)  im2 = 255 - im  # Kẹp giá trị pixel vào khoảng 100...200  im3 = (100.0 / 255) \* im + 100  # Bình phương giá trị pixel  im4 = 255.0 \* (im / 255.0) \*\* 2 |

Ví dụ đầu tiên đảo ngược các mức độ xám của hình ảnh, ví dụ thứ hai ép cường độ vào khoảng 100...200 và thứ ba áp dụng hàm bậc hai, làm giảm các giá trị của các pixel tối. Hình 1.4 trình bày các hàm và Hình 1.5 là kết quả. Có thể kiểm tra giá trị tối thiểu và tối đa của mỗi hình ảnh bằng cách sử dụng:

|  |
| --- |
| print(int(im.min()), int(im.max())) |

Thực hiện trên các ví dụ hình ảnh bên trên, sẽ nhận được kết quả đầu ra sau:

*255*

253

200

255

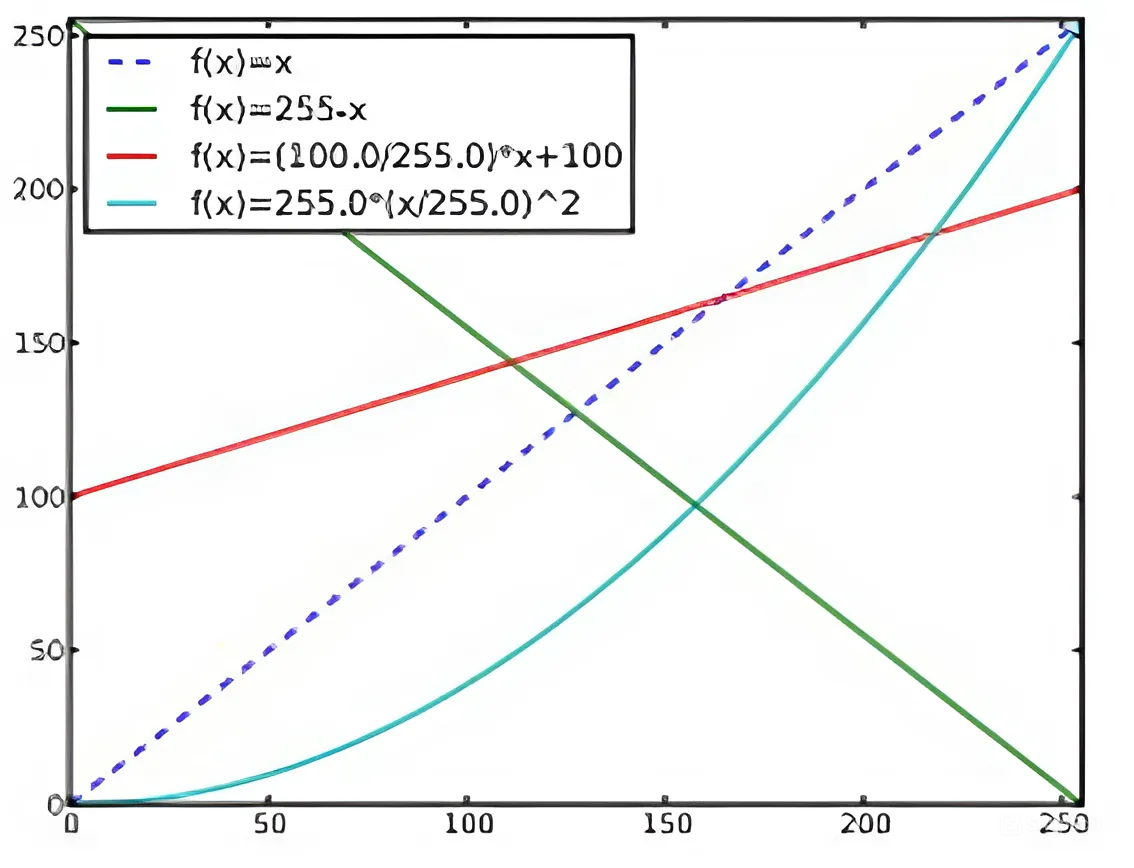
Việc đảo ngược phép biến đổi array() có thể được thực hiện bằng cách sử dụng hàm PIL fromarray() như sau:

|  |
| --- |
| pil\_in = Image.fromarray(im) |

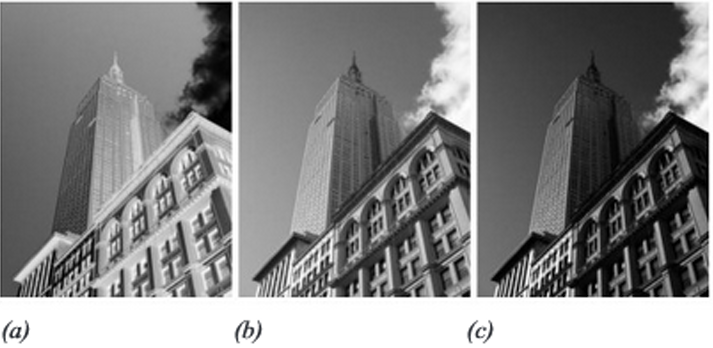
Nếu đã thực hiện một số thao tác để thay đổi loại từ “uint8” sang loại dữ liệu khác, ví dụ như im3 hoặc im4 trong ví dụ trên, cần chuyển đổi lại trước khi tạo hình ảnh PIL.

|  |
| --- |
| pil\_in = Image.fromarray(np.uint8(im)) |

Nếu không hoàn toàn chắc chắn về loại đầu vào, nên làm điều này vì đó là sự lựa chọn an toàn. Lưu ý rằng NumPy sẽ luôn thay đổi loại mảng thành loại “thấp nhất” có thể để mô tả cho dữ liệu. Phép nhân hoặc chia với số có kiểu biến số thực sẽ thay đổi một mảng kiểu số nguyên thành số thực.



*Hình 1.4: Ví dụ biến đổi mức xám. Ba hàm ví dụ cùng với biến đổi cường độ hiển thị dưới dạng đường đứt nét*



*Hình 1.5: Biến đổi mức xám. Áp dụng các hàm trong Hình 1.4*

*(a) Đảo ngược hình ảnh với f (x) = 255 - x,*

*(b) ép hình ảnh với f (x) = (100/255) x +100,*

*(c) biến đổi bậc hai với f(x) = 255(x/255).*

**Thay đổi kích thước hình ảnh**

Mảng NumPy sẽ là công cụ chính để làm việc với hình ảnh và dữ liệu. Không có cách đơn giản để thay đổi kích thước mảng đối với hình ảnh. Có thể sử dụng chuyển đổi đối tượng hình ảnh PIL được trình bày ở phần trước để thực hiện chức năng thay đổi kích thước hình ảnh đơn giản. Thêm phần sau vào imtools.py. Hàm này sẽ có ích cho các phân tích hình ảnh sau này.

|  |
| --- |
| def inresize(im, sz): *"""Resize an image array using PIL."""* pil\_in = Image.fromarray(uint8(im))  return array(pil\_in.resize(sz)) |

**1.3.3. Cân bằng lược đồ ảnh**

Một ví dụ rất dễ hiểu về biến đổi mức xám là cân bằng lược đồ ảnh. Dạng chuyển đổi này làm phẳng lược đồ mức xám của một hình ảnh sao cho tất cả các cường độ cân bằng nhất có thể. Đây thường là cách tốt để cân bằng cường độ hình ảnh trước khi xử lý thêm và cũng là một cách để tăng độ tương phản hình ảnh.

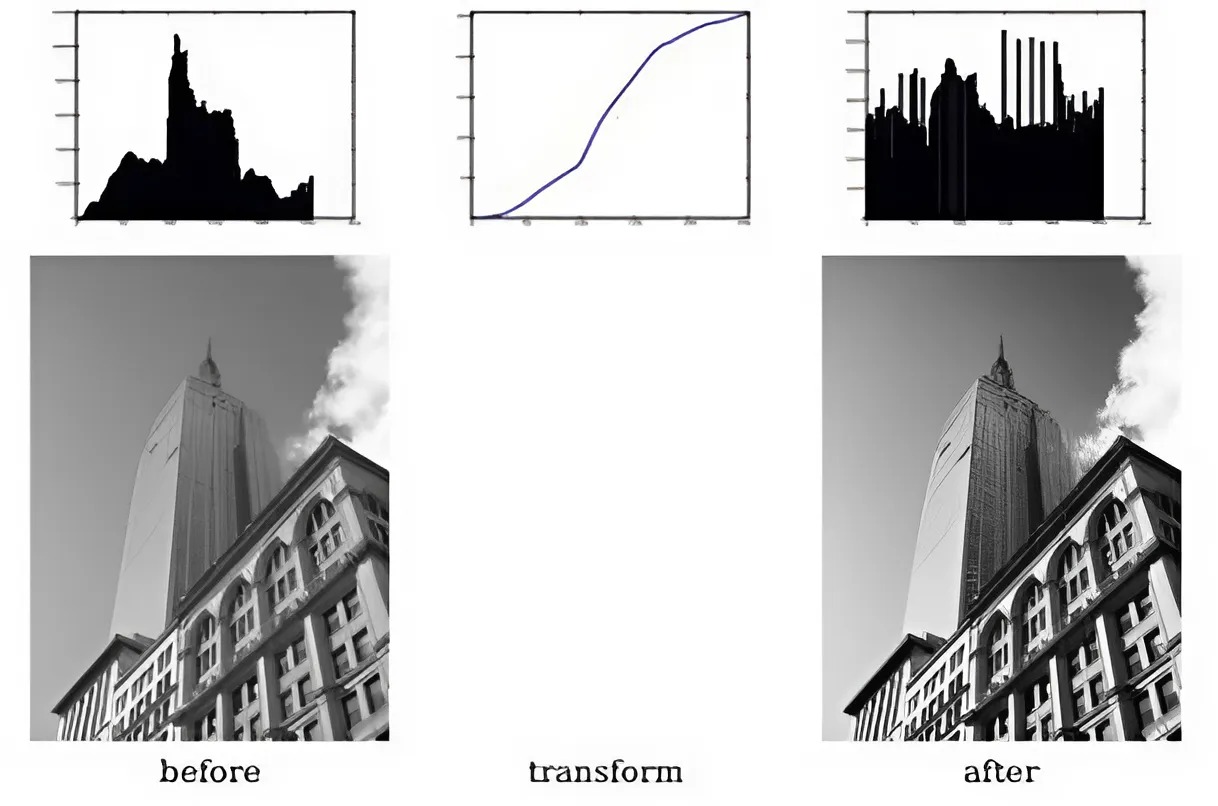
Trong trường hợp này, hàm biến đổi là hàm phân bố tích lũy (cd) của các giá trị pixel trong ảnh (được chuẩn hóa để ánh xạ phạm vi của các giá trị pixel đến phạm vi mong muốn). Sau đây là chương trình mẫu để thực hiện điều đó. Thêm chức năng này vào tập tin imtools.py.

|  |
| --- |
| def histeq(im, nbr\_bins=256):  *"""Histogram equalization of a grayscale image."""* # Lấy histogram của ảnh  inhist, bins = histogram(im.flatten(), nbr\_bins, density=True)   # Tính toán hàm phân phối tích lũy (CDF)  cdf = inhist.cumsum() # Cumulative distribution function  cdf = 255 \* cdf / cdf[-1] # Normalize CDF   # Dùng nội suy tuyến tính của CDF để tìm giá trị pixel mới  im2 = interp(im.flatten(), bins[:-1], cdf)   return im2.reshape(im.shape), cdf |

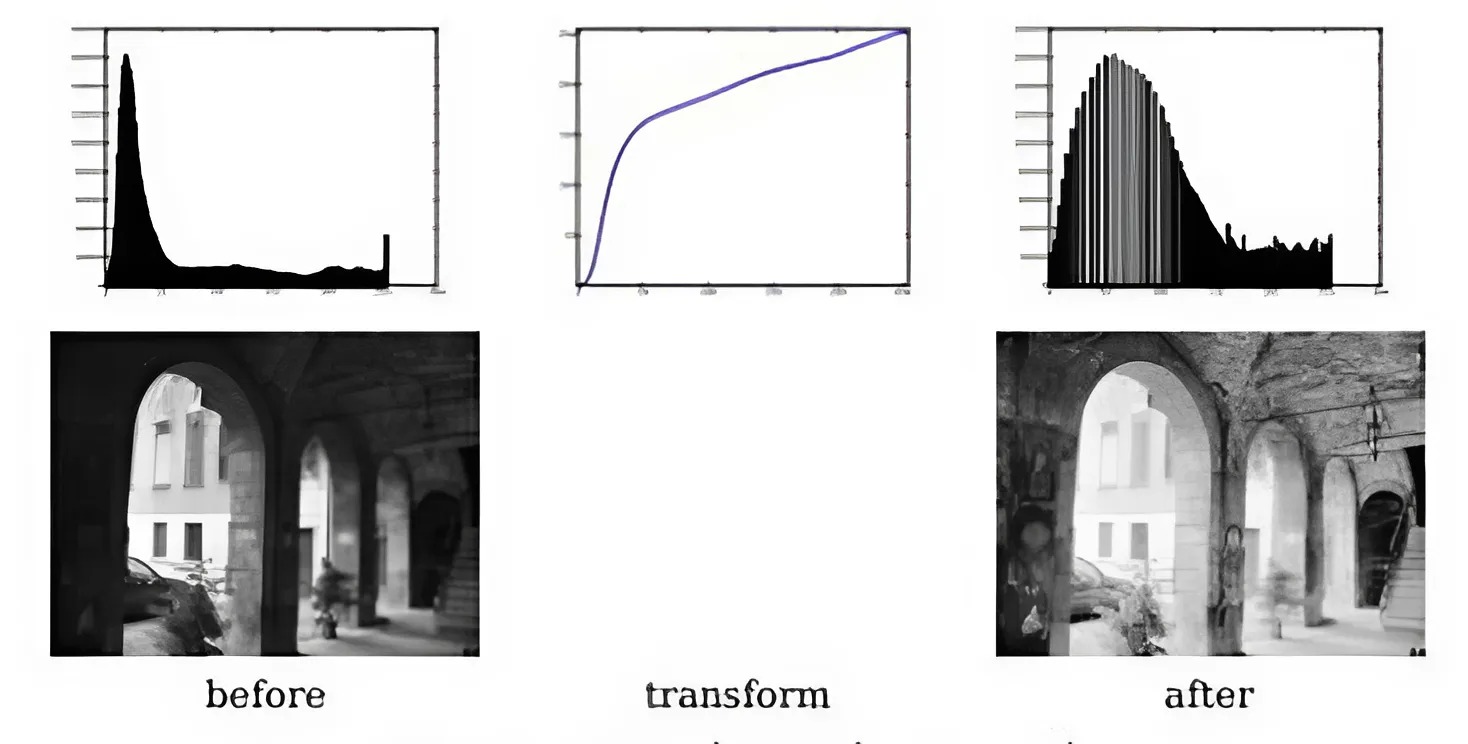
Hàm lấy hình ảnh thang độ xám và số lượng cột (bins) sẽ sử dụng trong lược đồ làm đầu vào và trả về một hình ảnh có cân bằng lược đồ cùng với hàm phân bố tích lũy được sử dụng để thực hiện ánh xạ các giá trị pixel. Lưu ý việc sử dụng phần tử cuối cùng (chỉ số -1) của cdf để chuẩn hóa nó trong khoảng từ 0... I. Thực hiện cân bằng trên một hình ảnh như sau:

|  |
| --- |
| from PIL import Image  import numpy import  im = array(Image.open("AquaTermi\_lowcontrast.jpg").convert('L'))  im2, cdf = intools.histeq(im) |

Hình 1.6 và 1.7 cho thấy các ví dụ về cân bằng lược đồ ảnh. Hàng trên cùng hiển thị lược đồ mức xám trước và sau khi cân bằng cùng với ánh xạ cdf.*Đồ thị ở giữa là hàm biển đổi mức xám*. Như có thể thấy, độ tương phản tăng và chi tiết của các vùng tối hiện rõ ràng.



*Hình 1.6: Vi dụ về cân bằng lược đồ*



*Hình 1.7: Ví dụ về cân bằng lược đồ*

**1.3.4. Phép tính trung bình pixel ảnh**

Phép tính trung bình pixel ảnh là một cách đơn giản để giảm nhiễu hình ảnh và cũng thường được sử dụng cho các hiệu ứng nghệ thuật. Tính toán một hình ảnh trung bình từ một tập các hình ảnh có thể được thực hiện khá đơn giản.

Giả sử tất cả các hình ảnh có cùng kích thước, có thể tính trung bình của tất cả các hình ảnh đó bằng cách tổng hợp chúng và chia cho số lượng hình ảnh. Thêm chức năng sau vào imtools.py.

|  |
| --- |
| def compute\_average(imlist):  *"""Compute the average of a list of images."""* # Mở ảnh đầu tiên và chuyển thành mảng với kiểu dữ liệu float  averagein = array(Image.open(imlist[0]), 'f')   for inname in imlist[1:]:  try:  averagein += array(Image.open(inname))  except:  print(inname + ' ..skipped')   averagein /= len(imlist)   # Trả về ảnh trung bình dưới dạng uint8  return array(averagein, 'uint8') |

Trong chương trình trên những hình ảnh không đọc được sẽ được bỏ qua trong phép tính trung bình. Có một cách khác để tính toán các hình ảnh trung bình bằng cách sử dụng hàm mean(). Điều này đòi hỏi tất cả các hình ảnh được xếp chồng lên nhau thành một mảng (và sẽ sử dụng nhiều bộ nhớ nếu có nhiều hình ảnh), hàm này sẽ được sử dụng trong các phần tiếp theo.

**1.3.5. Phân tích thành phần chính (PCA) của hình ảnh**

Phân tích thành phần chính (PCA) là một kỹ thuật hữu ích để giảm kích thước và tối ưu theo nghĩa là nó thể hiện dữ liệu huấn luyện đa dạng với kích thước càng ít càng tốt. Ngay cả một hình ảnh xám có kích thước 100x100 pixel cũng có 10.000 chiều và có thể được xem như là một điểm trong không gian 10.000 chiều. Một hình ảnh có kích thước megapixel sẽ có hàng triệu chiều. Với kích thước cao như vậy, không có gì ngạc nhiên khi giảm kích thước có ích trong nhiều ứng dụng thị giác máy tính. Ma trận chiếu từ PCA có thể được xem là sự thay đổi hệ tọa độ thành một hệ tọa đ a độ khác trong đó tọa độ sắp xếp theo thứ tự quan trọng giảm dần.

Để áp dụng PCA trên dữ liệu hình ảnh, hình ảnh cần được chuyển đổi thành đại diện vectơ một chiều, ví dụ: sử dụng lệnh flatten() của NumPy.

Các hình ảnh phẳng được thu thập trong một ma trận bằng cách xếp chúng, một hàng cho mỗi hình ảnh. Các hàng sau đó được tính giá trị trung bình trước khi tính toán các hướng chính. Để tìm các thành phần chính, phương pháp SVD (singular value decomposition) thường được sử dụng. Trong một vài trường hợp đặc biệt khi dữ liệu có nhiều chiều (high dimesionality), phương pháp SVD sẽ tính toán rất chậm. Đoạn chương trình mẫu sau đây được viết để thực hiện điều này:

|  |
| --- |
| from PIL import Image import numpy import   def pca(X):  *"""Principal Component Analysis  input: X, matrix with training data stored as flattened arrays in rows  return: projection matrix (with important dimensions first), variance and mean.  """* # Lấy kích thước của X  num\_data, din = X.shape   # Trung tâm dữ liệu  mean\_X = X.mean(axis=0)  X = X - mean\_X # Trung tâm hóa dữ liệu   if din > num\_data:  # PCA - Sử dụng trick compact  M = dot(X, X.T) # Ma trận hiệp phương sai  e, EV = linalg.eigh(M) # Tính trị riêng và vector riêng  tmp = dot(X.T, e, EV).T # Đây là trick compact  V = tmp[::-1] # Lật lại vì các vector riêng cuối cùng là những vector quan trọng  S = sqrt(e)[::-1] # Lấy căn bậc hai của các trị riêng và lật lại  for i in range(V.shape[1]):  V[:, i] /= S  else:  # PCA - Sử dụng SVD  U, S, V = linalg.svd(X)  V = V[:num\_data] # Chỉ giữ lại số lượng vector riêng bằng số mẫu    # Trả về ma trận chiếu, phương sai và giá trị trung bình  return V, S, mean\_X |