ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

BÁO CÁO
DIGITAL IMAGE PROCESSION AND COMPUTER VISION
PROJECT 1
GRAYSCALE AND COLOR IMAGES

GVHD: Ths. Võ Thanh Hùng

—o0o—

SVTH: Nguyễn Tiến Phát - 2011797

TP. HỒ CHÍ MINH, 3/2024

Mục lục

Danh sách hình vẽ

Chương 1

Giới thiệu

1.1 Grayscale

Trong nhiếp ảnh số, hình ảnh được tạo bằng máy tính, và màu sắc học, một hình ảnh xám (grayscale image) là một hình ảnh trong đó giá trị của mỗi điểm ảnh chỉ đại diện cho một lượng ánh sáng duy nhất; nghĩa là nó chỉ mang thông tin về cường độ. Hình ảnh xám, một loại hình ảnh đen trắng hoặc xám, được tạo ra hoàn toàn từ các tông màu xám. Độ tương phản của hình ảnh xám biến đổi từ đen ở cường độ yếu nhất đến trắng ở cường độ mạnh nhất.

Chúng ta cần phải phân biệt giữa hình ảnh xám (grayscale image) với hình ảnh đen trắng (bi-tonal) - hình ảnh chỉ có hai màu: đen hoặc trắng (còn được gọi là bilevel hoặc binary). Hình ảnh xám có nhiều tông xám ở giữa. Hình ảnh xám có thể là một tổ hợp tần số (hoặc bước sóng) cụ thể (hoặc trong thực tế, một dãy tần số hẹp). Một hình ảnh xám theo chuẩn màu sắc (hoặc cụ thể hơn là chuẩn màu sắc đo lường) là một hình ảnh có không gian màu xám được xác định, ánh xạ các giá trị mẫu số lưu trữ vào kênh không màu của một không gian màu chuẩn, và được dựa trên các tính chất được đo lường của thị giác con người.

Hình ảnh xám thường được sử dụng để hiển thị thông tin về cường độ ánh sáng, chẳng hạn như hình ảnh chụp X-quang, hình ảnh y học và hình ảnh khoa học. Nó cũng là một phần quan trọng của các thuật toán xử lý hình ảnh, bao gồm việc làm mở, phát hiện biên và phân đoạn.

1.2 Color image

Hình ảnh màu là một loại hình ảnh chứa thông tin về màu sắc tại từng điểm ảnh. Thay vì chỉ có một giá trị cường độ như hình ảnh xám, hình ảnh màu bao gồm các kênh màu khác nhau. Các kênh này thường là đỏ (R), xanh lá cây (G) và xanh dương (B).

Khi kết hợp các kênh màu này, chúng ta có thể tạo ra hình ảnh màu đầy đủ. Cách chúng ta kết hợp các kênh màu này sẽ ảnh hưởng đến màu sắc cuối cùng của hình ảnh. Ví dụ:

- Nếu chúng ta kết hợp R = 255, G = 0, và B = 0, chúng ta sẽ có màu đỏ đậm.
- Nếu chúng ta kết hợp R = 0, G = 255, và B = 0, chúng ta sẽ có màu xanh lá cây đậm.
- Nếu chúng ta kết hợp R = 0, G = 0, và B = 255, chúng ta sẽ có màu xanh dương đậm.

Để tạo ra hình ảnh màu từ hình ảnh xám, chúng ta cần phải hiểu cách ánh xạ giữa các kênh màu và cường độ ánh sáng. Quá trình này thường được thực hiện thông qua không gian màu như RGB (Red-Green-Blue) hoặc HSV (Hue-Saturation-Value).

Chương 2

Hiện thực

2.1 Cơ sở lý thuyết

2.1.1 Grayscale

Một chiến lược phổ biến là sử dụng các nguyên tắc quang học hoặc màu sắc học để tính toán các giá trị màu xám (trong không gian màu xám mục tiêu) sao cho có cùng độ sáng với hình ảnh màu gốc. Ngoài cùng độ sáng, phương pháp này cũng đảm bảo rằng cả hai hình ảnh sẽ có cùng độ sáng tuyệt đối khi hiển thị. Độ sáng được định nghĩa bằng một mô hình tiêu chuẩn về thị giác con người, vì vậy bảo toàn độ sáng trong hình ảnh xám cũng bảo toàn độ đo cường độ ánh sáng nhận thức khác. Để chuyển đổi một màu từ một không gian màu dựa trên một mô hình màu RGB (phi tuyến) thông thường sang một biểu diễn màu xám của độ sáng của nó, chức năng giải nén gamma phải được loại bỏ trước khi qua phép mở rộng gamma (tuyến tính hóa) để chuyển đổi hình ảnh sang một không gian màu RGB tuyến tính, để có thể áp dụng tổng trọng số thích hợp cho các thành phần màu tuyến tính($R_{\text{linear}}, G_{\text{linear}}, B_{\text{linear}}$) để tính độ sáng tuyến tính Ylinear, sau đó có thể được nén gamma trở lại nếu kết quả màu xám cũng được mã hóa và lưu trữ trong một không gian màu phi tuyến thông thường. Đối với không gian màu sRGB phổ biến, phép mở rộng gamma được định nghĩa như sau:

$$\mathbf{C}_{\text{linear}} = \begin{cases} \frac{C_{\text{srgb}}}{12.92}, & \text{if } C_{\text{srgb}} \leq 0.04045 \\ \left(\frac{C_{\text{srgb}}+0.055}{1.055}\right)^{2.4}, & \text{otherwise} \end{cases}$$

Trong đó, Csrgb đại diện cho bất kỳ ba màu sắc sRGB nén gamma ($R_{\text{srgb}}, G_{\text{srgb}}, B_{\text{srgb}}$, mỗi màu trong phạm vi [0,1]) và C_{linear} là giá trị cường độ tuyến tính tương ứng ($R_{\text{linear}}, G_{\text{linear}}, B_{\text{linear}}$, cũng trong phạm vi [0,1]). Sau đó, độ sáng tuyến tính được tính bằng tổng trọng số của ba giá trị cường độ tuyến tính. Không gian màu sRGB được định nghĩa dựa trên độ sáng tuyến tính CIE 1931 Y_{linear}, được cho bởi:

$\mathbf{Y}_{\text{linear}} = \mathbf{0.2126R}_{\text{linear}} + \mathbf{0.7152G}_{\text{linear}} + \mathbf{0.0722B}_{\text{linear}}$

Để hiểu rõ hơn, ba hệ số cụ thể này đại diện cho sự nhận thức về cường độ (độ sáng) của con người trung bình đối với ánh sáng của ba màu chính cộng gộp chính xác REC. 709 (màu sắc) được sử dụng trong định nghĩa của sRGB. Có thể hiểu rằng, lý do có công thức là do thị giác của con người nhạy cảm nhất với màu xanh lá, vì vậy nó có giá trị hệ số lớn nhất (0.7152), và ít nhạy cảm nhất với màu xanh da trời, vì vậy nó có giá trị hệ số nhỏ nhất (0.0722). Để mã hóa cường độ màu xám trong RGB tuyến tính, mỗi thành phần màu có thể được đặt bằng độ sáng tuyến tính được tính toán (thay bằng các giá trị để có được màu xám tuyến tính này), sau đó thường cần phải được nén gamma để quay trở lại một biểu diễn phi tuyến thông thường. Đối với sRGB, mỗi màu chính của nó sau đó được đặt bằng Ysrgb nén gamma được cho bởi nghịch đảo của phép mở rộng gamma ở trên như:

$$\mathbf{Y}_{\text{srgb}} = \begin{cases} 12.92Y_{\text{linear}}, & \text{if } Y_{\text{linear}} \leq 0.0031308 \\ 1.055Y_{\text{linear}}^{1/2.4} - 0.055, & \text{otherwise} \end{cases}$$

Bởi vì ba thành phần sRGB sau đó bằng nhau, cho thấy rằng nó thực sự là một hình ảnh xám (không màu), nên chỉ cần lưu trữ các giá trị này một lần, và chúng tôi gọi đây là hình ảnh màu xám kết quả. Đây là cách nó sẽ được lưu trữ thông thường trong các định dạng hình ảnh tương thích với sRGB hỗ trợ biểu diễn một kênh màu xám, chẳng hạn như JPEG hoặc PNG. Trình duyệt web và phần mềm khác nhận ra hình ảnh sRGB nên tạo ra cùng một hiển thị cho hình ảnh màu xám như vậy như nó sẽ cho một hình ảnh "màu" sRGB có cùng giá trị trong ba kênh màu.

2.1.2 Color image

Khi đã có ảnh xám từ phần trước, bây giờ chúng ta sẽ xem cách kết hợp các channel màu để tạo lại ảnh màu ban đầu. Mục tiêu là khôi phục lại màu sắc và chi tiết của hình ảnh màu gốc hoặc đơn giản là làm cho một bức ảnh xám trở thành ảnh màu. Có nhiều cách để thực hiện điều này, ví dụ:

- *Hệ màu LAB*:Nếu đã chuyển đổi ảnh xám sang không gian màu LAB, chúng ta có thể sử dụng các kênh màu a* và b* để tái tạo màu sắc.
- *Hệ màu RGB*: Nếu sử dụng các thông tin màu từ ảnh gốc, chúng ta có thể áp dụng các trọng số tương ứng cho các channel màu trong không gian màu RGB.
- *Hệ màu CMYK*: thường được sử dụng trong in ấn, CMYK là viết tắt của cơ chế hệ màu trừ, thiên về màu tối, bao gồm có: C = Cyan (xanh), M = Magenta (đỏ cánh sen hay hồng sẫm), Y = Yellow (vàng chanh), K = Black (đen) (Black là màu được bổ sung vào hệ nhằm tăng cường độ tương phản).

Ở bài tập lớn lần này sử dụng hệ màu RGB do sự phổ biến rộng rãi của nó. Ngoài ra, chúng ta còn có thể áp dụng các thuộc tính khác khi chỉnh màu cho một bức ảnh như Độ Bão Hòa (Saturation), Nhiệt Độ Màu (Temperature), Màu Sắc (Hue):

- *Độ bão hòa*: Nhân giá trị màu của mỗi kênh với saturation_factor giúp tăng cường sức mạnh của màu sắc, làm cho ảnh trở nên sống động hơn.
- *Nhiệt độ*: Nhân và chia giá trị của red và blue cho temperature_factor có thể làm thay đổi màu sắc tổng thể của ảnh, làm ấm hoặc làm lạnh hình ảnh
- *Màu sắc*: Sử dụng ma trận xoay giúp duy trì tỷ lệ màu sắc và chuyển đổi chúng theo giá trị hue_shift. Điều này cho phép điều chỉnh màu sắc mà không làm thay đổi độ chói lọt của hình ảnh.

2.2 Hiện thực với python

Source code có thể được tìm thấy tại: google colab

2.2.1 Grayscale

Sử dụng thư viện Pillow kết hợp với NumPy để chuyển từng pixel sang con số mong muốn theo như công thức đã nêu phía trên.

```
from PIL import Image
from PIL import ImageShow
import matplotlib.pyplot as plt
import numpy as np

def to_grayscale_built(image):
  gray_image = []
  # Access pixels using ‘getdata()‘
  pixels = image.getdata()
  # Process each pixel and build the grayscale image
  for pixel in pixels:
    gray_value = pixel[0] * 0.2126 + pixel[1] * 0.7152 + pixel[2] * 0.0722
    # Append to a new list (not gray_values directly)
    gray_image.append(gray_value)
    # Create a new grayscale image based on dimensions
  height = image.height
  width = image.width
  # Reshape the 1D list into a 2D array using NumPy
  gray_image_2d = np.array(gray_image).reshape(height, width)
  # Create a new grayscale image
  gray_image_2d_img = Image.fromarray(gray_image_2d.astype('uint8'))
  return gray_image_2d_img
```

Sử dụng thư viện OpenCV với hàm chuyển ảnh sang xám có sẵn.

```
import cv2
from google.colab.patches import cv2_imshow

def to_grayscale(image):
  """Convert to grayscale image"""
  return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

2.2.2 Color image

Sau khi đã giới thiệu một số chiến lược lý thuyết trong phần trước, bây giờ chúng ta sẽ tiếp tục với phần thực hành, triển khai các chiến lược này bằng mã nguồn Python sử dụng thư viện NumPy và Pillow. Đoạn mã dưới đây mô tả cách áp dụng các phương pháp chúng ta đã thảo luận vào một ảnh màu.

```
import numpy as np
from PIL import Image
import random

# Adjust saturation
def saturation_increase(red, green, blue):
    saturation_factor = 1.2
    red = red * saturation_factor
    green = green * saturation_factor
    blue = blue * saturation_factor
    return red, green, blue

# Adjust temperature
def temperature_adjustment(red, green, blue):
    temperature_factor = 1.0
    red = red * temperature_factor
    blue = blue / temperature_factor
    return red, green, blue

# Adjust hue
def hue_adjustment(red, green, blue):
    hue_factor = 1.0
    hue_shift = np.pi * hue_factor
    rot_matrix = np.array([[0.9692, 0.2578, 0.0722],
                           [0.0031, 0.9995, -0.0029],
                           [0.0079, -0.0015, 1.0000]])

    rgb_linear = np.stack([red, green, blue], axis=-1)
    rgb_linear = np.dot(rgb_linear, rot_matrix.T)
    red, green, blue = np.split(rgb_linear, 3, axis=-1)

    return red.squeeze(), green.squeeze(), blue.squeeze()

# Applay the change into image
def apply_colorization(img_array, operations):
    red_channel = np.zeros_like(img_array)
    green_channel = np.zeros_like(img_array)
    blue_channel = np.zeros_like(img_array)

    red_channel[:, :] = img_array
    green_channel[:, :] = img_array
    blue_channel[:, :] = img_array

    for operation in operations:
        red_channel, green_channel, blue_channel = operation(red_channel, green_channel,
            blue_channel)

    color_img_array = np.stack([red_channel, green_channel, blue_channel], axis=-1)
    colored_img = Image.fromarray(np.uint8(color_img_array))
    return colored_img
```

Chương 3

Kết quả đạt được và tổng kết

3.1 Kết quả

3.1.1 Grayscale

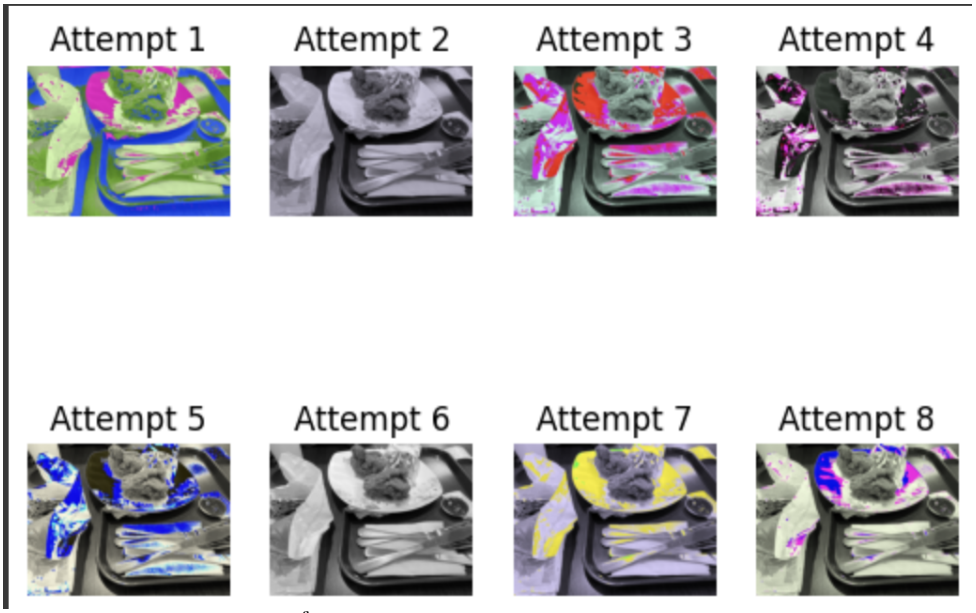
Hình 3.1: *Ảnh gốc*

Hình 3.2: *Ảnh xám xử lý pixel bằng Pillow*

Hình 3.3: *Ảnh xám từ OpenCV*

Giữa hai ảnh không có sự khác biệt đáng kể, không thể phân biệt bằng mắt thường. Cả hai đều đã được chuyển sang ảnh xám một cách hoàn hảo và chính xác.

3.1.2 Color image



Hình 3.4: *Ảnh được tạo thành từ các channel màu khác nhau*

Sự thay đổi của các thông số màu sắc tạo ra những bức ảnh với màu sắc khác nhau. Tuy nhiên, chúng không đạt được chất lượng như ảnh gốc. Nguyên nhân chủ yếu là do chúng ta khó thể nào xác định được chính xác những hệ số liên quan để có thể kết hợp chúng tạo ra những màu sắc mong muốn.

3.1.3 Tổng kết

Dự án đã đạt được mục tiêu của mình, tức là tách ảnh màu thành ảnh xám và tái tạo lại ảnh màu từ ảnh xám. Quá trình này không chỉ giúp giảm kích thước dữ liệu mà còn tạo ra những biểu diễn ảnh màu động và phong phú.

Các chiến lược lý thuyết và mã nguồn Python triển khai đã được tích hợp một cách hiệu quả, và việc thực hiện các thay đổi màu sắc đã mang lại kết quả ảnh màu sáng tạo và đa dạng. Tùy thuộc vào nhu cầu sử dụng cụ thể, có thể điều chỉnh các tham số hoặc thêm các chiến lược khác để đạt được hiệu quả màu sắc mong muốn.

Dự án này mang lại sự linh hoạt trong việc thao tác màu sắc của ảnh và có thể được mở rộng để tích hợp thêm các tính năng và chiến lược khác nhằm cải thiện chất lượng và đa dạng của quá trình xử lý màu sắc.

Thông qua dự án, chúng ta đã học được rất nhiều kiến thức từ việc hiểu cấu tạo của ảnh, cách biến đổi cho để ra output mong muốn đến thao tác phối màu cho ảnh. Đây sẽ là nền tảng vững chắc để ta có thể đi đến những kiến thức chuyên sâu hơn trong việc học xử lý ảnh số, cụ thể chúng ta có thể áp dụng deep learning để tô màu cho ảnh thay vì cài đặt cứng các thông số như trong dự án lần này.

3.2 Tài liệu tham khảo

1. Johnson, Stephen (2006). Stephen Johnson on Digital Photography. O'Reilly. ISBN 0-596-52370-X.
2. Poynton, Charles (2012). Digital Video and HD: Algorithms and Interfaces (2nd ed.). Morgan Kaufmann. pp. 31–35, 65–68, 333, 337. ISBN 978-0-12-391926-7. Retrieved 2022-03-31
3. , Charles A. (2022-03-14). Written at San Jose, Calif.. Rogowitz, B. E.; Pappas, T. N. (eds.). Rehabilitation of Gamma (PDF). SPIE/IST Conference 3299: Human Vision and Electronic Imaging III; January 26–30, 1998. Bellingham, Wash.: SPIE. doi:10.1117/12.320126. Archived (PDF) from the original on 2023-04-23.
4. Stokes, Michael; Anderson, Matthew; Chandrasekar, Srinivasan; Motta, Ricardo (1996-11-05). "A Standard Default Color Space for the Internet – sRGB". World Wide Web Consortium – Graphics on the Web. Part 2, matrix in equation 1.8. Archived from the original on 2023-05-24.
5. Burger, Wilhelm; Burge, Mark J. (2010). Principles of Digital Image Processing Core Algorithms. Springer Science Business Media. pp. 110–111. ISBN 978-1-84800-195-4.