**BÀI TẬP GIỮA KỲ**

**XỬ LÝ ẢNH (25/03/2024)**

**Nhóm 2**

|  |  |  |
| --- | --- | --- |
| **Tên** | **STT** | **Tỉ lệ (%)** |
| Nguyễn Ngọc Trung | 69 | 35 |
| Hoàng Bảo Long | 15 | 34 |
| Lê Thị Ánh Trinh | 67 | 31 |

**Bài 1**.

**SOURCE PYTHON CODE:**

import cv2

import numpy as np

import time

from matplotlib import pyplot as plt

I = cv2.imread("plumage.png", 0)

h = np.array([[1, 2, 4, 2, 1]]) / 10

h\_phay = np.transpose(h)

H = np.array([[1, 2, 4, 1, 2],

[2, 4, 8, 4, 2],

[4, 8, 16, 8, 4],

[2, 4, 8, 4, 2],

[1, 2, 4, 2, 1]]) / 100

**# câu a**

I1 = cv2.filter2D(I, -1, h)

I2 = cv2.filter2D(I1, -1, h\_phay)

**# câu b**

I3 = cv2.filter2D(I, -1, H)

**# câu c**

start\_time\_a = time.time()

I1 = cv2.filter2D(I, -1, h)

I2 = cv2.filter2D(I1, -1, h\_phay)

end\_time\_a = time.time()

time\_taken\_a = end\_time\_a - start\_time\_a

start\_time\_b = time.time()

I3 = cv2.filter2D(I, -1, H)

end\_time\_b = time.time()

time\_taken\_b = end\_time\_b - start\_time\_b

print("Thời gian thực hiện cho câu a:", time\_taken\_a)

print("Thời gian thực hiện cho câu b:", time\_taken\_b)

plt.subplot(1, 2, 1), plt.imshow(I2, cmap='gray')

plt.title('I2'), plt.xticks([]), plt.yticks([])

plt.subplot(1, 2, 2), plt.imshow(I3, cmap='gray')

plt.title('I3'), plt.xticks([]), plt.yticks([])

plt.show()

**# câu d**

f\_img = np.fft.fft2(I)

f\_img\_shifted = np.fft.fftshift(f\_img)

magnitude\_spectrum = 20 \* np.log(np.abs(f\_img\_shifted))

rows, cols = I.shape

u = np.fft.fftfreq(rows)

v = np.fft.fftfreq(cols)

plt.figure()

plt.imshow(magnitude\_spectrum, cmap='gray')

plt.title('Magnitude Spectrum')

plt.xlabel('u')

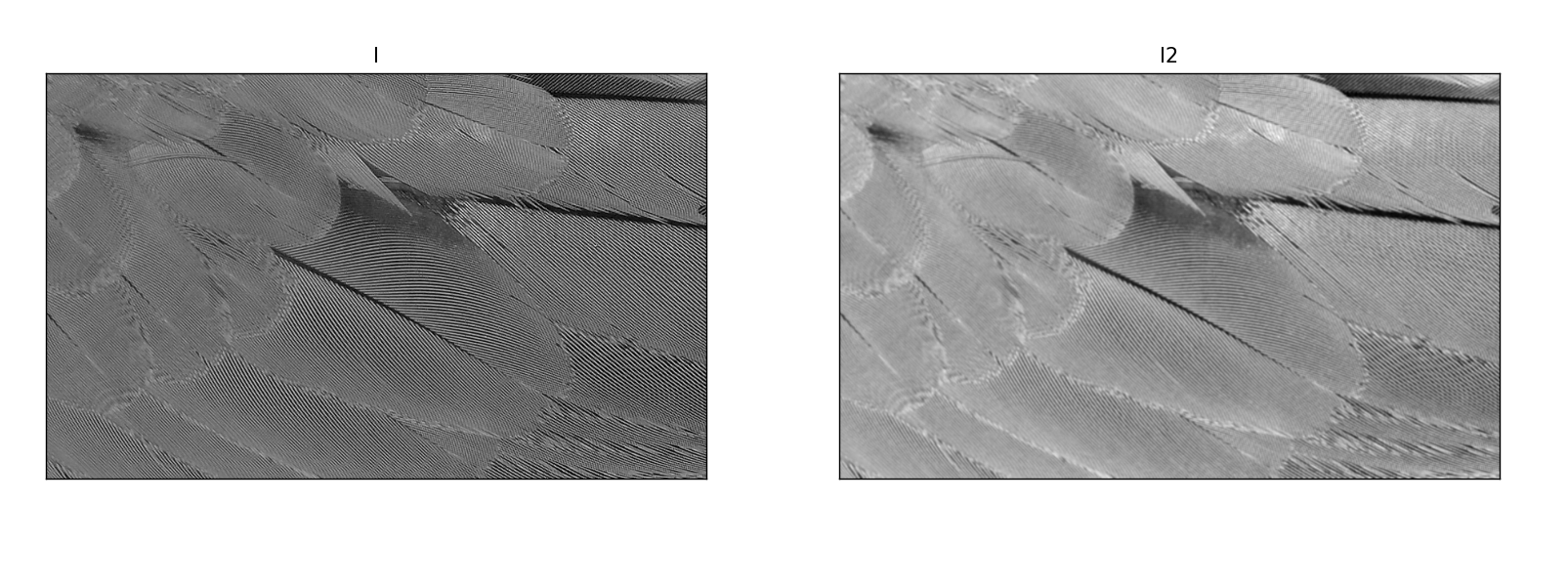
plt.ylabel('v')

plt.xticks([0, cols], [-0.5, 0.5])

plt.yticks([0, rows], [-0.5, 0.5])

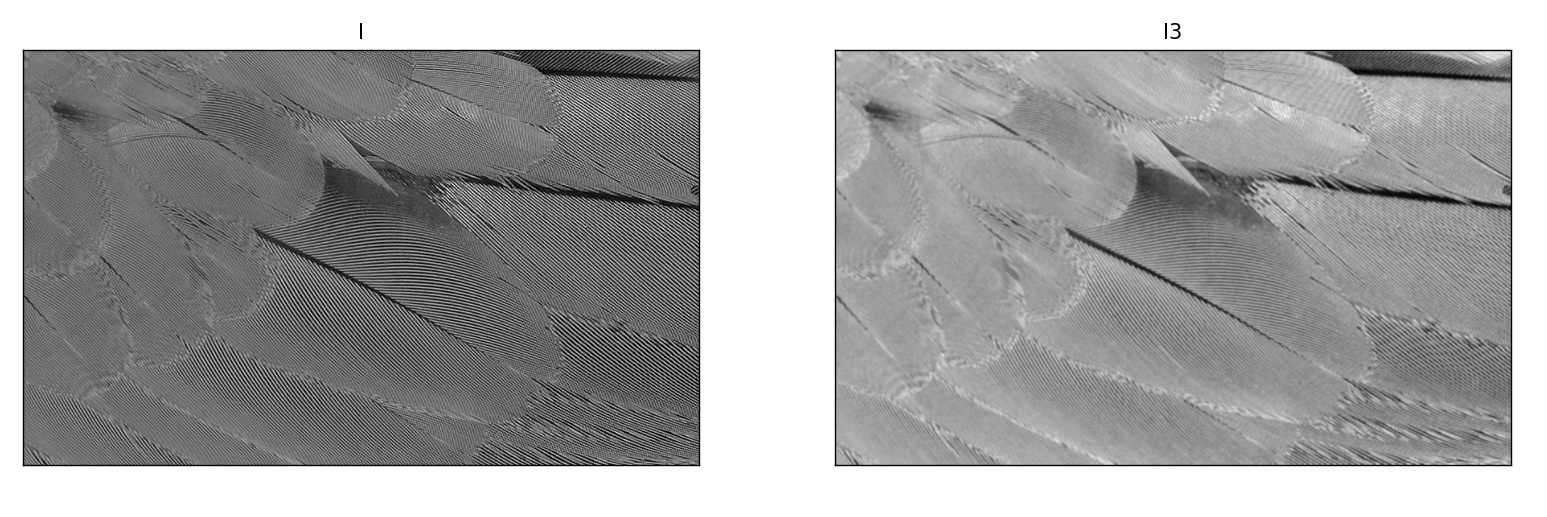
plt.colorbar()

plt.show()

1. **Hiển thị ảnh I2**

*Hình 1.1 Hiển thị ảnh I2*

1. **Hiển thị ảnh I3**



*Hình 1.2 Hiển thị ảnh I3*

* **So sánh I2 và I3:**

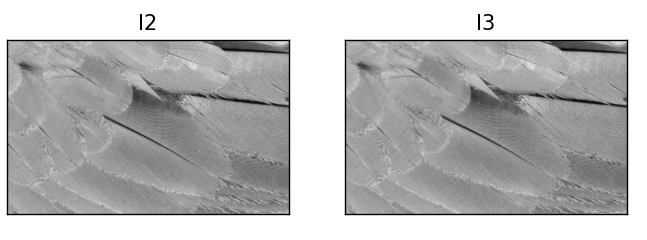
**+** ẢnhI2 và I3 là 2 kết quả khác nhau sau khi áp dụng bộ lọc. I2 được tạo ra bằng

cách áp dụng bộ lọc giữa h và h’ lần lượt trên ảnh gốc, trong khi I3 được tạo ra bằng cách áp dụng bộ lọc H.

+ Ảnh I2 là kết quả của việc lọc thông tin bên trong ảnh để giảm nhiễu và làm mờ

cạnh, trong khi ảnh I3 là kết quả của việc áp dụng bộ lọc trung bình trên toàn bộ

ảnh để làm mịn ảnh.

****

*Hình 1.3 So sánh I2 và I3*

* **Nhận xét:**

**+** I2 được lọc qua bộ lọc chuyển vị h từ ma trận ngang thành dọc nên ảnh đầu ra sẽ có xu hướng nhìn rõ các đường ngang của ảnh hơn.

**+** Nhìn chung, I2 được áp dụng để giảm nhiễu và I3 để làm mịn ảnh. Tuy

nhiên, I2 có xu hướng làm mất một số chi tiết nhỏ và cạnh, trong khi I3 làm

mất nhiều chi tiết hơn.

*Bảng 1****.****1**thời gian thực hiện giữa hai cách làm ở câu a và b*

|  |  |  |
| --- | --- | --- |
| **Thời gian (s) Câu** | **a** | **b** |
| Lần 1 | 0.004448413848876953 | 0.0010039806365966797 |
| Lần 2 | 0.008519649505615234 | 0.0 |
| Lần 3 | 0.0030069351196289062 | 0.0005178451538085938 |
| Lần 4 | 0.0 | 0.0 |
| Lần 5 | 0.0010051727294921875 | 0.001984119415283203 |
| Lần 6 | 0.0 | 0.0 |
| Lần 7 | 0.0 | 0.004530429840087891 |
| Lần 8 | 0.0 | 0.004386425018310547 |
| Lần 9 | 0.0 | 0.004903078079223633 |
| Lần 10 | 0.0007328987121582031 | 0.0007977485656738281 |
| Lần 11 | 0.0004162788391113281 | 0.001558065414428711 |
| Lần 12 | 0.0 | 0.0 |
| Lần 13 | 0.0003895759582519531 | 0.0 |
| Lần 14 | 0.0004181861877441406 | 0.0010066032409667969 |
| Lần 15 | 0.000514984130859375 | 0.0 |
| **Trung bình** | **0.001296806335** | **0.001379219691** |

* **Nhận xét và giải thích:**

*Bảng 1.2 Đánh giá tiêu chí câu a và câu b*

|  |  |  |
| --- | --- | --- |
| Đánh giá Câu | a | b |
| Thời gian (s) | Nhanh hơn | Chậm hơn |
| Tính giản đơn mã | Đơn giản hơn | Phức tạp hơn |
| Độ hiệu quả | Ít hiệu quả | Hiệu quả hơn |

* **Thời gian thực hiện:**

**+ Câu a:** Thời gian thực hiện cho phép tích chập theo câu a là thời gian tích chập hai lần với các bộ lọc h và h’nhưng do h và h’ có kích thước khá nhỏ nên do đó thời gian thực hiện có nhanh hơn đôi chút so với thời gian thực hiện câu b

**+ Câu b:** Thời gian thực hiện cho phép tích chập theo câu b là thời gian tích chập một lần với kernel H có kích thước lớn hơn nên thời gian thực hiện lớn hơn câu a.

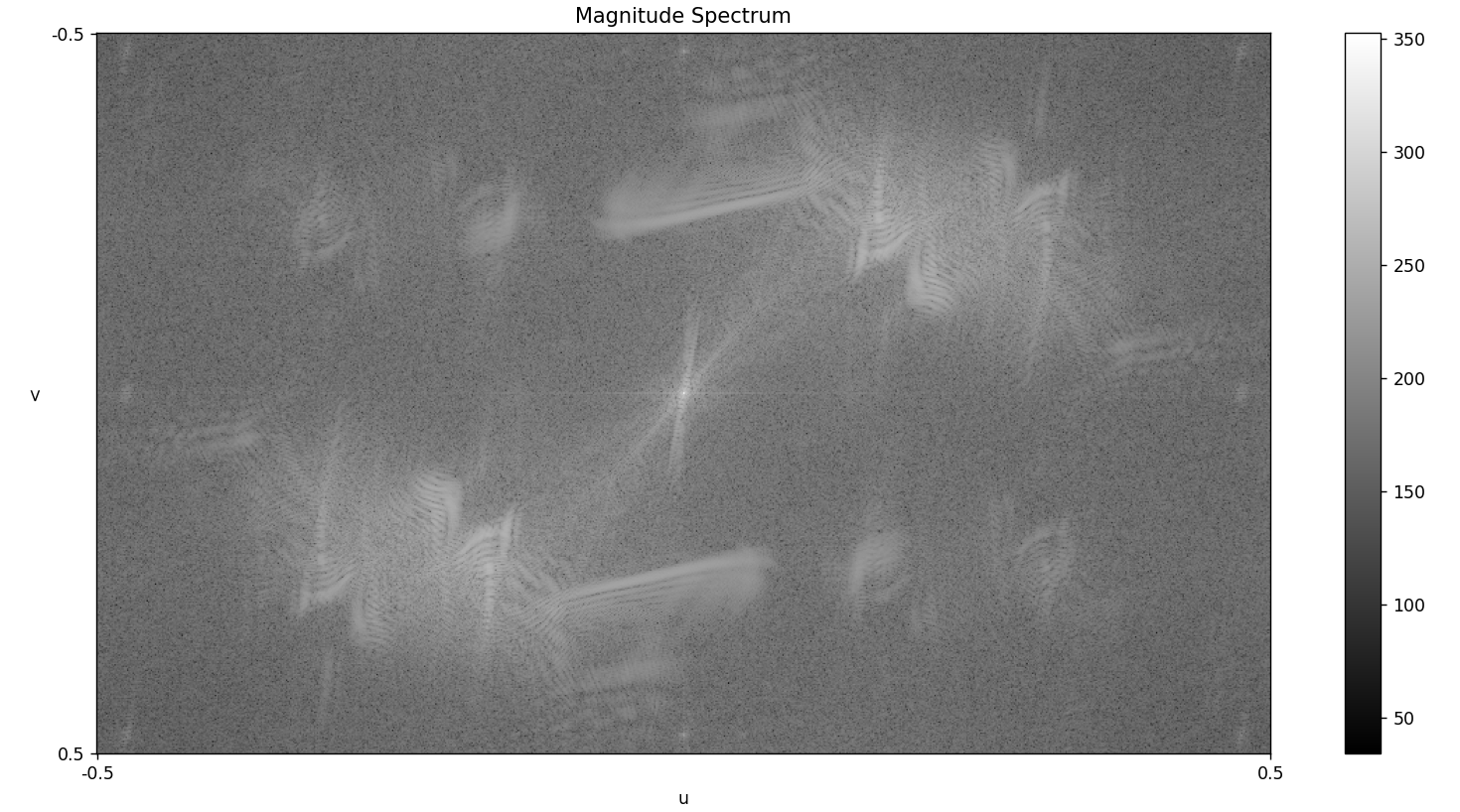
* **Đơn giản của mã:**

**+ Câu a:** Tích chập 2 lần với các bộ lọc h và h’nhưng do h và h’ khá đơn giản nên khi tính tích chập 2 lần các phép toán trở nên đơn giản hơn

**+ Câu b:** Tích chập 1 lần với bộ lọc H có kích thước lớn hơn nên khi tính tích chập sẽ gây ra độ phức tạp về mặt tính toán hơn so với câu a

* **Kết quả hiển thị :**

**+** Kết quả hiển thị cả 2 câu khá giống nhau vì cả 2 đều là kết quả của phép tích chập trên cùng một ảnh đầu vào. Tuy nhiên, kết quả vẫn có sự khác biệt nho nhỏ mà khó thể thấy bằng mắt thường do sự khác biệt của đặc điểm bộ lọc.

1. ****

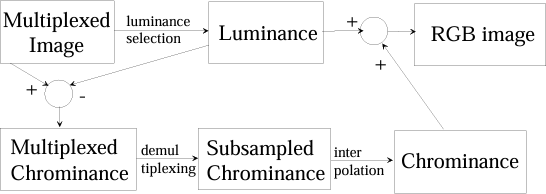
*Hình 1.4 Phổ biên độ của ảnh I theo tần số (u,v) với*

* **Nhận xét kết quả câu a, b:**

**+** Quan sát phổ biên độ trong khoản tần số thấp , chúng ta có thể đánh giá được ảnh hưởng của các phép toán tích chập lên các thành phần tần số của ảnh và từ đó hiểu rõ hơn về các biến đổi trong miền tần số của ảnh.

**+ Câu a:** Phép tích chập được thực hiện theo bộ lọc h và h' ta thực hiện một loại làm mờ trung bình trên ảnh gốc. Kết quả là các thành phần tần số thấp trong phổ biên độ sẽ được tăng cường rõ hơn, trong khi các thành phần tần số cao sẽ bị giảm đáng kể. Điều này có thể dẫn đến hiệu ứng làm mờ trongảnh kết quả**.  
+ Câu b :** Phép tích chập được thực hiện theo bộ lọc H ta thực hiện một loại làm mờ trung bình trên ảnh gốc. Kết quả là các thành phần tần số thấp trong phổ biên độ sẽ được tăng cường rõ hơn, ảnh sẽ mịn hơn và bị mất một số chi tiết trong ảnh.

**Bài 2**:

* **Tóm tắt phương pháp demosacing của Alleyson:**

*Hình 2.1 Sơ đồ giải thuật Alleysson*

**Bước 1: Tách Luminance (tính toán Limunance)**

**+** Đọc ảnh màu theo mô hình Bayer CFA *Hình 2.1* và tách nó thành các channel màu tương ứng. Trong mô hình Bayer CFA, mỗi pixel chỉ có thông tin về một trong ba màu đỏ, xanh hoặc lục.

+ Áp dụng bộ lọc trung bình trên ảnh màu để tính toán độ sáng trung bình của từng pixel.

# FL = /64 *(2.1)*

**Bước 2: Tách Chrominance**

**+** Thành phần Chrominance được tính bằng cách lấy ảnh màu theo mô hình Bayer CFA trừ đi thành phần Luminance (L) đã tính được ở **Bước 1**:

Chrominance = {R-L;G-L;B-L} *(2.2)*

**Bước 3: Nội suy tái tạo lớp Chrominance**

­+ Cần khôi phục lại các giá trị Chrominance còn thiếu trong ảnh theo mô hình Bayer CFA. Quá trình này được thực hiện bằng cách sử dụng một thuật toán tối ưu hóa để dự đoán các giá trị màu bị thiếu dựa trên các giá trị màu có sẵn của các pixel lân cận.

+ Sử dụng bộ lọc:

FR,B = /4; FG = /4; *(2.3)*

+ Sử dụng tích chập bộ lọc *(2.3)* khôi phục các giá trị Chrominance.

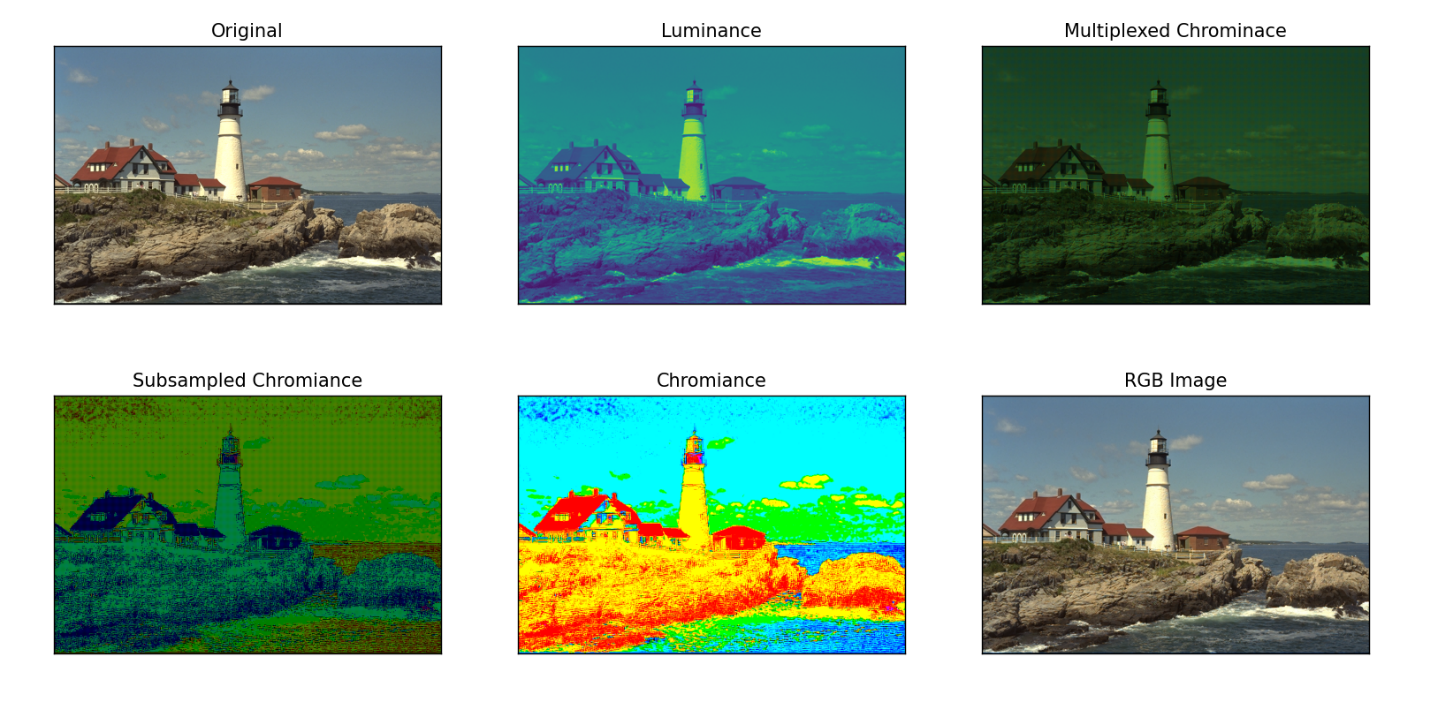
Trong đó:  
FR,B: Bộ lọc nội suy tuyến tính cho các lưới màu đỏ và xanh dương

FG: Bộ lọc nội suy tuyến tính cho các lưới màu xanh lục

**Bước 4: Kết hợp Luminance và chrominance sau khi tái tạo để tạo ảnh màu hoàn chỉnh**

**+** Các giá trị Chrominance sau khi khôi phục sẽ bố trí theo mô hình Bayer CFA, để tái tạo lại ảnh màu hoàn hình, cần kết hợp thành phần luminance và thành phần chrominance sau khi khôi phục.

**Bước 5: Kết quả**



*Hình 2.2 Kết quả qua các bước sử dụng phương pháp demosacing của Alleyson*

* **Nhấn mạnh điểm chính của phương pháp này:**

**+** Điểm chính của phương pháp này là sự kết hợp giữa các kỹ thuật xử lý hình ảnh và mô hình toán học để tái tạo ảnh màu có chất lượng cao từ dữ liệu ảnh đơn sắc.

+ Luminance và Chrominance là hai thành phần của màu sắc được sử dụng để mô tả màu sắc trong hệ thống màu sắc. Luminance là thành phần độ sáng của màu sắc, trong khi Chrominance là thành phần độ bão hòa màu sắc. Luminance và Chrominance được sử dụng để xác định màu sắc của một điểm ảnh trong hệ thống màu sắc, như RGB (Red, Green, Blue) hoặc YUV (Lu- minance, Chrominance Blue, Chrominance Red). Việc tách Luminance và Chrominance rất quan trọng trong xử lý ảnh và video để thực hiện các bước tiền xử lý như đồng bộ hóa màu sắc, cân bằng màu sắc, nâng cao chất lượng ảnh và video.

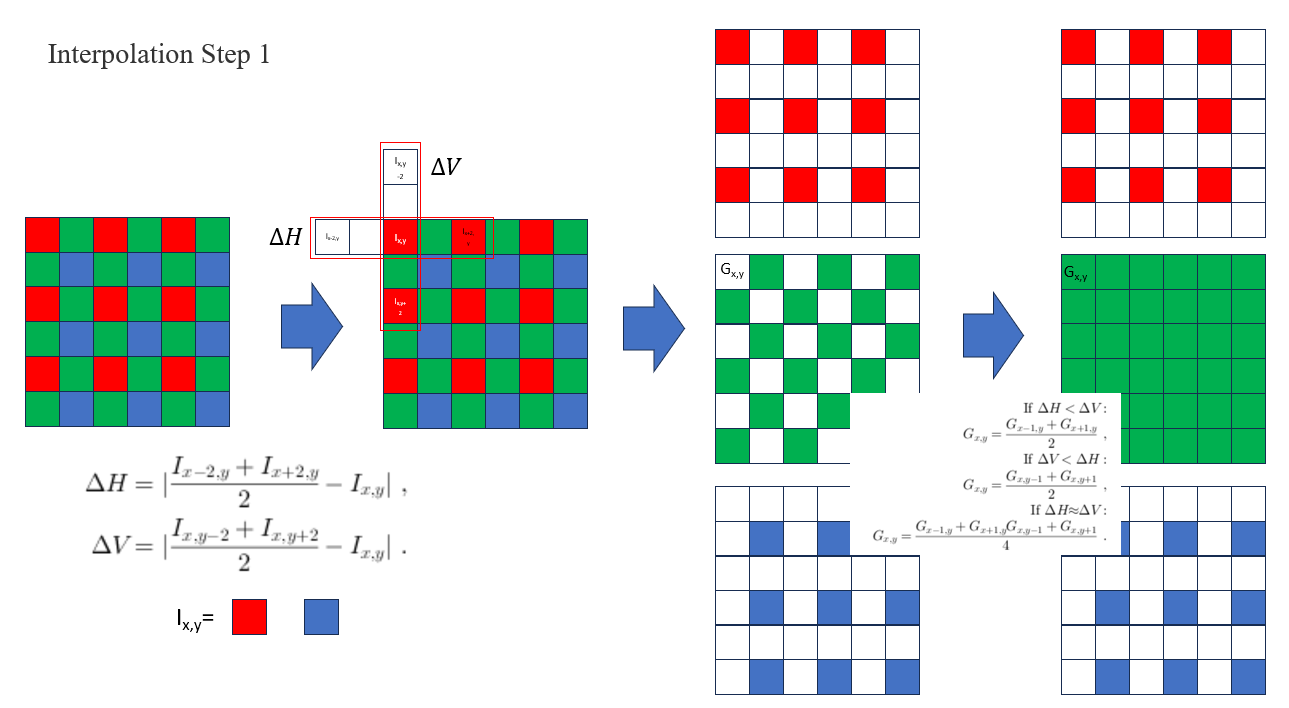
+ Phương pháp này giải quyết vấn đề này bằng cách sử dụng hai bộ lọc khác nhau để tách riêng luminance và chrominance từ ảnh gốc.

**Bài 3**:

1. **Mô tả phương pháp Laroche**

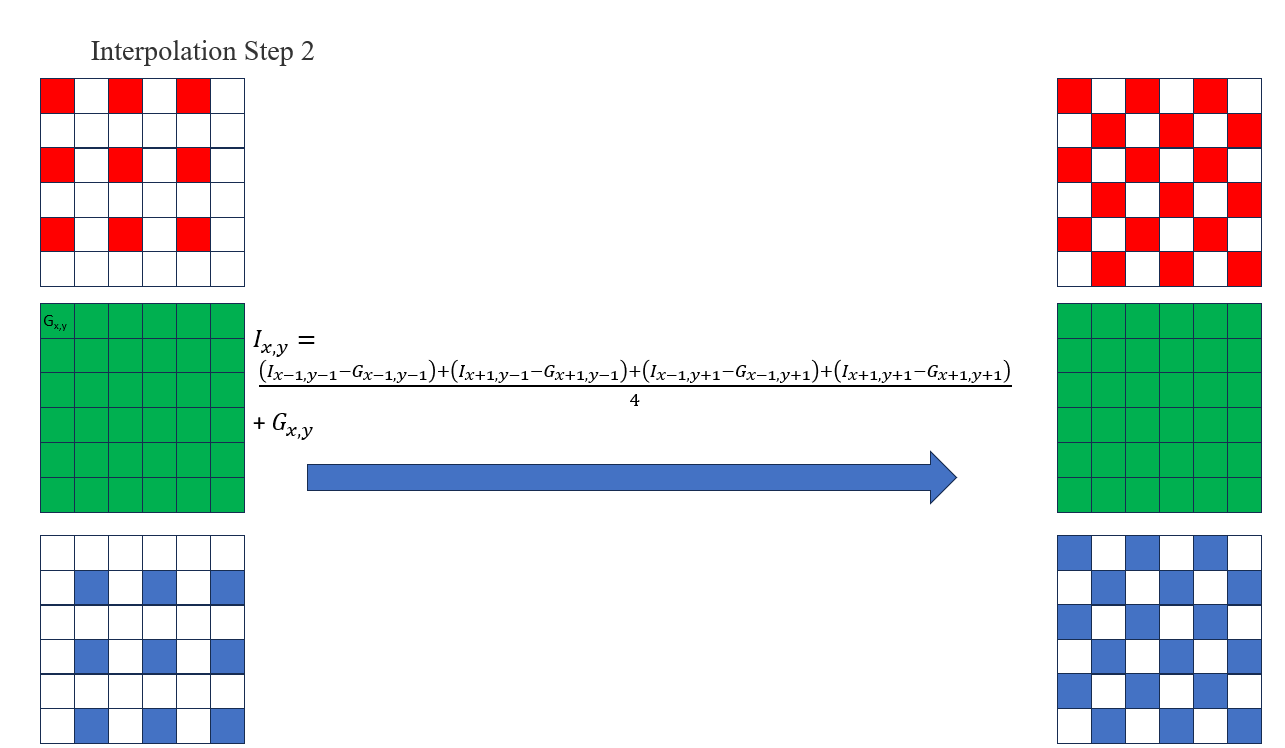
* Phương pháp của Laroche phụ thuộc vào loại hướng cạnh trong việc phục hồi màu từ dữ liệu Bayer. Thực tế, nó xem xét hướng của cạnh để giảm thiểu sai số nội suy, tức là nội suy được thực hiện dọc theo cạnh. Sự nội suy như vậy được áp dụng trước tiên vào kênh màu Xanh lá. Khi kênh màu Xanh lá được hoàn thành, các kênh màu Đỏ/ Xanh lá cây được tái tạo dựa trên các pixel về sự khác biệt màu sắc không đổi.
* Giải thích:

+ Bước 1:

* Tìm tất cả các các giá trị pixel màu xanh lá cây được ngoại suy với các gradient sắc độ ngang và dọc được xác định bằng kernel 5xs5 như *Hình 3.1*
* Tính toán các giá trị Gx,y dựa theo công thức trong *Hình 3.1* nhằm đảm bảo độ chính xác cao tại các giá trị pixel tại thời điểm ta tính.
* Nếu Tức giá trị pixel tại điểm ta tính có giá trị lấy trung bình cộng 2 pixel lân cận theo trục x
* Nếu Tức giá trị pixel tại điểm ta tính có giá trị lấy trung bình cộng 2 pixel lân cận theo trục y
* Nếu Tức giá trị pixel tại điểm ta tính có giá trị lấy trung bình cộng 4 pixel lân cận

*Hình 3.1 Mô tả bước 1*

+ Bước 2:

*Hình 3.2 Mô tả quá trình thực hiện bước 2*

* Từ các giá trị đã tìm được ở Bước 1 ta tính các giá trị pixel màu đỏ và xanh lam bị thiếu có thể được tính toán theo công thức (3.1) tức cộng trung bình giá trị của 4 pixel lân cận trừ đi giá trị pixel đã tính ở Bước 1 như được mô tả ở *Hình 3.2*:

*(3.1)*

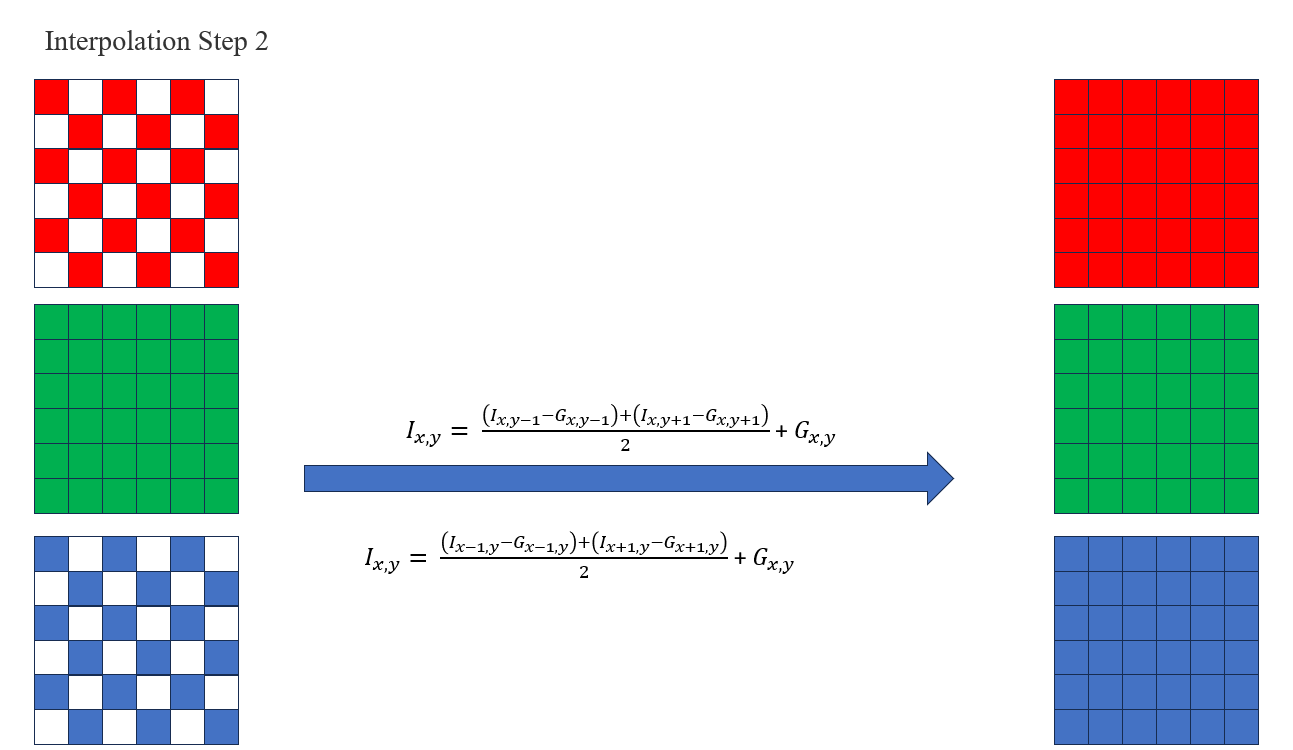
* Tiếp theo ta tính giá trị pixel màu xanh lam trên pixel màu xanh lục trong đường màu đỏ hoặc màu đỏ trên pixel màu xanh lục trong đường màu xanh lam theo công thức *(3.2)*

+ (3.2)

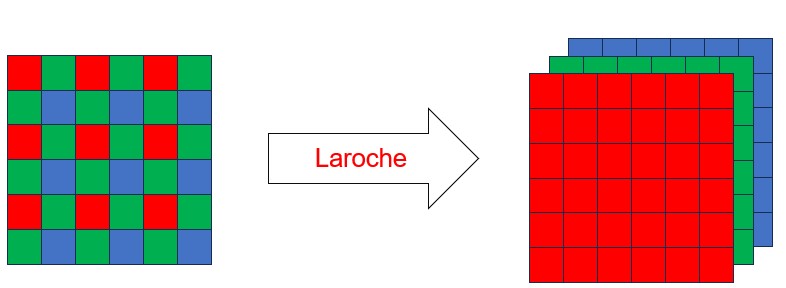
* Tiếp theo ta tính giá trị pixel màu xanh lam trên pixel màu xanh lục trong đường màu xanh lam và màu đỏ trên pixel màu xanh lục trong đường màu đỏ theo công thức *(3.3)*

+ *(3.3)*

* Mô phỏng rõ ràng hình ảnh pixel được tính và nội suy giá trị pixel thông qua *Hình 3.3*



*Hình 3.3 Mô tả quá trình bước 2 nội suy ra pixel xanh dương – đỏ*

* Tổng kết: kết quả thu được trong *Hình 3.4*

*Hình 3.4 Kết quả thu được từ thuật toán Laroche*

1. **SOURCE CODE**

import numpy as np

import cv2

import matplotlib.pyplot as plt

from skimage.metrics import structural\_similarity as ssim

def createBayerImage(image:np.array):

b,g,r = cv2.split(image)

h, w = image.shape[:2]

bayer\_image = np.zeros((h,w), dtype=np.uint8)

red\_id = np.array([[1,0],[0,0]])

green\_id = np.array([[0,1],[1,0]])

blue\_id = np.array([[0,0],[0,1]])

red\_mask = np.tile(red\_id, (h//2, w//2))

green\_mask = np.tile(green\_id, (h//2, w//2))

blue\_mask = np.tile(blue\_id, (h//2, w//2))

bayer\_image = r\*red\_mask + g\*green\_mask + b\*blue\_mask

return bayer\_image

def demoaicingLorache(bayer\_image:np.array):

h, w = bayer\_image.shape[:2]

demoaicing\_image = np.zeros((h,w,3), dtype=np.uint8)

red\_mask = np.tile(np.array([[1, 0], [0, 0]]), (h//2, w//2))

green\_mask = np.tile(np.array([[0, 1], [1, 0]]), (h//2, w//2))

blue\_mask = np.tile(np.array([[0, 0], [0, 1]]), (h//2, w//2))

demoaicing\_image[:,:,2] = bayer\_image \* red\_mask

demoaicing\_image[:,:,1] = bayer\_image \* green\_mask

demoaicing\_image[:,:,0] = bayer\_image \* blue\_mask

border = 2

image\_bordered = cv2.copyMakeBorder(src=bayer\_image, top=border, bottom=border, left=border, right=border, borderType=cv2.BORDER\_CONSTANT)

**# Interpolation Step 1**

for i in range(0, w):

for j in range(0, h):

if (i%2 == 0 and j % 2 == 0):

deltaH = abs((image\_bordered[j + border, i + border - 2] + image\_bordered[j + border, i + border + 2]) / 2 - image\_bordered[j+ border, i + border])

deltaV = abs((image\_bordered[j + border - 2, i + border] + image\_bordered[j + border + 2, i + border]) / 2 - image\_bordered[j+ border, i + border])

if (deltaH < deltaV):

demoaicing\_image[j, i, 1] = (image\_bordered[j+border, i+border-1] + image\_bordered[j+border, i+border+1])//2

elif (deltaV < deltaH):

demoaicing\_image[j, i, 1] = (image\_bordered[j+border-1, i+border] + image\_bordered[j+border+1, i+border])//2

else:

demoaicing\_image[j, i, 1] = (image\_bordered[j+border, i+border-1] + image\_bordered[j+border, i+border+1] + image\_bordered[j+border-1, i+border] + image\_bordered[j+border+1, i+border])//4

elif ((i+j)%2 == 1):

pass

else:

deltaH = abs((image\_bordered[j + border, i+border - 2] + image\_bordered[j + border, i+border + 2]) /2 - image\_bordered[j+ border, i + border])

deltaV = abs((image\_bordered[j + border - 2, i+border] + image\_bordered[j + border + 2, i+border]) /2 - image\_bordered[j+ border, i + border])

if (deltaH < deltaV):

demoaicing\_image[j, i, 1] = (image\_bordered[j+border, i+border-1] + image\_bordered[j+border, i+border+1])//2

elif (deltaV < deltaH):

demoaicing\_image[j, i, 1] = (image\_bordered[j+border-1, i+border] + image\_bordered[j+border+1, i+border])//2

else:

demoaicing\_image[j, i, 1] = (image\_bordered[j+border, i+border-1] + image\_bordered[j+border, i+border+1] + image\_bordered[j+border-1, i+border] + image\_bordered[j+border+1, i+border])//4

border = 1

image\_bordered = cv2.copyMakeBorder(src=bayer\_image, top=border, bottom=border, left=border, right=border, borderType=cv2.BORDER\_CONSTANT)

green\_bordered = cv2.copyMakeBorder(src=demoaicing\_image[:,:,1], top=border, bottom=border, left=border, right=border, borderType=cv2.BORDER\_CONSTANT)

**# Interpolation Step 2**

for i in range(0, w):

for j in range(0, h):

if (i%2 == 0 and j % 2 == 0):

demoaicing\_image[j, i, 0] = (image\_bordered[j+border-1, i+border-1] - green\_bordered[j+border-1, i+border-1] + image\_bordered[j+border-1, i+border+1] - green\_bordered[j+border-1, i+border+1] + image\_bordered[j+border+1, i+border-1] - green\_bordered[j+border+1, i+border-1] + image\_bordered[j+border+1, i+border+1] - green\_bordered[j+border+1, i+border+1])//4 + green\_bordered[j+border, i+border]

pass

elif ((i+j)%2 == 1):

if i%2 == 0:

c1 = 2

c2 = 0

else:

c1 = 0

c2 = 2

demoaicing\_image[j, i, c1] += (image\_bordered[j+border-1, i+border] - green\_bordered[j+border-1, i+border] +\

image\_bordered[j+border+1, i+border] - green\_bordered[j+border+1, i+border] )//2 + green\_bordered[j+border, i+border]

demoaicing\_image[j, i, c2] += (image\_bordered[j+border, i+border-1] - green\_bordered[j+border, i+border-1] +\ image\_bordered[j+border, i+border+1] - green\_bordered[j+border, i+border+1] )//2 + green\_bordered[j+border, i+border]

pass

else:

demoaicing\_image[j, i, 2] = (image\_bordered[j+border-1, i+border-1] - green\_bordered[j+border-1, i+border-1] +image\_bordered[j+border-1, i+border+1] - green\_bordered[j+border-1, i+border+1] + image\_bordered[j+border+1, i+border-1] - green\_bordered[j+border+1, i+border-1] +image\_bordered[j+border+1, i+border+1] - green\_bordered[j+border+1, i+border+1])//4 + green\_bordered[j+border, i+border]

return demoaicing\_image

bgr\_img = cv2.imread('kodim24.png')

bayer\_image = createBayerImage(bgr\_img)

rgb\_image = demoaicingLorache(bayer\_image)

**# Tính PSNR**

psnr = cv2.PSNR(bgr\_img, rgb\_image)

print("PSNR:", psnr)

**# Tính SSIM**

gray\_img1 = cv2.cvtColor(bgr\_img, cv2.COLOR\_BGR2GRAY)

gray\_img2 = cv2.cvtColor(rgb\_image, cv2.COLOR\_BGR2GRAY)

ssim\_value = ssim(gray\_img1, gray\_img2)

print("SSIM:", ssim\_value)

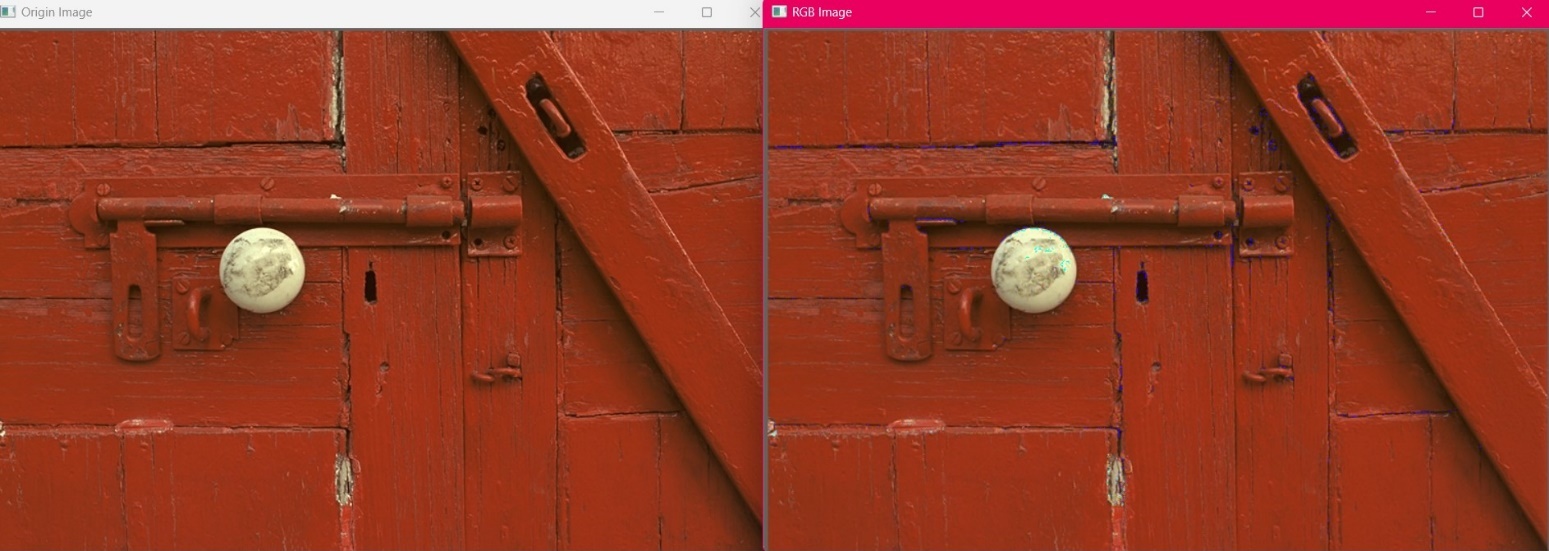
cv2.imshow("Origin Image", bgr\_img)

cv2.imshow("RGB Image", rgb\_image.astype('uint8'))

cv2.waitKey(0)

cv2.destroyAllWindows()

**KẾT QUẢ**



*Bảng 3.1 Thống kê PSNR của 3 phương pháp với bộ ảnh Kodak*

|  |  |  |  |
| --- | --- | --- | --- |
| Ảnh | PSNR  Bilinear Interpolation | PSNR  Alleysson’s Method | PSNR  Laroche’s Method |
| Kodim01 | 25.95771169717905 | 32.29956651069124 |  |
| Kodim02 | 31.88477211787727 | 35.96722021659926 |  |
| Kodim03 | 32.786363362422726 | 37.47356833970346 |  |
| Kodim04 | 32.62267354981076 | 37.65572749198228 |  |
| Kodim05 | 26.348149736679602 | 34.0855390517087 |  |
| Kodim06 | 27.157361526883253 | 32.957838549653026 |  |
| Kodim07 | 32.37406878444729 | 38.14344495347352 |  |
| Kodim08 | 23.462685090471002 | 29.1573000598674 |  |
| Kodim09 | 31.682097144821327 | 37.35754713937641 |  |
| Kodim10 | 31.576392118726947 | 37.99287247813162 |  |
| Kodim11 | 32.62267354981076 | 34.96292795519799 |  |
| Kodim12 | 32.13965122850916 | 37.69149243064172 |  |
| Kodim13 | 23.75850425925706 | 31.638885683542505 |  |
| Kodim14 | 28.52616254548884 | 33.29284173224768 |  |
| Kodim15 | 30.635234345794977 | 34.90618203690113 |  |
| Kodim16 | 30.31149012880686 | 35.01634099788792 |  |
| Kodim17 | 32.000352817633015 | 39.22928741428746 |  |
| Kodim18 | 27.871346986802422 | 35.17876778306218 |  |
| Kodim19 | 28.099410116619865 | 34.117308477516424 |  |
| Kodim20 | 29.959825017317158 | 35.68881124104105 |  |
| Kodim21 | 28.160495302904213 | 34.63515170717589 |  |
| Kodim22 | 30.208934975277216 | 35.40691483987397 |  |
| Kodim23 | 34.07035495436864 | 38.36371803712281 |  |
| Kodim24 | 26.575872496079477 | 33.37039920115577 |  |
| Trung bình cộng | 29.61 | 35.27 |  |

*Bảng 3.2 Thống kê SSIM của 3 phương pháp*

|  |  |  |  |
| --- | --- | --- | --- |
| Ảnh | SSIM  bilinear interpolation | SSIM  Alleysson’s method | SSIM  Laroche’s method |
| Kodim01 | 0.9081589064856154 | 0.9823257065294101 |  |
| Kodim02 | 0.9415881094573291 | 0.9824409502933816 |  |
| Kodim03 | 0.9635015887005847 | 0.9891617609522873 |  |
| Kodim04 | 0.9548502214331582 | 0.9872200568566207 |  |
| Kodim05 | 0.9406875625943272 | 0.9902875944987762 |  |
| Kodim06 | 0.9548502214331582 | 0.9822532725129333 |  |
| Kodim07 | 0.9758011217387694 | 0.9917319779321366 |  |
| Kodim08 | 0.9160256724254063 | 0.9806992308527196 |  |
| Kodim09 | 0.9592381523420109 | 0.9876452355839497 |  |
| Kodim10 | 0.960039085088055 | 0.989008558499257 |  |
| Kodim11 | 0.933284035981776 | 0.9849745345411225 |  |
| Kodim12 | 0.9531971090034893 | 0.986169868001557 |  |
| Kodim13 | 0.8853821425769401 | 0.9845161110224079 |  |
| Kodim14 | 0.9372857086708567 | 0.9867210232835518 |  |
| Kodim15 | 0.9524108554131608 | 0.9857269885226382 |  |
| Kodim16 | 0.9377466422528964 | 0.9843947850965225 |  |
| Kodim17 | 0.9636907600986326 | 0.992278866826779 |  |
| Kodim18 | 0.934580337382259 | 0.9875642288795624 |  |
| Kodim19 | 0.9355240124645947 | 0.9853685395803923 |  |
| Kodim20 | 0.9558333731950279 | 0.9840875091784969 |  |
| Kodim21 | 0.9443803988708319 | 0.9862840571350534 |  |
| Kodim22 | 0.9390201338717924 | 0.981864173430908 |  |
| Kodim23 | 0.9763570741653506 | 0.9885313613158733 |  |
| Kodim24 | 0.9384182319867689 | 0.9880829840090269 |  |
| Trung bình cộng | 0.944 | 0.986 |  |

* **Nhận xét:**