ĐẠI HỌC ĐÀ NẴNG

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN TỬ VIỄN THÔNG

BÁO CÁO CUỐI KÌ

XỬ LÝ ẢNH

GVHD : TS. Trần Thị Minh Hạnh

SVTH : Lý Huỳnh Hữu Trí 19DTCLC2

Đà Nẵng, tháng 12 năm 2022

Câu 1 : Cải thiện chất lượng hình ảnh

Cách 1:

(1) Phương pháp và lý do chọn phương pháp là :

- **Phương pháp :** Sử dụng bộ lọc trung vị (Median Filter) kết hợp tích chập với ma trận Kernel để tăng độ sắc nét của bức ảnh
- **Lý do chọn phương pháp:** Chúng ta sử dụng bộ lọc trung vị (Median Filter) là vì như ta thấy ở ảnh gốc nhiễu chủ yếu của bức ảnh này là nhiễu muối-tiêu (Salt – and – pepper noise) xuất hiện ở các pixel ngẫu nhiên trong cả một bức ảnh nên khi sử lọc trung vị thì sẽ loại bỏ được các giá trị cực trị khỏi ảnh , giữ lại các giá trị thật của ảnh và cải thiện được ảnh
- Ta sử dụng hàm cv2.filter2D(image , depth , kernel): Hàm này có chức năng kết hợp ma trận 2d với hình ảnh ở mức pixel và tạo ra hình ảnh đầu ra cụ thể được thực hiện bằng cách nhân các giá trị của mỗi giá trị pixel với giá trị hạt nhân ở các vị trí tương ứng và thêm tất cả các giá trị vừa được Kernel tính trọng số bằng cách nhân và tạo thành một điểm ảnh (pixel) lặp lại cho đến hết các điểm ảnh (pixel) trong bức ảnh
- 3 tham số trong hàm là :

    + image : ảnh truyền vào

    + depth : mặc định là -1 (nghĩa là độ sâu của ảnh đầu ra giống với độ sâu của ảnh đầu vào)

    + Kernel : ma trận dùng để tích chập với ảnh cần xử lý (Kernel có nhiều loại khác nhau . Vidu: Kernel để làm sắc nét , Kernel để tìm cạnh , Kernel làm mờ,..v,..v)

(2) Kết quả:

Ảnh Gốc **Ảnh sau khi xử lý**

- **Nhận xét :** Khi sử ta sử dụng phương pháp lọc trung vị để tiến hành loại bỏ khỏi nhiễu muối-tiêu (Salt – and – pepper noise) kết hợp với tích chập với ma trận Kernel tăng độ sắt nét thì hai phương pháp này thể hiện một cách khá tốt . nhiễu muối-tiêu đã được loại bỏ gần như hoàn toàn . bức ảnh đã cải thiện một cách rõ rệt

(3) Code Python:

```
# ------------------------------------#
# Author: Huu Tri                     #
# Update: 10/12/2022                  #
# Median Filter And Convolution Kernel #
# ------------------------------------#

import cv2
import numpy as np

img = cv2.imread('../XLA-cuoiky-2022/final1.bmp', 0)
new_img = cv2.imread('../XLA-cuoiky-2022/final1.bmp', 0)
prop = new_img.shape

for i in range(1, prop[0] - 1):
    for j in range(1, prop[1] - 1):

        win = []
        for x in range(i - 1, i + 2):
            for y in range(j - 1, j + 2):
                win.append(img[x][y])
        win.sort()

        new_img[i][j] = win[4]

sharpen_kernel = np.array([[0, -1, 0],
                           [-1, 5, -1],
                           [0, -1, 0]],
                          dtype=np.float32)

sharpen_image = cv2.filter2D(src=new_img,
                             ddepth=-1,
                             kernel=sharpen_kernel)

cv2.imshow('image_root', img)
cv2.imshow('image_remove_salt_pepper', new_img)
cv2.imshow('image_improve', sharpen_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Cách 2:

(1) Phương pháp và lý do chọn phương pháp :

- **Phương pháp:** Cân bằng Histogram cho bức ảnh rồi sau đó dùng lọc trung vị (Median Filter) để lọc bỏ nhiễu muối và tiêu (Salt – and – pepper noise)

- **Lý do chọn phương pháp:** Do bức ảnh gốc ban đầu điểm ảnh (pixel) bị mờ nên ta sử dụng phương pháp cân bằng Histogram để thực hiện cân bằng cho các điểm ảnh , khi ảnh đã được cân bằng lên thì các điểm ảnh sẽ được cải thiện hơn cường độ phân bố điểm ảnh sẽ không bị rời rạc ảnh sẽ sắc nét hơn , và để loại bỏ được muối và tiêu thì cách tốt nhất đó là dùng lọc trung vị (Median Filter) trong miền không gian để loại bỏ

(2) Kết quả , so sánh nhận xét:

Ảnh gốc **Ảnh sau khi xử lý**

- **Nhận xét**: Khi ta sử dụng cân bằng Histogram kết hợp với lọc trung vị (Median Filter) thì ảnh cho kết quả đầu ra tốt hơn so với cách 1 là (Lọc trung vị + Tích chập với ma trận (Kernel)) ảnh đã được cải thiện rõ rệt vùng màu đen được cải thiện hơn , vùng màu trắng thì giảm được độ chói của bức ảnh ban đầu

(3) Code Python:

```
# -------------------------------------#
# Author: Huu Tri                      #
# Update: 10/12/2022                   #
# Median Filter And Cân bằng Histogram #
# -------------------------------------#
import cv2
import numpy as np

img = cv2.imread('../XLA-cuoiky-2022/final1.bmp', 0)
new_img = cv2.imread('../XLA-cuoiky-
2022/final1.bmp', 0)
hist = np.zeros((256,), np.uint8)
cv2.imshow('anh goc ', img)


#  tính toán histogram
def compute_hist(img):
    hist = np.zeros((256,), np.uint8)
    h, w = img.shape[:2]
    for i in range(h):
        for j in range(w):
            hist[img[i][j]] += 1
    return hist


#  cân bằng histogram
def equal_hist(hist):
    cumulator = np.zeros_like(hist, np.float64)
    for i in range(len(cumulator)):
        cumulator[i] = hist[:i].sum()
    print(cumulator)
    new_hist = (cumulator - cumulator.min()) /
(cumulator.max() - cumulator.min()) * 255
    new_hist = np.uint8(new_hist)
    return new_hist


hist = compute_hist(img).ravel()
new_hist = equal_hist(hist)
```

```
h, w = img.shape[:2]

# thay thế lại tất cả các điểm ảnh từ hàm tính cân
bằng histogram trên
for i in range(h):
    for j in range(w):
        img[i, j] = new_hist[img[i, j]]

new_img = img
prop = new_img.shape

for i in range(1, prop[0] - 1):
    for j in range(1, prop[1] - 1):

        win = []
        for x in range(i - 1, i + 2):
            for y in range(j - 1, j + 2):
                win.append(img[x][y])
        win.sort()

        new_img[i][j] = win[4]

cv2.imshow('image_result', new_img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Câu 2: Phát hiện đường /cạnh

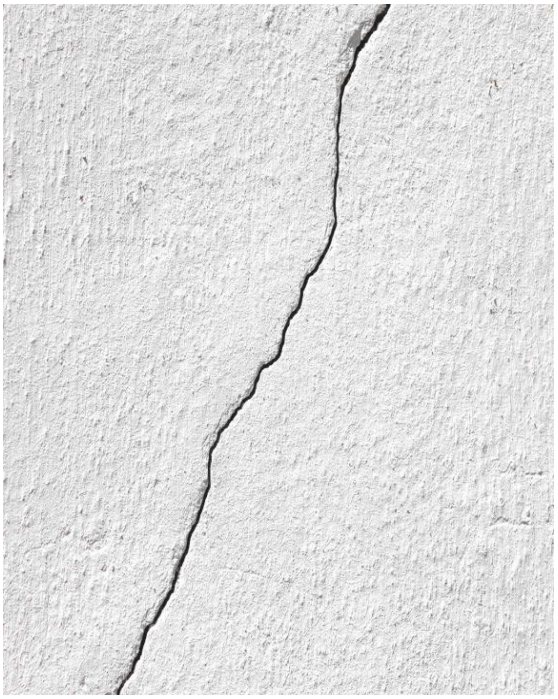
Cách 1 :

(1) Phương pháp và lý do chọn phương pháp:

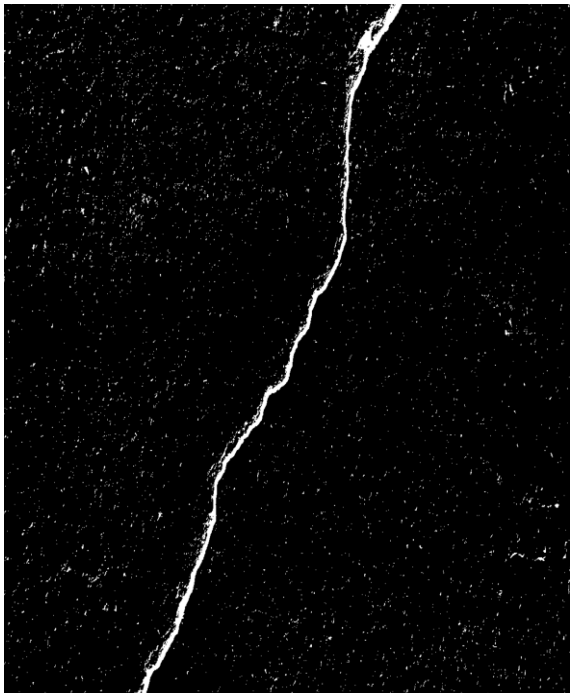
- **Phương pháp :** Sử dụng các bước tiền xử lí như: Chuyển sang ảnh xám sau đó chuyển sang ảnh nhị phân kết hợp với các phép toán hình thái học với phần tử cấu trúc là ma trận 1 (5x5) để khử nhiễu , sau đó dùng Canny() tìm cạnh (Canny(): dựa vào sự khác biệt giữa điểm ảnh vùng ảnh để tìm ra đường biên của đối tượng)
- **Lý do chọn phương pháp:** Vì đây là hình ảnh màu và ta biết rằng ảnh được biểu diễn dưới dạng một ma trận gồm nhiều điểm ảnh , mỗi điểm ảnh sẽ là một hệ số (R,G,B) .Nên muốn phát hiện được vết nứt thì điều đầu tiên là phải chuyển ảnh sang thang xám thì khi đó mỗi điểm ảnh sẽ có giá trị từ 0-255 để khi chuyển tiếp tục sang ảnh nhị phân thì sẽ dễ dàng phân biệt giữa đối tượng cần phát hiện và nền (ví dụ như ảnh đề cho đối tượng là vết nứt và nền là bức tường) thì khi chuyển sang ảnh nhị phân và lấy ngưỡng phù hợp. Do là vết nứt ở trên mặt tường bị sần sùi nên sau khi chuyển sang nhị phân thì sẽ bị ảnh hưởng bởi các đốm trắng li ti nên ta sẽ kết hợp với các phép toán hình thái học để loại bỏ thì vết nứt sẽ hiện một cách rõ rệt và chúng ta chỉ cần tính toán để phân biệt điểm ảnh (pixel) nào thuộc về cạnh (trong Python tích hợp sẵn hàm dùng Canny() để thực hiện công việc này)
  - Ta sử dụng hàm cv2.Canny(image, threshold1, threshold2)
  - Trong đó các tham số :
    + image: Ảnh đầu vào
    + threshold1: Giá trị ngưỡng cao của gradient cường độ.
    + threshold2: Giá trị ngưỡng thấp của gradient cường độ.
    + Hiểu một cách đơn giản của phân ngưỡng (Threshold1 , threshold2) trong Canny() tìm cạnh:
    + Ví dụ : Chúng ta có 3 điểm A, B, C nếu điểm ảnh (Pixel) A cao hơn ngưỡng cao nhất thì điểm ảnh (Pixel) A sẽ là cạnh , Nếu điểm ảnh (Pixel) B nằm giữa ngưỡng cao nhất và ngưỡng thấp nhất thì điểm ảnh (Pixel) B không kết nối với điểm ảnh (Pixel) A , Nếu điểm ảnh (Pixel) C nằm dưới ngưỡng thấp nhất thì điểm ảnh (Pixel) C hoàn toàn không được coi là một cạnh . Mục đích của quá trình này là để loại bỏ (triệt tiêu) các điểm ảnh (pixel) không mong muốn tạo thành cạnh chuyển các điểm ảnh (pixel) này về 0 . Kết quả nhận được là một hình ảnh nhị phân với "các cạnh mỏng".

(2) Kết quả , so sánh nhận xét:

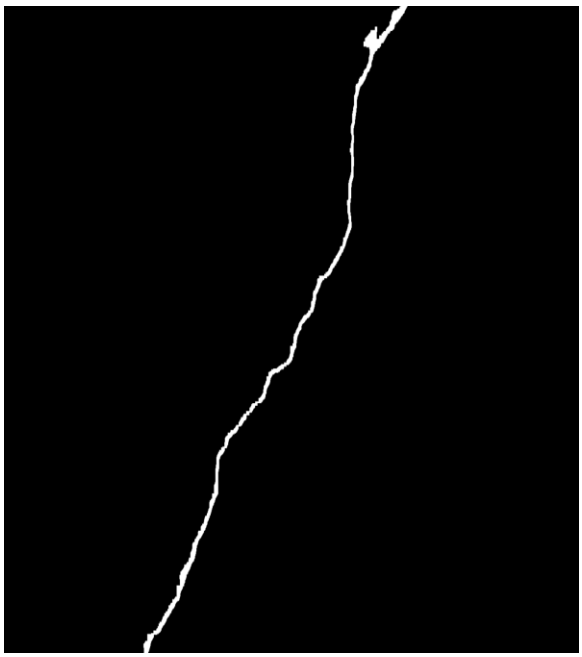
Ảnh Gốc



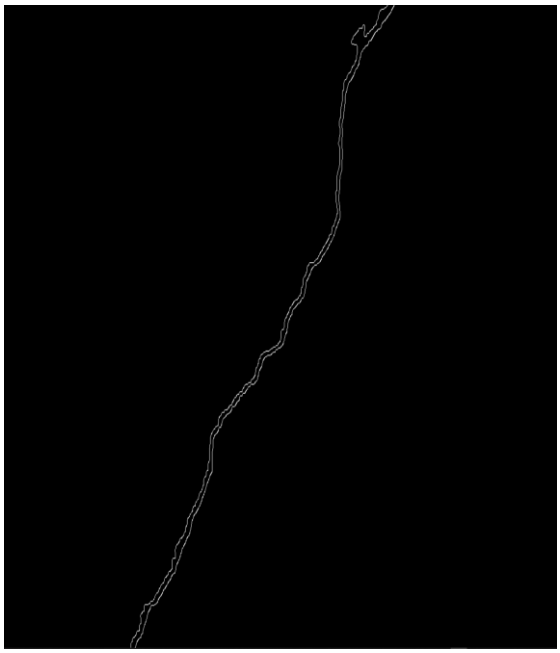
Ảnh sau khi chuyển thành binary



Ảnh sau khi đã được khử nhiễu



Kết quả



- **Nhận xét:**

  + Ta có thể thấy rõ được khi sử dụng các phương pháp tiền xử lý và sau đó dùng(Canny() để tìm cạnh) thì ảnh đầu ra đạt được kết quả khá tốt , phát hiện được gần như 99% vết nứt trên trường

(3) Code Python:

```
# ----------------------------------------#
# Author: Huu Tri                         #
# Update: 10/12/2022                      #
# Phương pháp: Chuyển sang ảnh xám        #
# và chuyển sang nhị phân                 #
# dùng phép hình thái học để lọc bỏ nhiễu  #
# sau đó dùng Canny tìm cạnh              #
# để phát hiện vết nứt trong hình         #
# ----------------------------------------#

import cv2
import numpy as np

img_gray = cv2.imread("../XLA-cuoiky-
2022/final2.jpg", cv2.IMREAD_GRAYSCALE)

ret, thresh = cv2.threshold(img_gray, 160, 255,
cv2.THRESH_BINARY_INV)

cv2.imshow("img_binary", thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()

kernel = np.ones((5, 5), np.uint8)
mask2 = cv2.morphologyEx(thresh, cv2.MORPH_OPEN,
kernel)
mask2 = cv2.morphologyEx(mask2, cv2.MORPH_CLOSE,
kernel)

cv2.imshow("img_filter", mask2)
cv2.waitKey(0)
cv2.destroyAllWindows()

edges = cv2.Canny(mask2, threshold1=50,
threshold2=600)

cv2.imshow("result", edges )
cv2.imwrite("result_ex_2.jpg", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Cách 2:

(1) Phương pháp và lý do chọn phương pháp:

- **Phương pháp:** Sử dụng các bước tiền xử lí như : chuyển sang ảnh xám sau đó chuyển sang ảnh nhị phân kết hợp với các phép toán hình thái học với phần tử cấu trúc là ma trận 1 (5x5) để khử nhiễu , sau đó dùng tích chập với ma trận Kernel để phát hiện cạnh

- **Lý do chọn phương pháp:** Ngoài phương pháp sử dụng Canndy để tìm cạnh thì còn một số phương pháp khác như (Sobel ,…) thì còn một phương pháp khá đơn giản để phát hiện đó là phép chập với một ma trận Kernel phát hiện cạnh
- Ta sử dụng hàm cv2.filter2D(image , depth , kernel) hàm này có chức năng kết hợp ma trận 2d với hình ảnh ở mức pixel và tạo ra hình ảnh đầu ra cụ thể được thực hiện bằng cách nhân các giá trị của mỗi giá trị pixel với giá trị hạt nhân ở các vị trí tương ứng và thêm tất cả các giá trị vừa được kernel tính trọng số bằng cách nhân và tạo thành một pixel lặp lại cho đến hết các pixel trong bức ảnh

- 3 tham số trong hàm là :

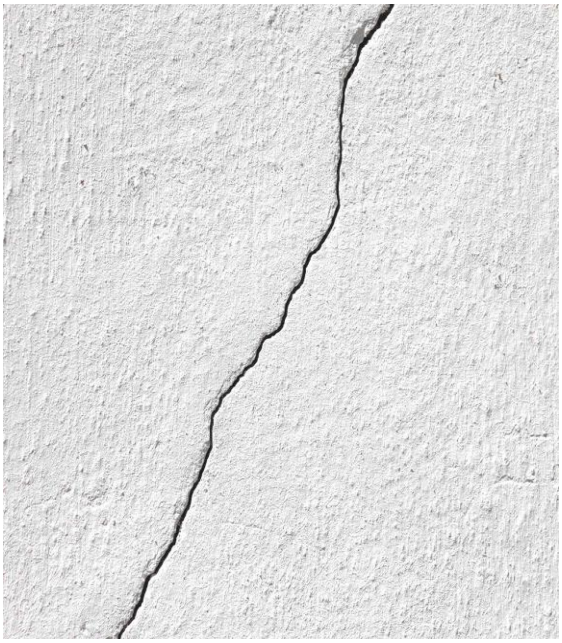
  + image : ảnh truyền vào

  + depth : mặc định là -1 (nghĩa là độ sâu của ảnh đầu ra giống với độ sâu của ảnh đầu vào)

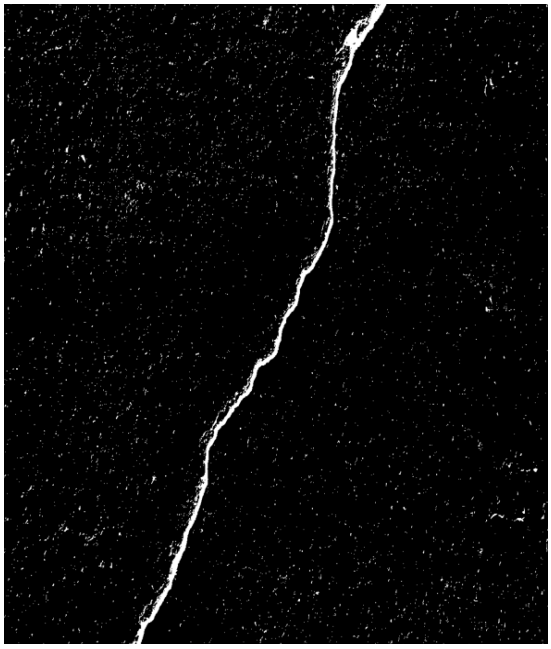
  + Kernel : ma trận dùng để tích chập với ảnh cần xử lý (Kernel có nhiều loại khác nhau . Vidu: Kernel để làm sắc nét , Kernel để tìm cạnh , Kernel làm mờ ,….)

(2) Kết quả , so sánh , nhận xét:

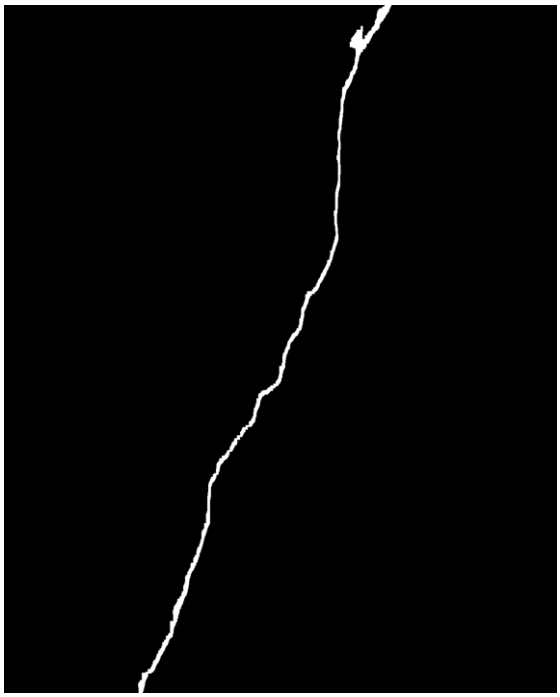
Ảnh Gốc



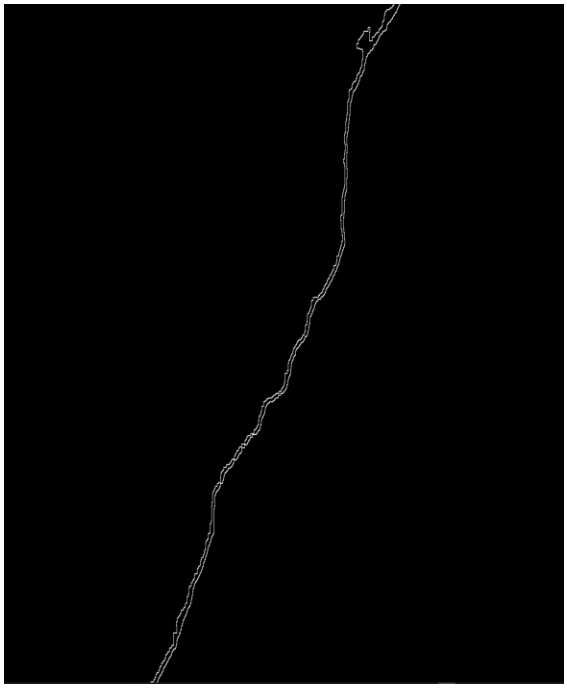
Ảnh sau khi chuyển thành binary



Ảnh sau khi đã được khử nhiễu



Kết quả



- **Nhận xét :** Ta có thể thấy rõ khi sử dụng các phương pháp tiền xử lý và sau đó dùng tích chập với Kernel thì ảnh đầu ra đạt được kết quả khá tốt , phát hiện được gần như 99% vết nứt trên tường nhờ vào quá trình tiền xử lý ảnh bằng các phương pháp (lấy ngưỡng sang ảnh nhị phân kết hợp với các phép toán hình thái học) nên kết quả đầu ra của phương pháp chập bằng Kernel tương tự với sử dụng Canny nhưng có vài điểm ảnh lại bị điểm màu.

- **Kết luận:** Phương pháp của cách 2 nó không tốt bằng Cách 1 (Tiền xử lý (lấy ngưỡng, dùng hình thái học để loại bỏ nhiễu) sau đó dùng Canny()) nhưng đối với Cách 1 dùng Canny() thì yêu cầu độ về quy tình , tính toán phức tạp hơn so với sử dụng tích chập của cách 2

(3) Code Python:

```
# ------------------------------------------------#
# Author: Huu Tri                                 #
# Update: 11/12/2022                              #
# Phương pháp: Chuyển sang ảnh xám                #
# và chuyển sang nhị phân                         #
# dùng phép hình thái học để lọc bỏ nhiễu         #
# sau đó dùng tích chập với ma trận Kernel        #
# để phát hiện vết nứt trong hình                 #
# ------------------------------------------------#

import cv2
import numpy as np

img_gray = cv2.imread("../XLA-cuoiky-
2022/final2.jpg", cv2.IMREAD_GRAYSCALE)

ret, thresh = cv2.threshold(img_gray, 160, 255,
cv2.THRESH_BINARY_INV)

cv2.imshow("image_binary", thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()

kernel = np.ones((5, 5), np.uint8)
mask2 = cv2.morphologyEx(thresh, cv2.MORPH_OPEN,
kernel)
mask2 = cv2.morphologyEx(mask2, cv2.MORPH_CLOSE,
kernel)

cv2.imshow("image_remove_noise", mask2)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()

kernel2 = np.array([[-1, -1, -1],
                    [-1, 8, -1],
                    [-1, -1, -1]])

# Applying the filter2D()
img = cv2.filter2D(src=mask2, ddepth=-1,
kernel=kernel2)

cv2.imshow('Result_Kernel', img)

cv2.waitKey()
cv2.destroyAllWindows()
```

Câu 4 : Ứng dụng các phép toán hình thái học

(1) Phương pháp và lý do chọn phương pháp

- **Phương pháp:** Lấy ngưỡng ảnh chuyển về ảnh nhị phân , sử dụng phép toán hình thái học (erosion, closing và opening) , dùng Canny để trích xuất đường bao
- **Lý do chọn phương pháp**: Để sử dụng phương pháp Canny đạt được kết quả tốt nhất thì ta cần phải phân ngưỡng , và sử dụng các phép toán hình thái để loại bỏ được nhiễu
- Mô tả phương pháp hình thái học hàm tích hợp sẵn trong Python cv2.morphologyEx(Image, cv2.MORPH_CLOSE, kernel):
  - Phép toán hình thái học cv2.MORPH_CLOSE trong Python sẽ thực hiện phép hình thái học Dilation sau đó sử dụng Erosion
  - Để sử dụng được phép hình thái học Closing trong Python thì cần sử dụng hàm cv2.morphologyEx(Image, Morphological, kernel) gồm các tham số như sau :

    + Image: Ảnh đầu vào

    + Morphological: Phép toán hình thái muốn thực hiện ở đây ta sử dụng phép đóng (Closing) thì trong Python sẽ thực hiện (cv2.MORPH_CLOSE) , ví dụ với phép mở (cv2.MORPH_OPEN)

    + kernel: Phần tử cấu trúc , phần tử cấu trúc này được ta định nghĩa sẵn ban đầu , mục đích để quét qua các điểm ảnh (Pixel) với ma trận đầu vào để xác định xem điểm ảnh (Pixel) nào được loại bỏ , điểm ảnh (Pixel) nào được giữ lại , điểm ảnh (Pixel) nào phù hợp để thay thế bằng điểm ảnh (Pixel) mới thực hiện giãn ảnh
- Hàm Canny() trong CV2 của Python :

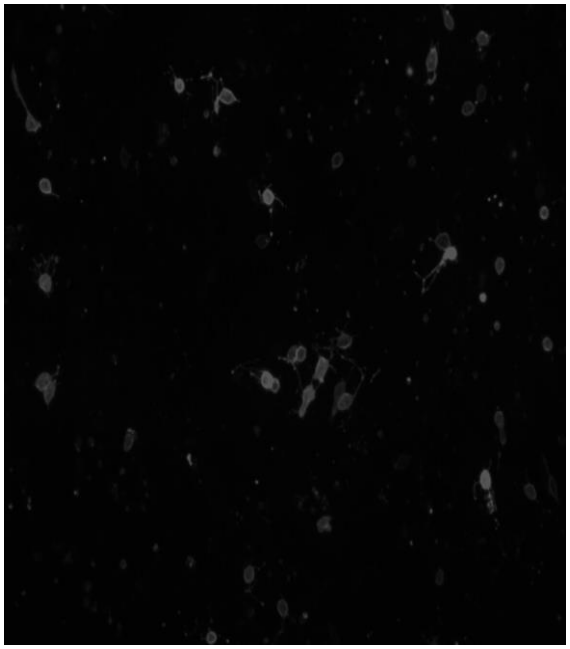
  + Thuật toán này thực hiện theo các quy trình như sau :
  - Bước 1 : Giảm nhiễu (Noise reduction) : Vì hình ảnh xử lý có thể nhiễu nên cần phải trải qua bước này để giảm nhiễu (Có thể thực hiện bộ lọc Gaussian để ảnh trả về được mịn hơn)
  - Bước 2 : Tính toán độ dốc của ảnh (Gradient Calculation): tính toán Gradient để phát hiện cường độ và hướng của cạnh. Có thể hiểu là tìm cường độ mạnh yếu của cạnh tại mỗi điểm ảnh (sử dụng toán tử Sobel sử dụng một cặp mặt nạ tích chập 3x3 để ước lượng Gradient theo hướng X và theo hướng Y)
  - Bước 3 : Triệt tiêu phi tối đa (Non-Maximum supperssion): Hình ảnh cuối cùng là cách cạnh mỏng nên chúng ta cần phải triệt tiêu để làm mảnh các

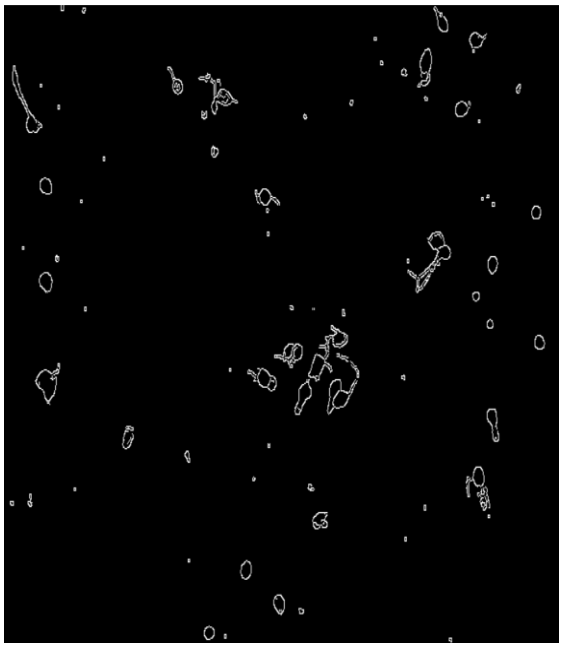
cạnh , thuật toán sẽ đi qua tất cả các điểm ảnh (Pixel) trên ma trận Gradient tìm tất cả điểm ảnh (Pixel) có giá trị lớn nhất

- Bước 4: Ngưỡng kép (Double threshold) : Mục đích để xác định 3 loại điểm ảnh (Mạnh (nếu điểm ảnh lớn hớn ngưỡng lớn nhất), yếu (Nhỏ hơn ngưỡng nhỏ nhất), không liên quan(Nằm giữa ngưỡng lớn nhất và ngưỡng nhỏ nhất))
- Bước 5: Edge tracking by Hysteresis: Được xử lý chuyển đổi pixel yếu thành mạnh nếu chỉ khi tất ít nhất một trong các điểm ảnh (Pixel) xung quanh được xử lý là điểm ảnh (Pixel) mạnh . Thì điểm ảnh được coi là yếu sẽ chuyển sang mạnh tránh được quá trình đường bao đối tượng bị đứt gãy , đường bao không bao hết đối tượng

(2) Kết quả , nhận xét:

Ảnh gốc **Ảnh sau khi sử lý**





- **Kết quả và nhận xét :** Từ kết quả thu được ta thấy ảnh đầu ra đã phát hiện rõ , và đường bao đã bao quanh tế bào, tuy nhiên ở một số tế bào vẫn chưa phát hiện được .
- **Kết luận:** Thuật toán Canny phụ thuộc nhiều vào các tham số có thể điều chỉnh như độ lệch chuẩn cho bộ lọc Gaussian , lựa chọn các giá trị ngưỡng cao , thấp thì chúng ta có thể điều chỉnh cho phù hợp với các tham số này . Thuật toán Canny này rất phứt tạp về mặt tính toán so với các thuật toán khác như (Sobel , Robert ,….). Tuy nhiên thuật toán phát hiện cạnh của Canny tốt hơn các thuật toán khi trong hầu hết các tình huống

- Để cải thiện hơn trong quá trình tìm đường bao thì ta có thể tìm hiểu về các lựa chọn ngưỡng cho phù hợp bằng thuật toán chọn ngưỡng OTSU

(3) Code Python :

```
# ------------------------------------------------#
# Author: Huu Tri                                 #
# Update: 12/12/2022                              #
# Phương pháp:                                    #
# Lấy ngưỡng chuyển sang nhị phân                 #
# dùng phép hình thái học để lọc bỏ nhiễu         #
# sau đó dùng Canny tìm đường bao                 #
# ------------------------------------------------#

import cv2
import numpy as np

image = cv2.imread('../XLA-cuoiky-2022/final4.jpg')
image = cv2.resize(image, (0, 0), fx=0.5, fy=0.5)

ret, edges = cv2.threshold(image, 150, 255,
cv2.THRESH_BINARY_INV)

kernel = np.ones((3, 3), np.uint8)

#  thực hiện giãn nở (Dilation) sau đó Co Erosion
mask2 = cv2.morphologyEx(edges, cv2.MORPH_CLOSE,
kernel)

edges = cv2.Canny(image, 40, 170)

cv2.imshow('edges', edges)

cv2.imshow('image', image)

cv2.waitKey(0)
```