Báo cáo cài đặt giải thuật Background Removing-LQN

# Thuật toán Background Removing LQN v1

## Giải thuật

Giải thuật **Background Removing-LQN-v1**

**Input**: Ảnh màu I.

**Output**: Ảnh J.

**Các bước thực hiện**:

• Chuyển ảnh màu I sang grayscale gọi là Ig.

• Thực hiện inverse Ig thành Iginv = 255 - Ig (vì chữ tối hơn nền).

• Tạo ảnh J cùng kích thước với Iginv (J có giá try 0 ở tất cả các pixel ngoại trừ các pixel ở biên. Tại biên của ảnh J, các pixel sẽ có giá trị bằng với giá trị ở cùng vị trí với pixel trong ảnh gốc Iginv).

• Khởi tạo ảnh 𝐾 cùng kích thước với Iginv.

• Lặp lại cho đến khi ổn định (không có pixel nào trong J thay đổi giá trị)

− Bước 1: Với mỗi pixel 𝑝 ∈ Iginv

𝐾(𝑝) ← 𝑚𝑎𝑥{𝐽(𝑞), 𝑞 ∈ 𝑁\_G(𝑝) ∪ {𝑝}}

N\_G (p): các pixel trong vùng lận cận G của p

− Bước 2: Với mỗi pixel 𝑝 ∈ Iginv

𝐽(𝑝) ← min{𝐾(𝑝), Iginv (𝑝)}

• Tạo ảnh BR(p) = Iginv(𝑝) – 𝐽(𝑝) (ảnh đã loại bớt background).

• Tiếp tục thực hiện phân ngưỡng toàn cục hoặc cục bộ để có ảnh kết quả mong muốn.

## Cài đặt

from copy import deepcopy

import numpy as np

import cv2

import os

from time import time

def bg\_rm\_lqn(

    I: np.ndarray, # grayscale

)-> np.ndarray:

    s = time()

    # region 0. Read the image and preprocess the image

    I = 255 - I  # Inverse pixel

    H, W = I.shape

    J = np.zeros(I.shape, dtype=np.uint8)

    K = np.zeros(I.shape, dtype=np.uint8)

    J[0, :] = I[0, :] # Top edge

    J[H-1, :] = I[H-1, :] # Bottom edge

    J[:, 0] = I[:, 0] # Left edge

    J[:, W-1] = I[:, W-1] # Right

    e = time()

    new\_J = deepcopy(J)

    # endregion

    count= 0

    lst\_time\_step\_1 = []

    lst\_time\_step\_2 = []

    while True:

        count+=1

        # region 1. Image K

        s = time()

        K = cv2.dilate(

            src = J,

            kernel= cv2.getStructuringElement(

                shape= cv2.MORPH\_RECT,

                ksize= (3,3)

            ),

            iterations= 1,

        )

        # print(f'K: {K}')

        e = time()

        lst\_time\_step\_1.append((e-s))

        # endregion

        # region 2. Image J

        # Step 2

        s = time()

        new\_J = cv2.min(

            src1 = K,

            src2 = I,

        )

        # endregion

        if cv2.countNonZero(cv2.subtract(new\_J,J)) == 0:

        # if np.all(J == new\_J):

            break

        J = new\_J

        e = time()

        lst\_time\_step\_2.append((e-s))

    BR = I - J

    print(f'Count: {count}')

    print(f"Average time step 1: {np.mean(lst\_time\_step\_1)}")

    print(f"Average time step 2: {np.mean(lst\_time\_step\_2)}")

    return BR, J

def run\_opencv\_morphology():

    # image\_fn= 'test.png'

    # image\_fn = 'in1.jpg'

    # image\_fn = '35227268\_1442578812555603\_8324419005091676160\_n.jpg'

    # image\_fn = 'IMG/sptn-page3-fig1-so01.jpeg'

    image\_fn = 'IMG/sptn-page3-fig2-so02.jpeg'

    image\_fn = 'IMG/sptn-page7-fig2-so13.jpeg'

    addition\_fn= 'opencv\_morphology'

    ROOT = ''

    # image\_fn = 'D:/Master/OCR\_Nom/fulllow\_ocr\_temple/remove\_text/background\_removeing\_lqn/test\_2.png'

    # image\_fn = 'D:/Master/OCR\_Nom/fulllow\_ocr\_temple/remove\_text/background\_removeing\_lqn/test\_4.png'

    # region get filename

    img\_name = image\_fn.split('/')[-1].split('.')[0]+'\_'+ addition\_fn

    fd\_name = os.path.join(ROOT, img\_name)

    os.makedirs(

        name = fd\_name,

        exist\_ok= True

    )

    # endregion

    I = cv2.imread(image\_fn, cv2.IMREAD\_GRAYSCALE)

    print(f'Shape image: {I.shape}')

    # I = np.random.randint(

    #     low = 0,

    #     high= 256,

    #     size = (5, 5),

    #     dtype= np.uint8

    # )

    s = time()

    BR, J = bg\_rm\_lqn(

        I = I,

    )

    e = time()

    print(f"Time remove background with code python: {e-s} s")

    # BR = I - J

    \_, thresh = cv2.threshold(BR, 0, 255, cv2.THRESH\_BINARY)

    cv2.imwrite(

        filename= os.path.join(fd\_name, f'I\_{img\_name}.png'),

        img= I

    )

    cv2.imwrite(

        filename= os.path.join(fd\_name, f'J\_{img\_name}.png'),

        img = J

    )

    cv2.imwrite(

        filename= os.path.join(fd\_name,f'BR\_{img\_name}.png'),

        img = BR

    )

    cv2.imwrite(

        filename= os.path.join(fd\_name, f'BR\_threshold\_{img\_name}.png'),

        img = thresh

    )

    cv2.imwrite(

        filename= os.path.join(fd\_name, f'J\_reinverse\_{img\_name}.png'),

        img = 255 - J

    )

if \_\_name\_\_ == "\_\_main\_\_":

    run\_opencv\_morphology()

## Kết quả

### Test 1

A close up of a black and white background

Description automatically generatedA black background with white characters

Description automatically generatedA grey and black background

Description automatically generated with medium confidenceA close up of a sign

Description automatically generated

D

C

B

A

Trong đó:

1. Ảnh màu ban đầu
2. Ảnh xám
3. Ảnh sau khi xóa nền
4. Ảnh tiền cảnh

### Test 2

A black background with writing on it

Description automatically generatedA close-up of a grey and white background

Description automatically generatedA close-up of a greyscale image

Description automatically generated

D

C

B

A

### Test 3

A close up of a fabric

Description automatically generatedA close up of a fabric

Description automatically generated

A

A black screen with white writing

Description automatically generatedA close-up of a carpet

Description automatically generated

B

D

C

### Test 4

A gold curtain with a pattern

Description automatically generated

A

A close-up of a curtain

Description automatically generatedA close-up of a curtain

Description automatically generated

C

B

A black screen with white dots

Description automatically generated

D

Số vòng lặp để thực thi test 4

## Quá trình cải thiện tốc độ

### Phương pháp ngây thơ nhất

Với phương pháp ngây thơ nhất, tôi dùng 2 vòng lặp for để duyệt qua các phần tử trong ảnh. Với mỗi điểm ảnh (pixel) tôi sẽ lấy các điểm ảnh thuộc lân cận 9 của điểm ảnh đang xét.

A computer screen shot of a program code

Description automatically generated

### Sử dụng bảng băm để lưu lại tọa độ các lân cận

Ý tưởng: Sau mỗi lần lặp while chúng ta lại phải duyệt qua bức ảnh thêm một lần nữa để tính các điểm ảnh lân cận. Do đó, tôi đề xuất sẽ dung một bảng băm để lưu lại tọa độ các điểm ảnh lân cận của một điểm ảnh đang xét. Khi đó, việc truy xuất bảng băm để lấy ra các tọa độ cần thiết chỉ là thay vì như phương pháp ngây thơ nhất.

### Sử dụng Cython

**Cython là gì?**

Cython là tiện ích của Python hỗ trợ việc gọi các hàm C/C++ và khai báo các kiểu dữ liệu C trên các biến và thuộc tính của lớp.

**Những cải tiến thực hiện trong Cython**

* Khai báo thêm kiểm dữ liệu như C/C++.
* Sử dụng memoryview để cải thiện tốc độ đọc và ghi trên ảnh.
* Sử dụng:
* @cython.boundscheck(False): Tránh kiểm tra chỉ mục có hợp lệ không
* @cython.wraparound(False):Không xử lý trường hợp âm

**Ưu điểm**: Tận dụng được các hàm trong C/C++

**Nhược điểm**:

* Cần phải cài đặt môi trường chạy C.
* Cần biên dịch thành mã máy.

### Dựa trên ý tưởng tích chập (convolution)

Tôi kế thừa ý tưởng từ bài viết: <https://medium.com/analytics-vidhya/implementing-convolution-without-for-loops-in-numpy-ce111322a7cd> , trong bài viết tác giải gợi ý cách lập trình phép toán tích chập mà không cần dùng vòng lặp, chỉ sử dụng các hàm của numpy.

**Mã nguồn lấy các điểm ảnh lân cận theo ý tưởng tích chập**

def get\_max\_pixel\_convonlution(

    image: np.ndarray

)-> np.ndarray:

    # region hyper\_paramater

    img\_h , img\_w = image.shape

    ker\_h, ker\_w = 3,3

    pad , stride = 1,1

    pad\_value = 0.0

    square\_img = np.zeros(2 \* [max(image.shape)], dtype=np.uint8)

    square\_img[:img\_h, :img\_w] = image

    # endregion

    # region get mask

    mask = np.zeros(2 \* [max(image.shape)], dtype=np.uint8)

    valid\_image = np.ones\_like(image)

    mask[:img\_h, :img\_w] = valid\_image

    # endregion

    # region padding

    pad\_img = np.pad(

        array= square\_img,

        pad\_width= (pad, pad),

        mode='constant',

        constant\_values=0

    )

    img\_h , img\_w = square\_img.shape

    # endregion

    # region index caculate

    i0=np.repeat(np.arange(ker\_h), ker\_h)

    i1=np.repeat(np.arange(img\_h), img\_h)

    j0=np.tile(np.arange(ker\_w), ker\_h)

    j1=np.tile(np.arange(img\_h), img\_w)

    i=i0.reshape(-1,1)+i1.reshape(1,-1)

    j=j0.reshape(-1,1)+j1.reshape(1,-1)

    # endregion

    # region calculate maximum pixel

    select\_img=pad\_img[i,j].squeeze().transpose()

    K = np.max(select\_img, axis=-1).reshape(img\_h, img\_w)

    # endregion

    # region bitwise and

    res = np.multiply(K,mask)

    nz = np.nonzero(res)  # Indices of all nonzero elements

    arr\_trimmed = res[nz[0].min():nz[0].max()+1,

                    nz[1].min():nz[1].max()+1]

    # endregion

    return arr\_trimmed

**Ưu điểm**: Cải thiện tốc độ tính toán

**Nhược điểm**: Yêu cầu ảnh vuông do đó phải tốn thêm chi phí padding và tích chập vào vùng padding thường vô nghĩa.

### Sử dụng sliding window của numpy

Tôi sử dụng hàm [numpy.lib.stride\_tricks.sliding\_window\_view](https://numpy.org/devdocs/reference/generated/numpy.lib.stride_tricks.sliding_window_view.html) để lấy lân cận của một điểm ảnh. Phương pháp này sẽ sử dụng một cửa sổ trượt để chạy từ trái sang phải, từ trên xuống dưới. Với việc xét cửa sổ là tương đương với việc tôi lấy được lân cận của một điểm ảnh.

**Ưu điểm**: Tính toán nhanh và dễ dàng cài đặt cách để lấy các điểm lân cận.

**Nhược điểm**:

Cần phải gán bằng 0 cho các phần tử ở biên của ảnh trước khi áp dụng numpy.lib.stride\_tricks.sliding\_window\_view.

        pad\_img = np.pad(

            array= J,

            pad\_width= (1,1),

            mode='constant',

            constant\_values=0

        )

Mã nguồn của numpy.lib.stride\_tricks.sliding\_window\_view vẫn chưa quá tối ưu, do đó có thể cải thiện thêm về tốc độ.

### Sử dụng hàm OpenCV

Sử dụng hoàn toàn các hàm của OpenCV:

* cv2.dilate để lấy điểm ảnh lớn nhất trong các điểm ảnh lân cận (bước 1).

        K = cv2.dilate(

            src = J,

            kernel= cv2.getStructuringElement(

                shape= cv2.MORPH\_RECT,

                ksize= (3,3)

            ),

            iterations= 1,

        )

* cv2.min để lấy (bước 2).

        new\_J = cv2.min(

            src1 = K,

            src2 = I,

        )

* cv2.subtract và cv2.countNonZero để kiểm tra điều kiện dừng.

        if cv2.countNonZero(cv2.subtract(new\_J,J)) == 0:

            break

**Ưu điểm**: Tận dụng được các hàm OpenCV được viết bằng C++.

### So sánh các phương pháp

#### Ảnh có kích thước nhỏ

Để kiểm tra tính hiểu quả và đo thời gian thực thi cho tất cả các phương pháp một cách nhanh chóng, tôi sử dụng ảnh kiểm tra có kích thước .

A close up of a sign

Description automatically generated

**Kết quả thực thi tất cả các phương pháp**

A computer screen shot of a program

Description automatically generated

#### Ảnh có kích thước lớn

Để đánh giá hiệu quả cải tiến thuật toán tôi sử dụng ảnh có kích thước để thực thi trên 3 phương pháp lần lượt là: Phương pháp ngây thơ nhất (baseline), phương pháp sử dụng sliding window của numpy và phương pháp sử dụng các hàm của OpenCV.

|  |  |
| --- | --- |
| Phương pháp | Thời gian thực thi (giây) |
| Phương pháp ngây thơ nhất (baseline) | 4377 |
| Phương pháp sử dụng sliding window của numpy | 132 |
| Phương pháp sử dụng các hàm của OpenCV | 3.5 |

**Chi tiết quá trình chạy baseline**

A computer screen shot of a program

Description automatically generated

**Chi tiết quá trình chạy phương pháp sử dụng sliding window của Numpy**

A screenshot of a computer code

Description automatically generated

**Chi tiết phương pháp sử dụng các hàm của OpenCV**

A computer screen shot of numbers and letters

Description automatically generated

# Phương pháp Background LQN v2

## Giải thuật

A math problem with black text

Description automatically generated with medium confidence

## Mã nguồn

import argparse

import cv2

ROOT = ''

import os

APPROVED\_PACKAGES ='custome\_opening'

B = 20

ITER = 1000

KSIZE = (13,13)

if \_\_name\_\_ == "\_\_main\_\_":

    # if not os.path.exists(ROOT):

    #     os.makedirs(ROOT, exist\_ok= True)

    # region construct the argument parser and parse the arguments

    ap = argparse.ArgumentParser()

    ap.add\_argument("-i", "--image", required=True,

    help="path to input image")

    args = vars(ap.parse\_args())

    # endregion

    # region load the image, convert it to grayscale, and display it to our screen

    image = cv2.imread(args["image"])

    gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    gray = 255 - gray

    # endregion

    # region get file name

    fn = args["image"].split('/')[-1].split('.')[0]

    fd\_path = os.path.join(ROOT, f'{fn}\_\_{APPROVED\_PACKAGES}\_\_{B}\_\_iter{ITER}\_\_kernel\_{KSIZE[0]}\_{KSIZE[1]}')

    os.makedirs(

        name = fd\_path,

        exist\_ok= True

    )

    cv2.imwrite(org\_path, gray)

    # endregion

    # region Apply a series of opening operationss

    kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, KSIZE)

    eroded = cv2.erode(gray.copy()-B, None, iterations=ITER)

    dilated = cv2.dilate(eroded+B, None, iterations=ITER)

    h = gray- dilated

    # endregion

    # region save file

    gray\_fn = os.path.join(fd\_path, f"{fn}\_\_grayscale.png")

    h\_fn = os.path.join(fd\_path, f"{fn}\_\_h\_{KSIZE[0]}\_{KSIZE[1]}.png")

    cv2.imwrite(gray\_fn, gray)

    # cv2.imwrite(dilated\_fn, dilated)

    cv2.imwrite(h\_fn, h)

    # endregion

## Thay đổi giá trị

## Thay đổi giá trị kernel

## Thay đổi số vòng lặp (iteration)

## Black hat morphology

# import the necessary packages

import argparse

import cv2

import os

ROOT = 'OUTPUT'

APPROVED\_PACKAGES ='blackhat\_tohat'

ITER = 1000

if \_\_name\_\_ == "\_\_main\_\_":

    if not os.path.exists(ROOT):

        os.makedirs(ROOT, exist\_ok= True)

    # region construct the argument parser and parse the arguments

    ap = argparse.ArgumentParser()

    ap.add\_argument("-i", "--image", required=True,

    help="path to input image")

    args = vars(ap.parse\_args())

    # endregion

    # region load the image, convert it to grayscale, and display it to our screen

    image = cv2.imread(args["image"])

    gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    # endregion

    # region get file name

    fn = args["image"].split('/')[-1].split('.')[0]

    fd\_path = os.path.join(

        ROOT,

        f'{fn}\_\_{APPROVED\_PACKAGES}'

    )

    os.makedirs(

        name = fd\_path,

        exist\_ok= True

    )

    org\_path = os.path.join(ROOT,fd\_path,f'{fn}\_grayscale.png')

    cv2.imwrite(org\_path, gray)

    # endregion

    # region Apply a series of opening operationss

    kernelSizes = [(3,3), (5,5), (7,7), (9,9), (11,11), (13,13)]

    # loop over the kernels sizes

    for kernelSize in kernelSizes:

        # construct a rectangular kernel from the current size and then

        # apply an "opening" operation

        kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, kernelSize)

        # opening = cv2.morphologyEx(gray, cv2.MORPH\_OPEN, kernel)

        blackhat = cv2.morphologyEx(gray, cv2.MORPH\_BLACKHAT, kernel,iterations=ITER)

        tophat = cv2.morphologyEx(gray, cv2.MORPH\_TOPHAT, kernel, iterations=ITER)

        black\_hat\_fn = os.path.join(fd\_path, f"{fn}\_\_Blackhat\_{kernelSize[0]}\_{kernelSize[1]}.png")

        to\_hat\_fn = os.path.join(fd\_path, f"{fn}Tohat{kernelSize[0]}\_{kernelSize[1]}.png")

        cv2.imwrite(black\_hat\_fn, blackhat)

        cv2.imwrite(to\_hat\_fn, tophat)

    # endregion

## Histogram Equalization

Adaptive

## Chia tập train - test

## Range grayscale

# Đánh giá sự hiệu quả của thuật toán trong việc phát hiện và nhận diện ký tự Hán Nôm

Chạy blendtext qua tập toàn bộ tập dữ liệu

Chạy blendtext qua ảnh từ phương pháp 1

Chạy blendtext qua ảnh từ phương pháp 2