BÀI TẬP LỚN 1

Trí tuệ nhân tạo

Nhóm 3

Ngày 25 tháng 2 năm 2024

Sinh viên thực hiện
106200284 - Hồ Đức Vũ - 20KTMT2
106200240 - Nguyễn Minh Phương - 20KTMT1
106200241 - Huỳnh Vũ Đình Phước - 20KTMT1

Giáo viên hướng dẫn
TS. Hoàng Lê Uyên Thục

Môn học
Trí tuệ nhân tạo

Đề bài
Nhận diện quả xoài trong ảnh sử dụng phương pháp **Grid to grid comparision**

Xuất bản
Đà Nẵng, Ngày 25 tháng 2 năm 2024

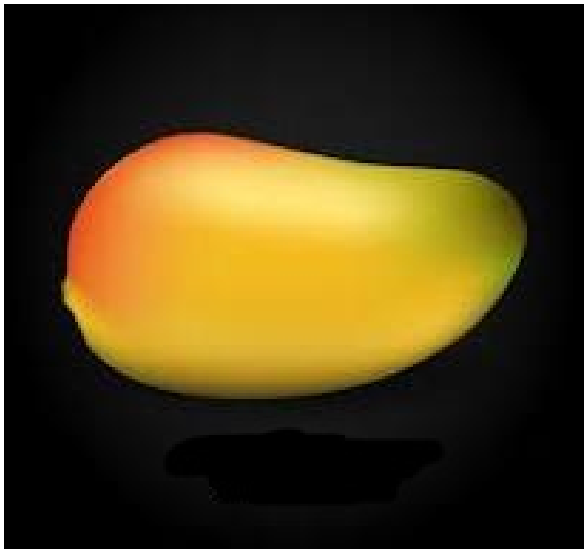
Số trang
10

Mục lục

Nội dung báo cáo

1 Thu thập dữ liệu

Nhóm thu thập dữ liệu từ google images, dữ liệu thu thập bao gồm 4 hình ảnh - 1 hình ảnh mẫu (hình 1.1) và 3 hình ảnh test (hình 1.2) - đối với ảnh test bao gồm 2 hình ảnh chứa một quả xoài và hình còn lại chứa một trái ớt. Cả 4 hình đều có background đen, lý do cho background là nền đen là khi ta sử dụng phương pháp grid to grid comparision thì ta cần chuyển ảnh sang nhị phân rồi về dạng grid, nếu background của ảnh là trắng thì dữ liệu sẽ không có sự phân biệt giữa đối tượng và nền - đều có giá trị là 1.



Hình 1.1: Hình ảnh mẫu

2 Quá trình xử lý

1. Điều chỉnh size ảnh mẫu:

Ảnh mẫu được điều chỉnh kích thước để làm khung mẫu cho bài toán, được gọi là patter_window.

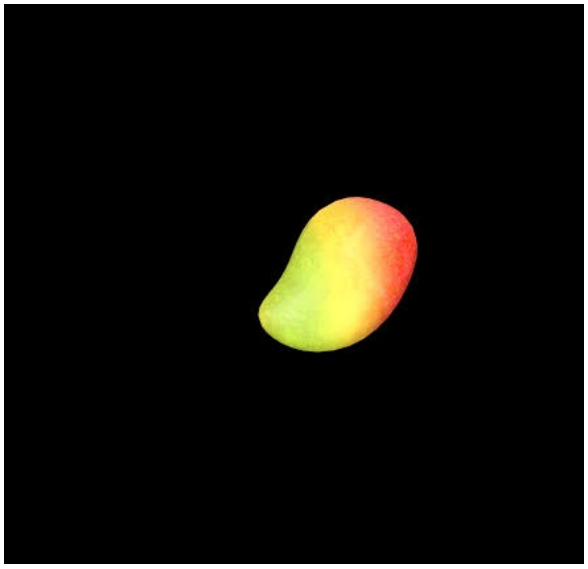
Nhóm đã thử nghiệm các size window - khung vuông - khác nhau từ (50, 50) đến (100, 100) sau các thử nghiệm, nhóm nhận thấy size window có giá trị (80, 80) là kích thước window tốt nhất trong bài toán nhóm thực hiện.

2. Tạo window trên ảnh test và thực hiện slide window:

Tạo một khung window có kích thước tương tự với kích thước patter_window - (80,80) - trên ảnh test, sau đó cho window trượt trên ảnh test, với mỗi lần trượt nhóm trích xuất hình

(a) Hình ảnh 1

(b) Hình ảnh 2



(c) Hình ảnh 3

Hình 1.2: Hình ảnh test

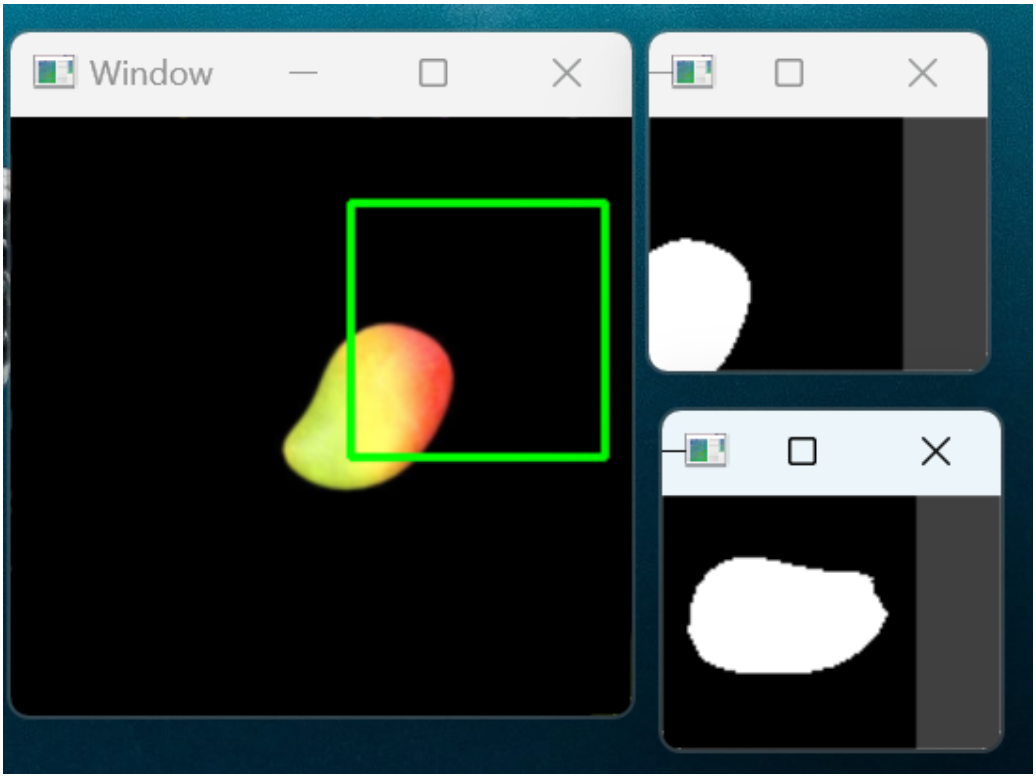
ảnh từ khung window này thành một ảnh có kích thước (80,80) tương ứng, gọi là test_window.

3. Thực hiện chuyển ảnh patter_window và test_window thành ảnh nhị phân, gọi 2 ảnh này là patter_window_binary và test_window_binary, mô tả tại hình 2.1.

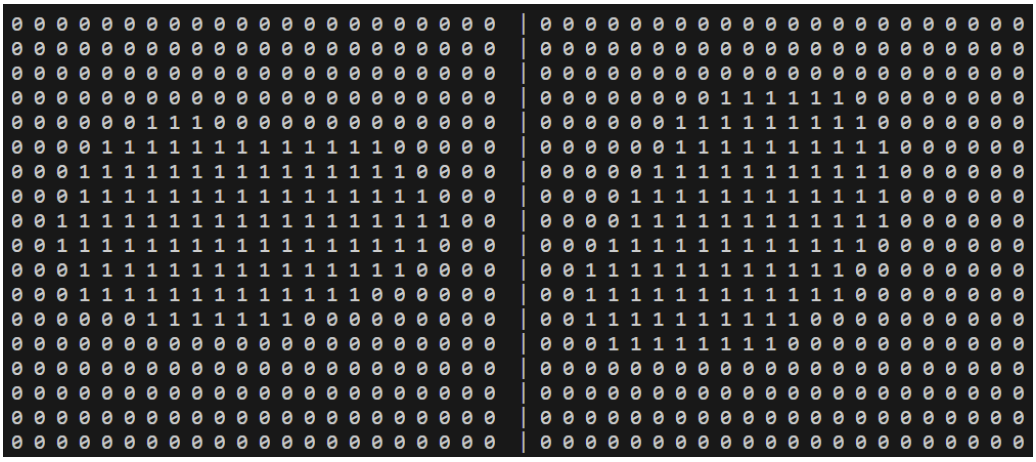
4. Sau khi đã có ảnh nhị phân như trên, nhóm tiếp tục thực hiện chuyển đổi ảnh nhị phân có được sang dạng grid, giảm kích thước grid xuống còn (18,22), mô tả tại hình 2.2.

5. So sánh 2 grid này, lưu toàn bộ giá trị so sánh được khi slide window trượt đến toàn bộ ảnh, sau đó lấy giá trị nhỏ nhất so sánh với ngưỡng (**80**), nếu như giá trị miss nhỏ nhất nhỏ hơn giá trị ngưỡng thì thực hiện bounding box tại vị trí window trên ảnh test, sau đó thực hiện khôi phục kích thước ảnh về kích thước gốc, mô tả tại hình 2.3.

6. Bước tiếp theo nhóm tiến hành giảm kích thước ảnh test theo chiều kim tự tháp bằng cách sử dụng **pyrDown()** trong thư viện opencv. Sau đó lặp lại quá trình từ bước 2 đến bước 5 cho đến khi ảnh test có kích thước chiều rộng hoặc chiều cao nhỏ hơn 1.



Hình 2.1: Trích xuất hình ảnh từ window và chuyển ảnh sang ảnh nhị phân



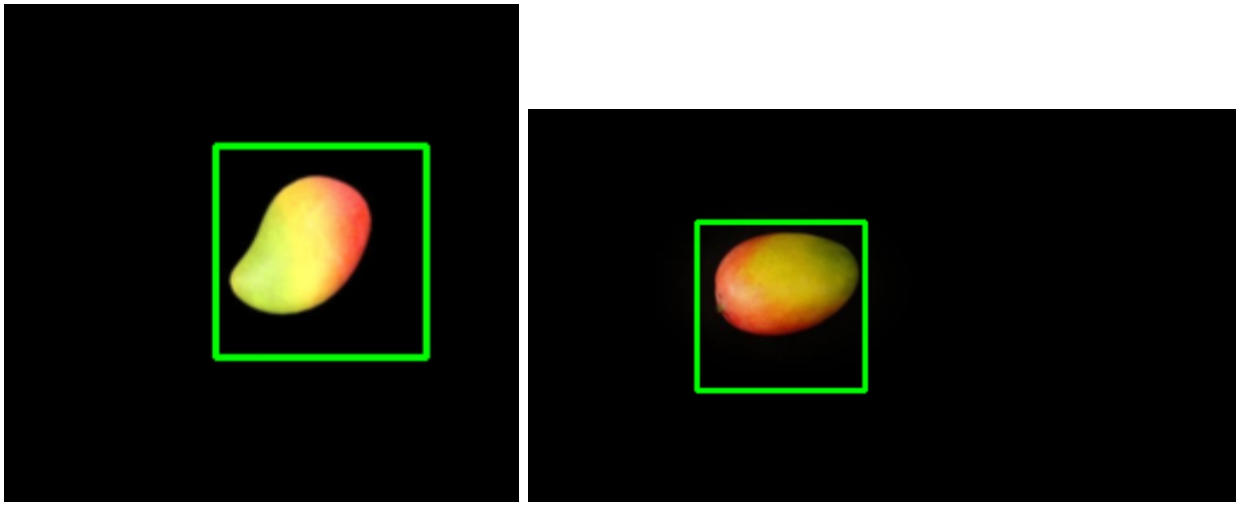
Hình 2.2: Chuyển patter_window_binary và test_window_binary sang dạng grid

7. Trong trường hợp không có giá trị miss nào nhỏ hơn giá trị ngưỡng, thì điều đó đồng nghĩa với việc không có quả xoài nào trong ảnh, kết quả xuất ra một thông báo ***There are no mangoes in the photo***, mô tả tại hình 3.2.

3 Kết quả

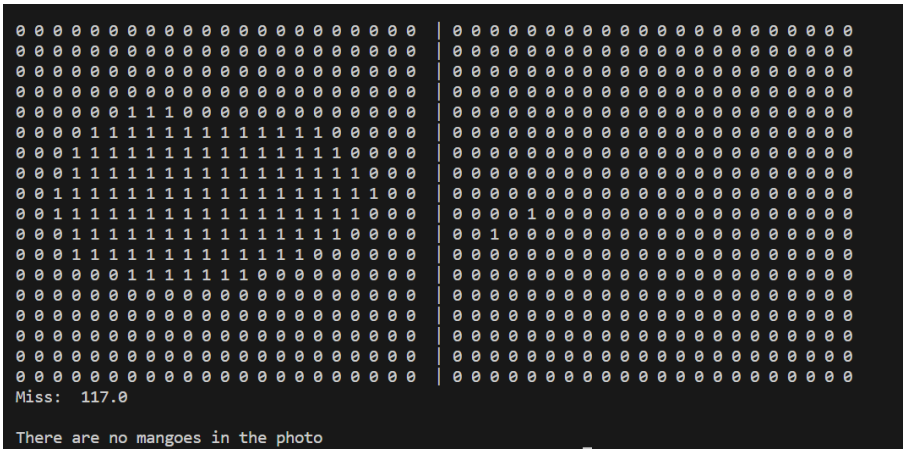
Kết quả được trình bày tại hình 3.1 và 3.2.

Hình 2.3: Thực hiện lấy giá trị miss trên toàn bộ ảnh được cắt ra từ slide window trên ảnh test



(a) Nhận diện quả xoài đứng (b) Nhận diện quả xoài ngang

Hình 3.1: Nhận diện quả xoài trong ảnh



Hình 3.2: Đối với trái ớt thì chương trình thông báo không tồn tại quả xoài nào trong hình ảnh

4 Code

```
import cv2
import numpy as np
from PIL import Image

def Image_to_grid(window, test, miss=0):
    # Open image, convert to black and white mode
    image = Image.fromarray(window).convert('1')
    w, h = image.size

    imagetest = Image.fromarray(test).convert('1')

    # Temporary NumPy array of type bool to work on
```

```
13      temp = np.array(image, dtype=bool)
14      temptest = np.array(imagetest, dtype=bool)
15
16      # Detect changes between neighboring pixels
17      diff_y = np.abs(np.diff(temp, axis=0).astype(bool))  #
           Convert diff_y to bool
18      diff_x = np.abs(np.diff(temp, axis=1).astype(bool))  #
           Convert diff_x to bool
19
20      diff_ytest = np.abs(np.diff(temptest, axis=0).astype(bool))
21      diff_xtest = np.abs(np.diff(temptest, axis=1).astype(bool))
22
23      # Create a union image of detected changes
24      temp = np.zeros_like(temp, dtype=bool)
25      temp[:h-1, :] |= diff_y
26      temp[:, :w-1] |= diff_x
27
28      temptest = np.zeros_like(temptest, dtype=bool)
29      temptest[:h-1, :] |= diff_ytest
30      temptest[:, :w-1] |= diff_xtest
31
32      # Calculate distances between detected changes
33      diff_y = np.diff(np.nonzero(np.diff(np.sum(temp, axis=0))))
34      diff_x = np.diff(np.nonzero(np.diff(np.sum(temp, axis=1))))
35
36      diff_ytest = np.diff(np.nonzero(np.diff(np.sum(temptest, axis
           =0))))
37      diff_xtest = np.diff(np.nonzero(np.diff(np.sum(temptest, axis
           =1))))
38
39      # Calculate tile height and width
40      ht = np.median(diff_y[diff_y > 1]) + 2
41      wt = np.median(diff_x[diff_x > 1]) + 2
42
43      httest = np.median(diff_ytest[diff_ytest > 1]) + 2
44      wttest = np.median(diff_xtest[diff_xtest > 1]) + 2
45
46      # Resize image w.r.t. tile height and width
47      if (ht > 0 and wt > 0) and (httest > 0 and wttest > 0):  #
           Ensure that ht/httest and wt/wttest are greater than 0
48          array = np.array(image.resize((int(w/wt), int(h/ht)))).
               astype(int)
49          arraytest = np.array(imagetest.resize((int(w/wt), int(h/
               ht)))).astype(int)
50          for y in range(0, int(h/ht)):
51              for x in range(0, int(w/wt)):
```

```
52                  print(array[y][x], end=" ")
53
54                  if array[y][x] != arraytest[y][x]:
55                      miss += 1
56              print(" | ",end="")
57              for x in range(0, int(w/wt)):
58                  print(arraytest[y][x], end=" ")
59              print("")
60          return miss
61      else:
62          print("Invalid tile height or width.")
63          return None
64
65
66  def convert_to_binary(img_grayscale, thresh=100):
67      thresh, img_binary = cv2.threshold(img_grayscale, thresh,
           maxval=255, type=cv2.THRESH_BINARY)
68      return img_binary
69
70  def sliding_window(image, step_size, window_size):
71      # get the window and image sizes
72      h, w = window_size
73      image_h, image_w = image.shape[:2]
74      # loop over the image, taking steps of size `step_size`
75      for y in range(0, image_h, step_size):
76          for x in range(0, image_w, step_size):
77              # define the window
78              window = image[y:y + h, x:x + w]
79              # if the window is below the minimum window size,
                   ignore it
80              if window.shape[:2] != window_size:
81                  continue
82              # yield the current window
83              yield (x, y, window)
84
85
86  path_image_test = "mangotest.jpg"
87  path_image_patter = "mangowithblackground.jpg"
88
89  image_patter = cv2.imread(path_image_patter, cv2.IMREAD_GRAYSCALE
       )
90  window_patter = cv2.resize(image_patter,(90,90))
91  window_patter = convert_to_binary(window_patter)
92
93  image_test = cv2.imread(path_image_test)
94  h_o, w_o, _ = image_test.shape # to resize clone to origin size
```

```
image_test_gray = cv2.imread(path_image_test, cv2.
    IMREAD_GRAYSCALE)
image_test_binnary = convert_to_binary(image_test_gray)
# Size of window
w, h = 90,90
# miss point
miss = np.zeros(1000)
countmiss = 0
isObject = False

while True:

    for (x, y, window) in sliding_window(image_test, step_size
       =30, window_size=(w, h)):

        test_window = image_test_binnary[y:y+h, x:x+w]
        miss[countmiss] = Image_to_grid(window_patter,
           test_window)

        print("Miss: ", miss[countmiss])

        clone = image_test.copy()
        cv2.rectangle(clone, (x, y), (x + w, y + h), (0, 255, 0),
            2)

        if miss[countmiss] is not None and miss[countmiss] < 80:
            none_zero = miss[miss!=0]
            if len(none_zero) > 0 and miss[countmiss] <= np.min(
               none_zero):
                clone = cv2.resize(clone, (w_o, h_o))
                cv2.imwrite('bounding_box_%d_%s' %(miss[countmiss
                   ], path_image_test) , clone)
                isObject = True

        cv2.imshow("Window", clone)
        cv2.imshow("Window_test", test_window)
        cv2.imshow("Window_patter", window_patter)

        cv2.waitKey(50)
        print("")
    if image_test.shape[0] <= 1 or image_test.shape[1] <= 1:
        break
    image_test = cv2.pyrDown(image_test)
    image_test_gray = cv2.cvtColor(image_test, cv2.COLOR_BGR2GRAY
       )
    image_test_binnary = convert_to_binary(image_test_gray)
```

```
134 if not isObject:
135     print("There are no mangoes in the photo")
```