

LỜI CẢM ƠN

Cảm ơn thầy Nguyễn Lý Thiên Trường đã hướng dẫn em hoàn thành đồ án này. Nhờ có sự chỉ dẫn nhiệt tình của thầy, em đã học được cách phân tích và giải quyết từng vấn đề trong quá trình thực hiện đồ án.

Thành phố Hồ Chí Minh, ngày 25 tháng 5 năm 2019

TÓM TẮT

Đồ án này trình bày về việc xây dựng một chương trình để phát hiện và phân loại trái Sori sử dụng các giải thuật máy học cơ bản và bộ thư viện mã nguồn mở OpenCV trên ngôn ngữ Python.

Đầu tiên chương trình sẽ sử dụng thuật toán Haar Cascade để xác định vùng chứa trái Sori, sau đó sử dụng thuật toán K-láng giềng gần nhất (K-NN) để dự đoán xem đó là trái Sori xanh hay chín.

Mục lục

1	GIỚI THIỆU	1
1.1	Tổng quan	1
1.2	Nhiệm vụ đề tài	1
2	LÝ THUYẾT	1
2.1	Thuật toán Haar Cascade [1]	1
2.1.1	Đặc trưng Haar-like	2
2.1.2	Tính Integral Image	3
2.1.3	Thuật toán Adaboost	4
2.1.4	Mô hình phân tầng Cascade	6
2.1.5	Hệ thống xác định vị trí vật thể	7
2.2	Hệ màu RGB	8
2.3	Thuật toán “K-Láng giềng gần nhất” (K-NN) [6]	8
2.4	Thư viện OpenCV	9
3	THIẾT KẾ VÀ THỰC HIỆN CHƯƠNG TRÌNH	9
3.1	Yêu cầu đặt ra	9
3.2	Chuẩn bị	9
3.3	Huấn luyện Haar Cascade [3, 4]	9
3.4	Tìm đặc trưng cho việc huấn luyện K-NN	11
3.5	Sơ đồ khối	11
3.5.1	Ảnh đầu vào	12
3.5.2	Dữ liệu Training K-NN	12
3.5.3	Tìm vùng chứa trái Sori	12
3.5.4	Trích đặc trưng	13
3.5.5	Phân loại	13
4	KẾT QUẢ THỰC HIỆN	14
4.1	Tiến hành thực nghiệm	14
4.2	Kết quả thu được sau thực nghiệm	14
4.3	Giải thích và phân tích kết quả	16
5	KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	16
5.1	Kết luận	16
5.2	Hướng phát triển	16
6	TÀI LIỆU THAM KHẢO	17
7	PHỤ LỤC	18

Danh sách hình vẽ

1.1	Trái Sori	1
2.1	4 đặc trưng cơ bản	2
2.2	Đặc trưng cạnh	2
2.3	Đặc trưng đường	2
2.4	Tính giá trị ảnh tích phân tại điểm có tọa độ (x, y)	3
2.5	Ví dụ tính tổng các giá trị mức xám trên vùng D	3
2.6	Khởi tạo trọng số cho các mẫu	4
2.7	Vòng lặp thứ 1	4
2.8	Vòng lặp thứ 2	5
2.9	Vòng lặp thứ 3	5
2.10	Kết hợp các bộ phân loại yếu lại	5
2.11	Mô hình phân tầng Cascade	6
2.12	Hệ thống xác định vị trí vật thể	7
2.13	Hệ màu RGB	8
2.14	Ví dụ K-NN	8
3.1	Tạo file vector chứa ảnh huấn luyện	10
3.2	Quá trình huấn luyện	10
3.3	Sơ đồ khối	11
3.4	$\text{minNeighbour} = 0$	13
3.5	$\text{minNeighbour} = 5$	13
4.1	Kết quả với một trái trên hình	15
4.2	Kết quả với nhiều trái trên hình	15
4.3	3 mặt của cùng một trái Sori bị phân loại sai	16

Danh sách bảng

3.1 Bảng giá trị trung bình các kênh màu trên trái Xanh và Chín 11

4.1 Kết quả nhận dạng với $K = 1$ 14

4.2 Kết quả nhận dạng với $K = 5$ 14

1 GIỚI THIỆU

1.1 Tổng quan

Máy học và thị giác máy tính là những chuyên ngành được phát triển khá lâu và có nhiều ứng dụng rộng rãi trong đời sống như: Tìm kiếm vật thể trong ảnh, nhận diện khuôn mặt, đọc biển số xe, nhận diện dấu vân tay,... Trong nông nghiệp: Phân loại rau củ quả, phát hiện bệnh trên các loại quả,...



Hình 1.1: Trái Sori

Trái Sori (hay Acerola) là loại quả mọng, vỏ nhẵn bóng và có vị ngọt, với hàm lượng Vitamin C gấp 20-40 lần cam, chanh. Sori còn chứa nhiều vi chất tốt cho sức khỏe như A, K, E, B1, B2, B3, B6, B12, sắt, magie, kali, canxi, kẽm, phospho, chất xơ, các axit amin và một số axit béo.... Trái có hình tròn, khi chín có màu đỏ. Hiện nay ở khu vực Gò Công - Tiền Giang, diện tích canh tác cây Sori khá lớn do phù hợp về thổ nhưỡng và khí hậu, góp phần không nhỏ vào việc ổn định kinh tế cho bà con nông dân. Tuy nhiên quá trình thu hoạch và các công đoạn sau thu hoạch còn tốn nhiều thời gian (do phải làm thủ công) dẫn đến hiệu quả kinh tế còn chưa cao.

1.2 Nhiệm vụ đề tài

Nhiệm vụ đặt ra ở đề án này là ứng dụng các thuật toán máy học đơn giản để xây dựng mô hình phần mềm phân loại trái Sori theo màu sắc (xanh và chín) với độ chính xác cao, thời gian trễ thấp, là bước đầu để xây dựng hoàn chỉnh hệ thống phân loại trái Sori.

2 LÝ THUYẾT

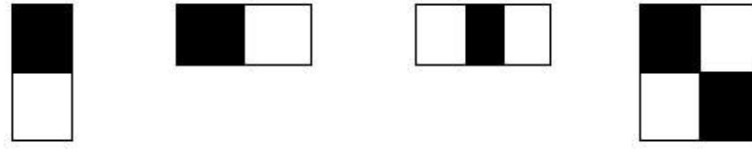
2.1 Thuật toán Haar Cascade [1]

Haar Cascade là một thuật toán trong máy học dùng để xác định các đối tượng trong hình ảnh hoặc video dựa trên các đặc trưng được đề xuất bởi Paul Viola và Michael Jones (2001). Thuật toán gồm 4 giai đoạn:

- Lựa chọn các đặc trưng Haar
- Tính Integral Images
- Huấn luyện Adaboost
- Thực hiện mô hình phân tầng Cascade

2.1.1 Đặc trưng Haar-like

Đặc trưng Haar-like được tạo thành bằng việc kết hợp các hình chữ nhật đen, trắng với nhau theo một trật tự, một kích thước nào đó. Có 4 đặc trưng Haar-like cơ bản:

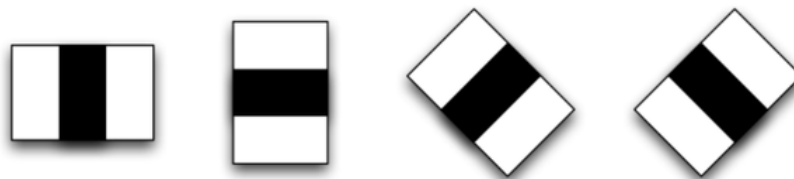


Hình 2.1: 4 đặc trưng cơ bản

Ngoài ra còn có các đặc trưng mở rộng:



Hình 2.2: Đặc trưng cạnh



Hình 2.3: Đặc trưng đường

Ứng với mỗi đặc trưng, một bộ phân lớp yếu $h_k(x)$ được định nghĩa như sau:

$$h_k(x) = \begin{cases} 1 & p_k f_k(x) < p_k \theta_k \\ 0 & \text{else} \end{cases} \quad (2.1a)$$

$$(2.1b)$$

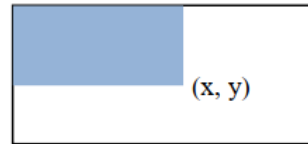
Trong đó:

- x : cửa sổ con cần xét.
- θ_k : ngưỡng.
- f_k : giá trị của đặc trưng Haar-Like.
- p_k : hệ số quyết định chiều của bất phương trình.

Ta hiểu công thức trên đơn giản như sau: Khi giá trị của đặc trưng Haar-Like k f_k tại cửa sổ con x vượt qua một ngưỡng θ_k thì một bộ phân lớp $h_k(x)$ sẽ kết luận cửa sổ con x là trái Sori, còn f_k không vượt qua ngưỡng đó thì không là trái Sori. Ngưỡng θ_k là giá trị rút ra sau quá trình huấn luyện bộ phân lớp, sẽ trình bày sau.

2.1.2 Tính Integral Image

Gía trị của đặc trưng Haar-like là sự chênh lệch giữa tổng các điểm ảnh của các vùng đen và các vùng trắng. Để tính nhanh các đặc trưng này, Viola và Jones đã giới thiệu khái niệm ảnh tích phân (Integral Image). Integral Image là một mảng hai chiều với kích thước bằng kích thước của ảnh cần tính đặc trưng Haar-like.

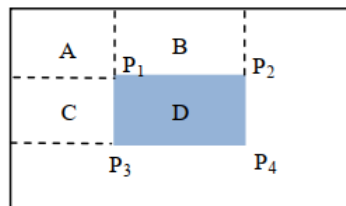


Hình 2.4: Tính giá trị ảnh tích phân tại điểm có tọa độ (x, y)

Gía trị ảnh tích phân tại điểm P có tọa độ (x, y) được tính như sau:

$$P(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \quad (2.2)$$

Sau khi đã tính được ảnh tích phân, việc tính tổng các giá trị mức xám của một vùng ảnh bất kỳ nào trên ảnh được thực hiện như sau.



Hình 2.5: Ví dụ tính tổng các giá trị mức xám trên vùng D

Giả sử ta cần tính tổng các giá trị mức xám của vùng D như trong hình 2.6 ta có thể tính như sau:

$$D = A + B + C + D - (A + B) - (A + C) + A \quad (2.3)$$

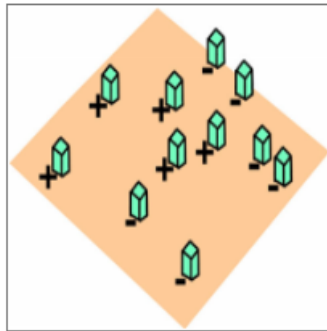
Hay

$$D = P4 + P1 - (P2 + P3) \quad (2.4)$$

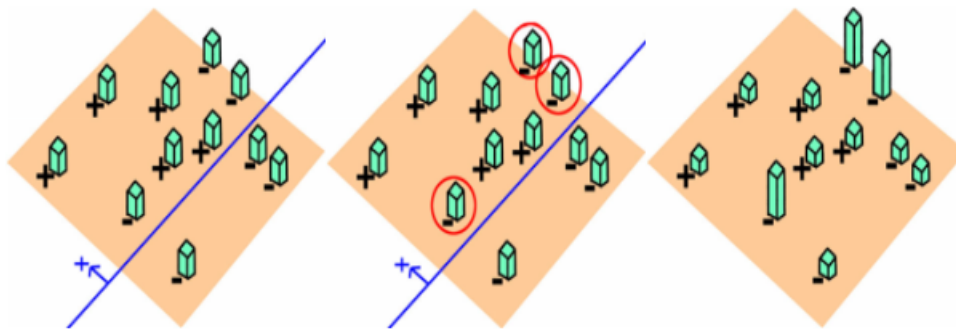
2.1.3 Thuật toán Adaboost

AdaBoost là một bộ phân loại mạnh phi tuyến phức, hoạt động dựa trên nguyên tắc kết hợp tuyến tính các bộ phân loại yếu để tạo nên một bộ phân loại mạnh. AdaBoost sử dụng trọng số để đánh dấu các mẫu khó nhận dạng. Trong quá trình huấn luyện cứ mỗi bộ phân loại yếu được xây dựng thì thuật toán sẽ tiến hành cập nhật lại trọng số để chuẩn bị cho việc xây dựng bộ phân loại tiếp theo. Cập nhật bằng cách tăng trọng số của các mẫu nhận dạng sai và giảm trọng số của các mẫu được nhận dạng đúng bởi bộ phân loại yếu vừa xây dựng. Bằng cách này thì bộ phân loại sau có thể tập trung vào các mẫu mà bộ phân loại trước nó làm chưa tốt. Cuối cùng các bộ phân loại yếu sẽ được kết hợp lại tùy theo mức độ tốt của chúng để tạo nên một bộ phân loại mạnh.

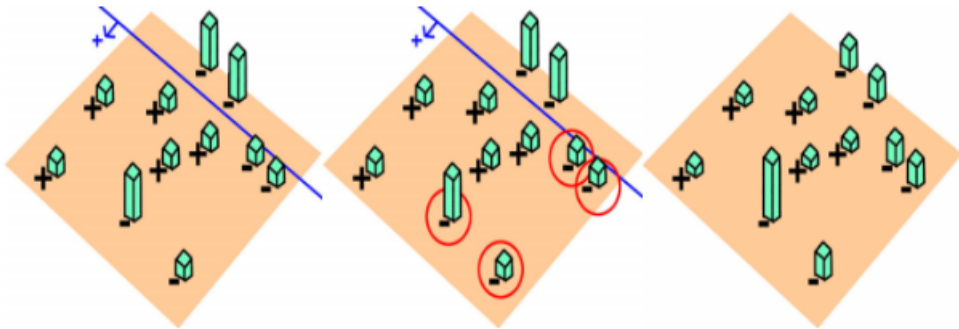
Để trực quan hơn, ta hãy quan sát qua chuỗi hình ảnh sau:



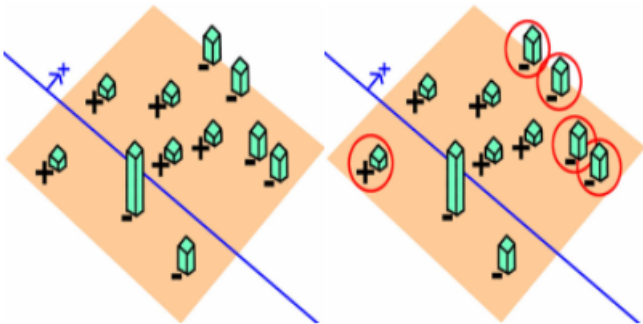
Hình 2.6: Khởi tạo trọng số cho các mẫu



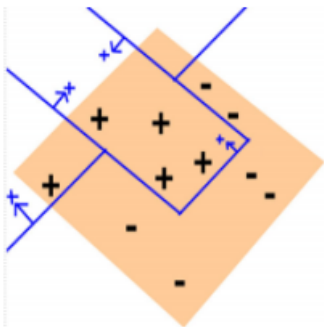
Hình 2.7: Vòng lặp thứ 1



Hình 2.8: Vòng lặp thứ 2

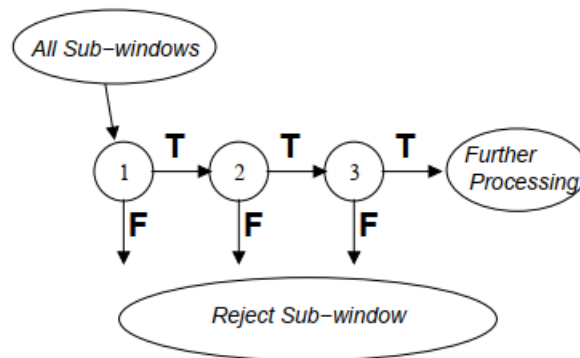


Hình 2.9: Vòng lặp thứ 3



Hình 2.10: Kết hợp các bộ phân loại yếu lại

2.1.4 Mô hình phân tầng Cascade



Hình 2.11: Mô hình phân tầng Cascade

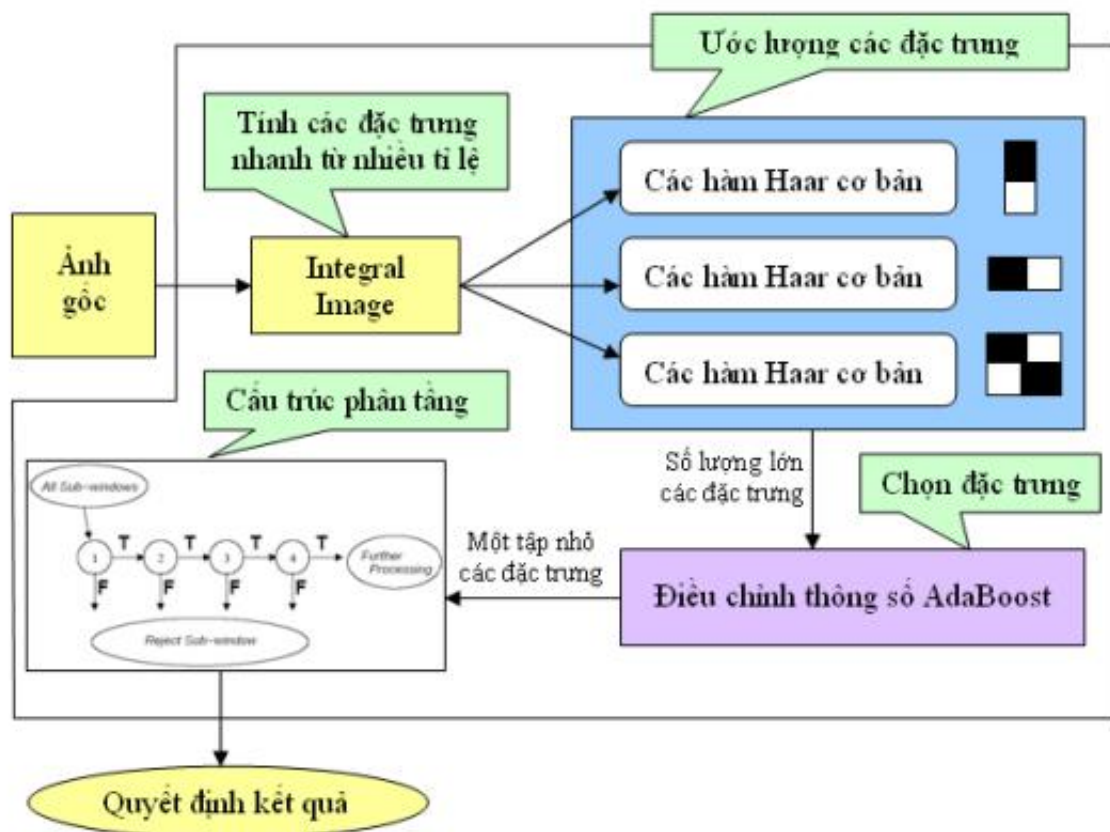
Mẫu đầu vào được đưa vào bộ phân loại đầu tiên, nếu kết quả là True (có vật thể) thì nó sẽ được đưa qua bộ phân loại thứ 2 và tiếp tục như vậy đến cuối cùng nếu vượt qua hết các bộ phân loại thì kết quả tổng hợp sẽ là True. Nếu kết quả là False (không có vật thể) ở bất kỳ bộ phân loại nào thì nó sẽ cho kết quả tổng hợp là False ngay lập tức.

Như đã đề cập ở phần 2.1.1, giá trị ngưỡng quyết định ở mỗi bộ phân loại là kết quả của quá trình huấn luyện được mô tả như sau:

- Giả sử ta có 2 tập ảnh dương (có vật thể) và ảnh âm (không có vật thể).
- Đầu tiên ta quy định số lượng bộ phân loại cần thực hiện.
- Ở mỗi bộ phân loại ta đưa vào một số lượng ảnh dương và số lượng đặc trưng Haar bất kỳ. Chương trình sẽ tính toán ra tỉ lệ nhận dạng thành công, tỉ lệ nhận dạng sai và các đặc trưng thu được. Sau đó chương trình sẽ tăng số lượng đặc trưng Haar rồi cập nhật lại các giá trị tỉ lệ. Liên tục như vậy cho đến khi tỉ lệ nhận dạng đúng và sai bằng nhau (đây chính là giá trị ngưỡng) thì chuyển sang bộ phân loại kế tiếp.

2.1.5 Hệ thống xác định vị trí vật thể

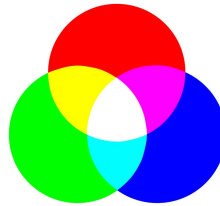
Trong hình 2.12, từ ảnh gốc ban đầu ta sẽ tính được Integral Image, mục đích là để tính nhanh tổng của các giá trị mức xám của một vùng hình chữ nhật bất kỳ trên ảnh gốc. Các vùng ảnh con này sẽ được đưa qua các hàm Haar cơ bản để ước lượng đặc trưng, kết quả ước lượng sẽ được đưa qua bộ điều chỉnh AdaBoost để loại bỏ nhanh các đặc trưng không có khả năng là đặc trưng của quả Sori. Chỉ có một tập nhỏ các đặc trưng mà bộ điều chỉnh AdaBoost cho là có khả năng là đặc trưng của quả Sori mới được chuyển sang cho bộ quyết định kết quả. Bộ quyết định sẽ tổng hợp kết quả là quả Sori nếu kết quả của các bộ phân loại yếu trả về là quả Sori.



Hình 2.12: Hệ thống xác định vị trí vật thể

2.2 Hệ màu RGB

Hệ màu RGB là viết tắt của 3 màu cơ bản là Red, Green và Blue là ba màu chính của ánh sáng trắng sau khi được tách ra nhờ lăng kính. Những màu này khi kết hợp theo tỉ lệ nhất định sẽ tạo ra rất nhiều màu khác nhau trong dải ánh sáng nhìn thấy, và khi kết hợp cả 3 màu lại với nhau với tỉ lệ 1 : 1 : 1 chúng ta sẽ được màu trắng. Bởi thế hầu hết các thiết bị điện tử sử dụng màu bằng cách phát quang như TV, màn hình máy tính, màn hình điện thoại... đều sử dụng RGB làm hệ màu chính. Và đó cũng là lý do mà các ảnh kỹ thuật số hiển thị trên máy tính đều sử dụng hệ RGB làm chuẩn.



Hình 2.13: Hệ màu RGB

2.3 Thuật toán “K-Láng giềng gần nhất” (K-NN) [6]

K-Nearest Neighbours (K-NN) là một trong những thuật toán máy học đơn giản nhất. K-NN là phương pháp để phân lớp các đối tượng (Query Point) dựa vào khoảng cách gần nhất giữa các đối tượng cần xếp lớp và tất cả các đối tượng trong Training Data.

Một đối tượng được phân lớp dựa vào K láng giềng của nó. K là số nguyên dương được xác định trước khi thực hiện thuật toán. Người ta thường dùng khoảng cách Euclidean để tính khoảng cách giữa các đối tượng.

Thuật toán K-NN được mô tả như sau:

1. Xác định giá trị tham số K.
2. Tính khoảng cách giữa các Query Point với tất cả các đối tượng trong Training Data.
3. Sắp xếp khoảng cách theo thứ tự tăng dần và xác định K láng giềng gần nhất với Query Point.
4. Lấy tất cả các lớp của K láng giềng gần nhất đã xác định.
5. Dựa vào phần lớn lớp của láng giềng gần nhất để xác định lớp cho Query Point.

Để hiểu K-NN được dùng để phân lớp thế nào ta xem minh họa dưới đây.



Hình 2.14: Ví dụ K-NN

Trong hình 2.13, Training Data là các hình ngôi sao và mặt trăng, hình tròn ở đây là đối tượng cần để xác định lớp (Query Point). Nhiệm vụ của chúng ta là dự đoán lớp của Query Point dựa vào việc lựa chọn số láng giềng gần nhất với nó.

Ta thấy rằng, với:

- 1-Nearest Neighbour: Kết quả sẽ là mặt trăng.
- 4-Nearest Neighbour: Kết quả sẽ không xác định được lớp cho Query Point vì số láng giềng gần nhất với nó là 4 trong đó có 2 cặp đối tượng thuộc 2 lớp khác nhau.
- 6-Nearest Neighbour: Kết quả sẽ là ngôi sao vì trong 6 láng giềng gần nhất với nó thì có 4 đối tượng thuộc lớp ngôi sao.

2.4 Thư viện OpenCV

OpenCV (Open Computer Vision) là một thư viện mã nguồn mở chuyên dùng để xử lý các vấn đề liên quan đến thị giác máy tính. Nhờ một hệ thống các giải thuật chuyên biệt, tối ưu cho việc xử lý thị giác máy tính, vì vậy tính ứng dụng của OpenCV là rất lớn, có thể kể đến như:

- Nhận dạng ảnh: nhận dạng khuôn mặt, các vật thể, ...
- Xử lý ảnh: khử nhiễu, điều chỉnh độ sáng, ...
- Nhận dạng cử chỉ.
- Và còn nhiều ứng dụng khác nữa.

Ta có thể sử dụng nhiều ngôn ngữ lập trình khác nhau để làm việc với OpenCV như C++, Java, Python, CSharp ...

Ở đồ án này sử dụng OpenCV 4.1.0 trên ngôn ngữ Python 3.7

3 THIẾT KẾ VÀ THỰC HIỆN CHƯƠNG TRÌNH

3.1 Yêu cầu đặt ra

- Nhận dạng và phân loại trái Sori với độ chính xác cao (trên 90 phần trăm).
- Thời gian nhận dạng và phân loại một trái dưới 0.5 giây.

3.2 Chuẩn bị

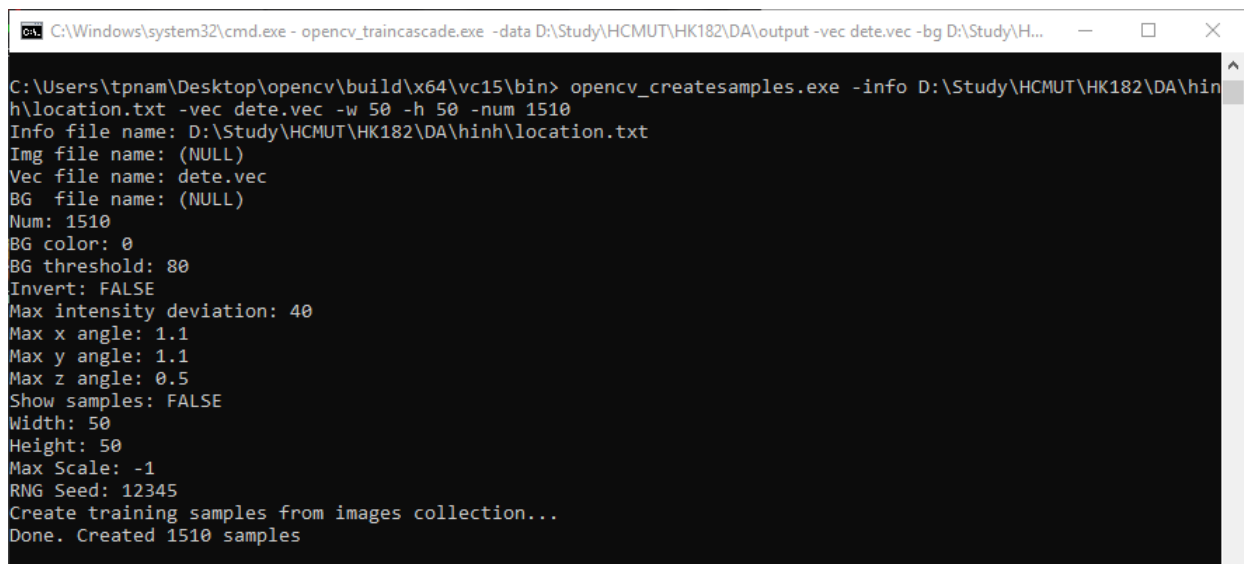
- Trước tiên cần tiến hành thu thập ảnh để làm Training Data.
- Để dễ dàng phát hiện được trái Sori, ở đây trái Sori được đặt trên nền đen.

3.3 Huấn luyện Haar Cascade [3, 4]

Tiến hành huấn luyện với 1510 ảnh dương (chứa trái Sori) và 3000 ảnh âm (không chứa trái Sori).

OpenCV cung cấp hai hàm để tiến hành huấn luyện:

- Sử dụng **opencv-createsamples.exe** để tạo ra file vector chứa các mẫu huấn luyện



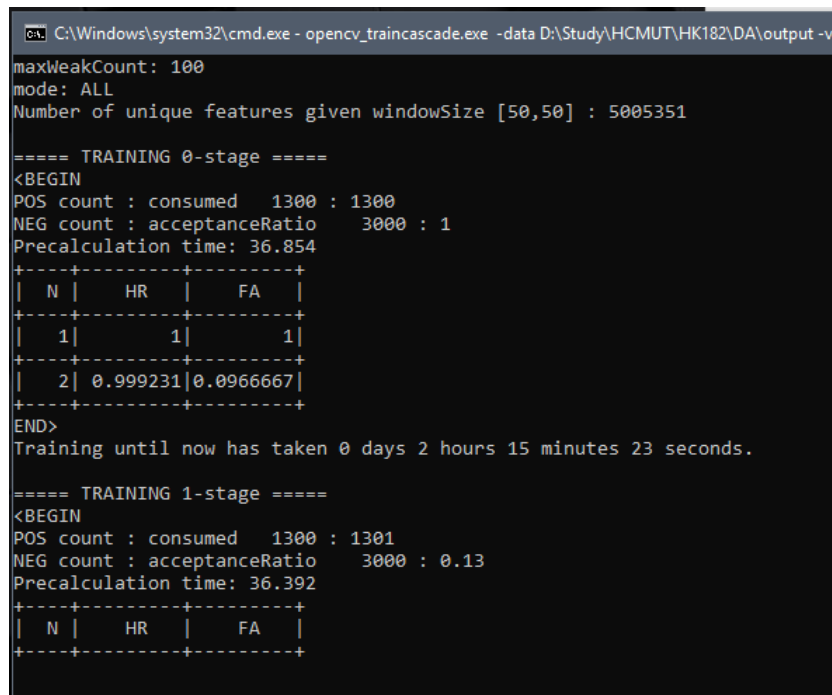
```

C:\Windows\system32\cmd.exe - opencv_traincascade.exe -data D:\Study\HCMUT\HK182\DA\output -vec dete.vec -bg D:\Study\H...
C:\Users\tpnam\Desktop\opencv\build\x64\vc15\bin> opencv_createsamples.exe -info D:\Study\HCMUT\HK182\DA\hin
h\location.txt -vec dete.vec -w 50 -h 50 -num 1510
Info file name: D:\Study\HCMUT\HK182\DA\hinh\location.txt
Img file name: (NULL)
Vec file name: dete.vec
BG file name: (NULL)
Num: 1510
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 50
Height: 50
Max Scale: -1
RNG Seed: 12345
Create training samples from images collection...
Done. Created 1510 samples

```

Hình 3.1: Tạo file vector chứa ảnh huấn luyện

- Sử dụng **opencv-traincascade.exe** lấy ảnh trong file vector đặt ngẫu nhiên vào các ảnh âm để rút trích đặc trưng.



```

C:\Windows\system32\cmd.exe - opencv_traincascade.exe -data D:\Study\HCMUT\HK182\DA\output -v
maxWeakCount: 100
mode: ALL
Number of unique features given windowSize [50,50] : 5005351

===== TRAINING 0-stage =====
<BEGIN
POS count : consumed 1300 : 1300
NEG count : acceptanceRatio 3000 : 1
Precalculation time: 36.854
+-----+
| N | HR | FA |
+-----+
| 1 | 1 | 1 |
+-----+
| 2 | 0.999231 | 0.0966667 |
+-----+
END>
Training until now has taken 0 days 2 hours 15 minutes 23 seconds.

===== TRAINING 1-stage =====
<BEGIN
POS count : consumed 1300 : 1301
NEG count : acceptanceRatio 3000 : 0.13
Precalculation time: 36.392
+-----+
| N | HR | FA |
+-----+

```

Hình 3.2: Quá trình huấn luyện

- Sau khi huấn luyện ta sẽ được file **cascade.xml** chứa các ngưỡng và đặc trưng trong các bộ phân loại yếu. Sử dụng file xml này để sử dụng cho chương trình chính.

3.4 Tìm đặc trưng cho việc huấn luyện K-NN

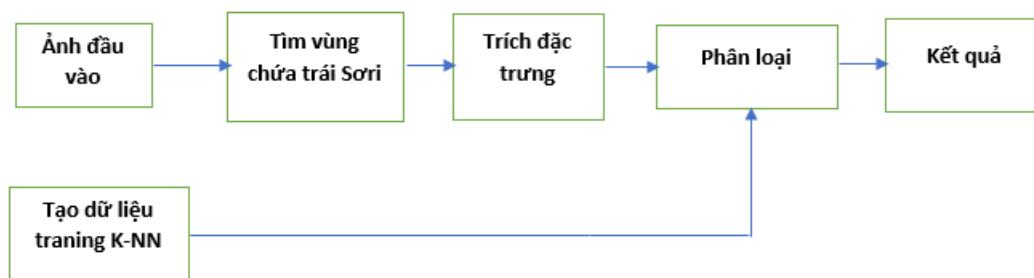
Như đã nói ở phần lý thuyết, mỗi một điểm ảnh trên một bức ảnh được biểu diễn bởi 3 giá trị kênh màu RGB. Tiến hành lấy trung bình cộng giá trị 3 kênh màu trên 100 ảnh trái chín và 100 trái xanh ta thu được kết quả sau:

STT	Trái xanh			Trái chín		
	B	G	R	B	G	R
1	8.2	47.4	31.0	37.3	46.6	98.4
2	22.1	108.3	103.7	41.5	44.7	80.2
3	19.6	90.0	76.2	36.4	45.5	79.5
4	22.9	71.3	71.4	42.5	50.3	84.3
5	21.3	91.5	101.9	45.4	68.2	120.9
6	28.5	74.6	75.4	34.5	36.9	75.8
7	24.1	96.6	100.4	42.4	61.7	89.3
8	24.2	69.5	67.0	62.5	81.5	113.2
9	14.7	68.6	74.1	34.1	40.9	78.0
10	22.9	71.1	63.3	39.4	40.9	89.1
...
Trung bình	41.4	85	81.3	40.0	57.4	109.9

Bảng 3.1: Bảng giá trị trung bình các kênh màu trên trái Xanh và Chín

Dựa vào bảng trên ta có thể thấy tỷ lệ giữa kênh màu G và R giữa trái xanh và trái chín có sự khác biệt khá rõ rệt nên ta sẽ chọn làm đặc trưng.

3.5 Sơ đồ khối



Hình 3.3: Sơ đồ khối

3.5.1 Ảnh đầu vào

Chụp từ Webcam (Logitech C270) độ phân giải 640x480 pixels, trái Sori đặt trên nền tối màu, không để các nguồn sáng ảnh hưởng đến hình ảnh thu được.

3.5.2 Dữ liệu Training K-NN

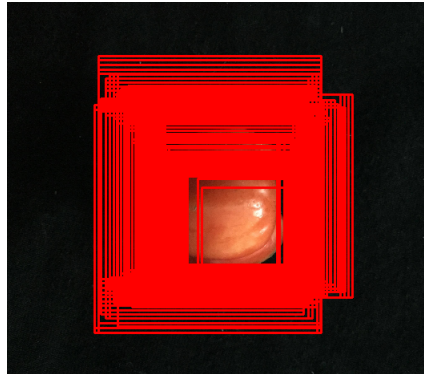
Được lấy từ 900 trái chín và 900 trái xanh khác nhau. Tiến hành lấy trung bình giá trị kênh màu G và R của từng ảnh, sau đó ghép lại thành một vector đặc trưng lớn. Sử dụng hàm `cv2.ml.KNearest-create()` do OpenCV cung cấp để tạo ra bộ phân loại. Do thuật toán Haar và Adaboost chỉ làm việc trên ảnh xám nên ta dùng hàm `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)` để chuyển ảnh này sang ảnh xám.

3.5.3 Tìm vùng chứa trái Sori

Từ dữ liệu đầu vào, sử dụng file `cascade.xml` có được sau huấn luyện Haar Cascade để tìm vùng chứa trái Sori. Để phát hiện trái Sori, hệ thống sẽ cho một cửa sổ con(sub-window) có kích thước cố định quét lên toàn bộ ảnh đầu vào. Như vậy sẽ có rất nhiều ảnh con ứng với từng cửa sổ con, các đặc trưng Haar-like sẽ được đặc lên các cửa sổ con này để từ đó tính ra giá trị của đặc trưng. Sau đó các giá trị này được bộ phân loại xác nhận xem khung hình đó có phải trái Sori hay không. OpenCV cung cấp sẵn một số hàm hỗ trợ cho việc này:

- `cv2.CascadeClassifier("file.xml")` : Hàm này khai báo file xml
- `detectMultiScale(image, ScaleFactor, minNeighbours)`: Hàm này sẽ phát hiện vùng chứa trái Sori có trong ảnh, trả về các giá trị tọa độ và độ rộng, độ cao của vùng, nhờ đó ta có thể vẽ được khung chứa trái Sori.
 - **image**: ảnh đầu vào.
 - **ScaleFactor**: Tỷ lệ tăng kích thước của khung cửa sổ tìm kiếm. Ví dụ nếu `ScaleFactor=1.1` thì sau khi quét hết bức ảnh 1 lần, khung cửa sổ sẽ tăng kích thước lên 10 phần trăm và thực hiện lần quét tiếp theo. Tham số này ảnh hưởng đến tốc độ xử lý và độ tin cậy của chương trình. Nếu để nó quá lớn thì tốc độ chương trình sẽ tăng lên do số lần quét giảm đi, tuy nhiên có thể chương trình có thể bỏ qua không phát hiện được một số trái có kích thước nằm giữa 2 khung cửa sổ liên tiếp do độ tăng kích thước của khung là quá lớn. Nếu để nó quá thấp thì ta có thể không bỏ sót bất kì trái nào nhưng chương trình sẽ tốn thời gian hơn vì tăng số lần quét lên.

- **minNeighbour**: giá trị tối thiểu số hình chữ nhật lân cận được gộp lại sau khi quá trình quét đã xong. Trong quá trình tìm kiếm trái Sori chương trình sẽ tìm được nhiều những khung hình chữ nhật chứa trái Sori cho dù đó chỉ là một trái và có nhưng trường hợp nhận diện nhầm.
Nếu ta để $\text{minNeighbour} = 0$ kết quả sẽ cho ra:

Hình 3.4: $\text{minNeighbour} = 0$

Còn khi cho $\text{minNeighbour} = 5$ kết quả sẽ là:

Hình 3.5: $\text{minNeighbour} = 5$

3.5.4 Trích đặc trưng

Sau khi xác định được khu vực nào chứa trái Sori, tiến hành tính trung bình giá trị kênh màu G và R của vùng đó để làm vector đặc trưng.

3.5.5 Phân loại

Sau khi đã có được đặc trưng của trái Sori cần phân loại, sử dụng hàm **findNearest**(**đặc trưng của trái cần phân loại, số láng giềng gần nhất**) do OpenCV cung cấp để dự đoán kết quả.

4 KẾT QUẢ THỰC HIỆN

4.1 Tiến hành thực nghiệm

Tiến hành thực nghiệm trên 350 mẫu xanh và 350 mẫu chín (mỗi ảnh chứa một trái, không bao gồm các ảnh đã huấn luyện). Sori được đặt trên nền đen, chụp bằng webcam (640x480 pixels). K láng giềng tăng dần từ 1 đến 20, scaleFactor là 1.2, minNeighbour là 10.

4.2 Kết quả thu được sau thực nghiệm

Kết quả với K-láng giềng bằng 1.

Trái xanh			Trái chín		
Detected	Phân loại đúng	Phân loại sai	Detected	Phân loại đúng	Phân loại sai
300	334	16	300	341	9

Bảng 4.1: Kết quả nhận dạng với $K = 1$

Tỷ lệ nhận dạng chính xác: **92.9** phần trăm.

Thời gian nhận dạng và phân loại một trái: 0.0346 giây.

Kết quả với K-láng giềng bằng 5

Trái xanh			Trái chín		
Detected	Phân loại đúng	Phân loại sai	Detected	Phân loại đúng	Phân loại sai
300	338	12	300	342	8

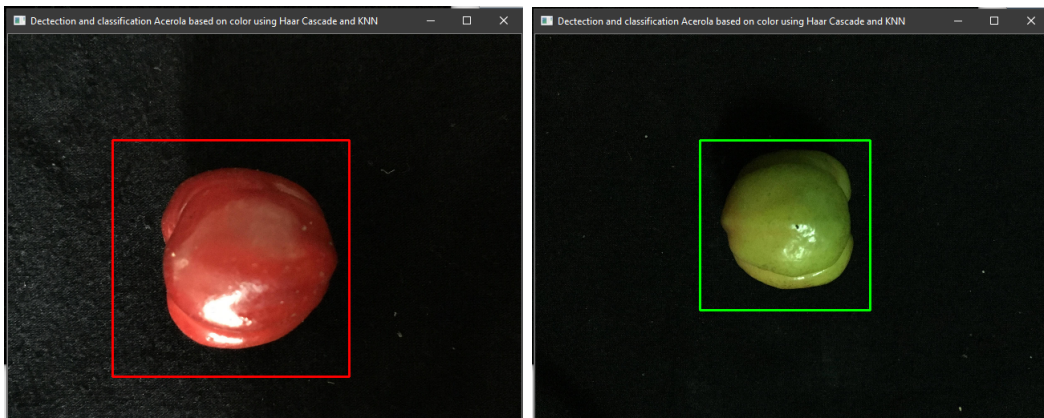
Bảng 4.2: Kết quả nhận dạng với $K = 5$

Tỷ lệ nhận dạng chính xác: **94.4** phần trăm.

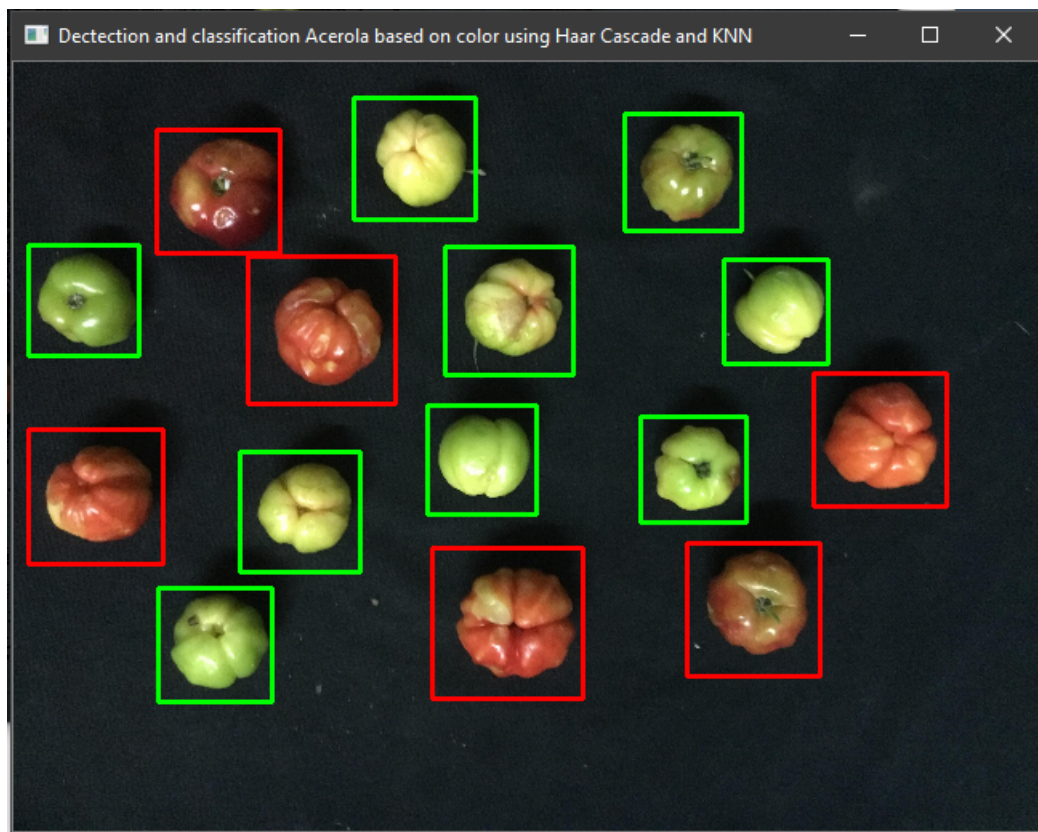
Thời gian nhận dạng và phân loại một trái: 0.055 giây.

Tăng thêm K thì Tỷ lệ chính xác vẫn không tăng thêm, thời gian nhận dạng và phân loại tăng.

Một số hình ảnh thu được:



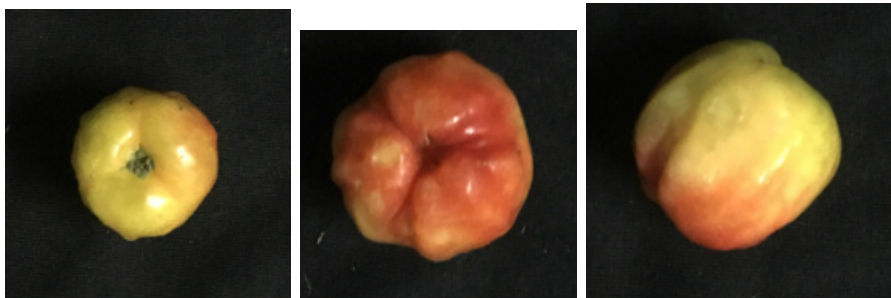
Hình 4.1: Kết quả với một trái trên hình



Hình 4.2: Kết quả với nhiều trái trên hình

4.3 Giải thích và phân tích kết quả

- Kết quả thu được cho thấy chương trình thực hiện khá tốt. Tỷ lệ chính xác còn tùy thuộc vào số K láng giềng tuy nhiên không ảnh hưởng nhiều do số lượng dữ liệu tham chiếu khá lớn. Thời gian nhận dạng và phân loại khá nhanh do ảnh có độ phân giải thấp (640x480 pixels) và giá scaleFactor lớn (1.2)
- Một số trái phân loại chưa chính xác do các nguyên nhân sau:
 - Dữ liệu sử dụng làm mẫu huấn luyện chưa chuẩn. Do được chọn lọc thủ công nên trong quá trình phân loại không tránh khỏi sai sót, mang yếu tố chủ quan.
 - Một số trái không chín hẳn cũng không xanh hẳn, điều này dẫn đến kết quả sẽ phụ thuộc vào góc chụp.



Hình 4.3: 3 mặt của cùng một trái Sori bị phân loại sai

5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết luận

- Kết quả thu được đáp ứng được mục tiêu đề ra ban đầu (nhận dạng đúng trên 90 phần trăm) thời gian nhận dạng và phân loại nhanh (nhỏ hơn 0.5 giây).
- Để có được kết quả tốt nhất, cần phải chuẩn bị dữ liệu huấn luyện thật chuẩn, nhiều.
- Điều kiện chụp: tránh ảnh hưởng của các nguồn sáng đến màu sắc của trái Sori.

5.2 Hướng phát triển

- Phát triển thêm để có thể nhận diện được các trái hư (hỏng).
- Kết hợp với phần cứng để tạo thành hệ thống phân loại hoàn chỉnh.

6 TÀI LIỆU THAM KHẢO

- [1] Paul Viola and Michael Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, 2001.
- [2] Cascade classifier class for object detection, https://docs.opencv.org/3.4/d1/de5/classcv_1_1CascadeClassifier.html
- [3] Train a Cascade Object Detector, <https://www.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html>
- [4] Phát hiện đối tượng, <https://thigiacmaytinh.com/phan-hien-doi-tuong-p1-ly-thuyet/>
- [5] The OpenCV Tutorials Release 2.4.13.7
- [6] Vũ Hữu Tiệp, K-nearest neighbors, <https://machinelearningcoban.com/2017/01/08/knn/>

7 PHỤ LỤC

Toàn bộ chương trình:

```
import numpy as np
import cv2

pathR = 'D:/Study/HCMUT/HK182/DA/Images/Samples/2Color/Red/'
pathG = 'D:/Study/HCMUT/HK182/DA/Images/Samples/2Color/Green/'

size = 20
train_ids = np.arange(1, 901)

def buildfilename(path, pre, im_ids):
    filename = []
    for im_id in im_ids:
        fn = path + pre + '-' + str(im_id) + '.png'
        filename.append(fn)
    return filename

def resize(filename):
    img = cv2.imread(filename)
    img = cv2.resize(img, (size, size))
    return img

def trainsamples(img_ids):
    filenamegreen = buildfilename(pathG, 'G', img_ids)
    filenamered = buildfilename(pathR, 'R', img_ids)
    filenamefull = filenamegreen + filenamered
    resized = []
    feature = []
    for i in range(len(filenamefull)):
        x = resize(filenamefull[i])
        resized.append(x)
    for i in range(len(resized)):
        x = resized[i]
        f = []
        # blue = 0
        green = 0
        red = 0
        for j in range(len(x)):
            cell = x[j]
            for z in range(len(cell)):
                # blue = blue + cell[j][0]
                green = green + cell[j][1]
                red = red + cell[j][2]
            # f.append(int(blue/(size*size)))
            f.append(int(green / (size * size)))
            f.append(int(red / (size * size)))
        feature.append(f)
    feature = np.array(feature).astype(np.float32)
```

```

    return feature

x = trainsamples(train_ids)
k = np.arange(2)
lable = np.repeat(k, len(x) / 2)[: , np.newaxis]
knn = cv2.ml.KNearest_create()
knn.train(x, 0, lable)

def test(filename):
    f = []
    t = []
    img = cv2.imread(filename)
    img = cv2.resize(img, (size, size))
    # blue = 0
    green = 0
    red = 0
    for i in range(len(img)):
        cell = img[i]
        for j in range(len(cell)):
            # blue = blue + cell[j][0]
            green = green + cell[j][1]
            red = red + cell[j][2]
    # f.append(int(blue/(size*size)))
    f.append(int(green / (size * size)))
    f.append(int(red / (size * size)))
    t.append(f)
    sp = np.array(t).astype(np.float32)
    kq1, kq2, kq3, kq4 = knn.findNearest(sp, 20)
    return kq2

soridetect = cv2.CascadeClassifier("cascade1.xml")

cap = cv2.VideoCapture(1)

while True:
    ret, frame = cap.read()
    if ret != 1:
        print("Video finished")
        break
    frame = cv2.resize(frame, (640, 480))
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    sori = soridetect.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=10,
    )
    for (x, y, w, h) in sori:
        img_crop = frame[y + 2:y + h - 2, x + 2:x + w - 2, :]
        cv2.imwrite('object.jpg', img_crop)
        c = test('object.jpg')
        font = cv2.FONT_HERSHEY_SIMPLEX

```



```
if c == 0:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    # cv2.putText(frame, 'xanh', (x, y - 5), font, 0.5, (0, 255, 0), 1,
    #               cv2.LINE_AA)
else:
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
    # cv2.putText(frame, 'chín', (x, y - 5), font, 0.5, (0, 255, 0), 1,
    #               cv2.LINE_AA)
cv2.imshow('Dectection and classification Acerola based on color using Haar
Cascade and KNN', frame)
if cv2.waitKey(1) == 27:
    print('Bye bye !!!')
    break
cap.release()
cv2.destroyAllWindows()
```
