HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO BÀI TẬP LỚN

MÔN HỌC: HỆ CƠ SỞ DỮ LIỆU ĐA PHƯƠNG TIỆN

Đề tài: Xây dựng hệ cơ sở dữ liệu nhận dạng ảnh hoa

Nhóm lớp : **D19-026**

Nhóm bài tập lớn : **9**

Giảng viên : **TS.Nguyễn Đình Hóa**

Sinh viên : **Nguyễn Văn Khánh – B19DCCN357**

Phan Quang Huy – B19DCCN321

Nguyễn Dương Kỳ Anh – B19DCCN021

Hà Nội, Tháng 6 Năm 2023

Mục lục

I. Đặc điểm dữ liệu 1
II. Các phương pháp trích chọn đặc trưng phổ biến 6
1. Đặc trưng màu sắc 6
1.1. Khái niệm 6
1.2. Không gian màu 6
1.3. Chia bin trong histogram 9
1.4. Global Color Histogram and Local Color Histogram 10
1.5. Cách tính Color Histogram 11
1.6. Một số nhược điểm của Color Histogram 12
2. Đặc trưng hình dạng 13
2.1. Đặc trưng hình dạng là gì? 13
2.2. Gradient là gì? 13
2.3. Histograms of oriented gradients (HOG) 14
2.4. Các bước trích chọn đặc trưng hình dạng ảnh sử dụng HOG 14
III. Xây dựng hệ thống nhận diện ảnh hoa 21
1. Sơ đồ khối của hệ thống 21
2. Các thuộc tính được sử dụng và các kỹ thuật trích rút 23
2.1. Màu sắc 23
2.2. Hình dạng 25
3. Lưu trữ và nhận dạng 27
3.1. Cách lưu trữ các thuộc tính ảnh hoa 27
3.2. Cách nhận dạng ảnh hoa 28
4. Demo hệ thống 32
5. Đánh giá kết quả 35
IV. Phụ lục 36
1. Code trích rút đặc trưng màu sắc 36
2. Code trích rút đặc trưng hình dạng 37
3. Trích rút đặc trưng từ tập dữ liệu 41
4. Code giao diện 42

I. Đặc điểm dữ liệu

1. Hoa cúc châu phi

- Màu sắc: Cánh hoa có màu vàng, nhụy có màu cam
- Hình dạng: Cánh hoa thuôn dài, nhụy tròn

2. Hoa cúc huân chương

- Màu sắc: Cánh hoa màu vàng, chân cánh hoa có màu đen, nhụy có màu cam
- Hình dạng: Cánh hoa thuôn dài, nhụy tròn

3. Hoa đồng tiền

- Màu sắc: Cánh hoa màu vàng, nhụy hoa màu nâu
- Hình dạng: Cánh hoa thuôn dài, số lớp cánh hoa nhiều

4. Hoa hướng dương

- Màu sắc: Cánh hoa màu vàng, nhụy hoa ở vùng trung tâm có màu rêu, ở rìa thì có màu nâu
- Hình dáng: Cánh hoa thuôn dài, nhụy thường chiếm diện tích lớn

5. Hoa bướm

- Màu sắc: Cánh hoa màu vàng, nhụy hoa ở vùng trung tâm có màu vàng giống cánh hoa
- Hình dáng: Cánh hoa độ dài vừa, ở rìa thì có hình răng cưa, nhụy hoa tròn, nhiều

6. Hoa mắt huyền

- Màu sắc: Cánh hoa màu vàng, nhụy hoa màu đen.
- Hình dáng: Cánh hoa 5, rời, độ mỏng dẹt hình trái tim

7. Hoa đông hầu

- Màu sắc: Cánh hoa màu vàng nhạt, nhị màu cam nhạt
- Hình dáng: cánh hoa giống hình quạt, nhụy hoa ít, đầu nhụy thẳng dài

8. Hoa hồng đá

- Màu sắc: Cánh hoa, nhị hoa và nhụy hoa đề màu vàng
- Hình dạng: Gồm 5 cánh, đều, rời, hơi giống hình tam giác, nhị hoa không đều xếp thành hình vòng tròn, có xu hướng tõe ra, nhụy hoa nhỏ

9. Hoa anh thảo

- Màu sắc: Cánh hoa có màu vàng đặc, chân cánh hoa có màu cam nâu, nhụy hoa màu vàng
- Hình dạng: Cánh hoa hơi tròn, đầu cánh hoa có hình giống trái tim, số lượng nhụy hoa ít

10. Hoa dâm bụt

- Màu sắc: Phần trên cánh hoa có màu vàng nhạt, phần chân cánh hoa màu đỏ, nhị hoa màu vàng
- Hình dạng: Cánh hoa 5, đều, rời,nhụy nhiều, không đều, đính trên đế hoa thành 1 vòng

II. Các phương pháp trích chọn đặc trưng phổ biến

1. Đặc trưng màu sắc

1.1. Khái niệm

- Color Histogram là một biểu đồ của một hình ảnh thể hiện sự phân bố màu sắc trong hình ảnh. Nó hiển thị các loại màu khác nhau xuất hiện và số lượng (hoặc tần suất) của mỗi loại màu xuất hiện.
- Color histogram chỉ tập trung vào tỷ lệ số lượng các loài màu khác nhau, bất kể vị trí không gian của màu. Các giá trị của biểu đồ màu là từ số liệu thống kê. Chúng cho thấy sự phân bố thống kê của màu sắc và tông màu cơ bản của hình ảnh.

1.2. Không gian màu

1.2.1. Mô hình màu RGB

- Mô hình màu RGB là một mô hình màu dựa trên sự kết hợp của ba màu cơ bản: Đỏ (R), Xanh lá (G) và Xanh dương (B). Trong mô hình này, mỗi màu cơ bản được biểu diễn bằng một giá trị số từ 0 đến 255 (8 bit), và màu sắc của một điểm ảnh được xác định bằng cách kết hợp các giá trị R, G, B theo tỉ lệ nhất định. Với mô hình biểu diễn 24bit, số lượng màu tối đa sẽ là:

$$255 \times 255 \times 255 = 16581375 \text{ màu}$$

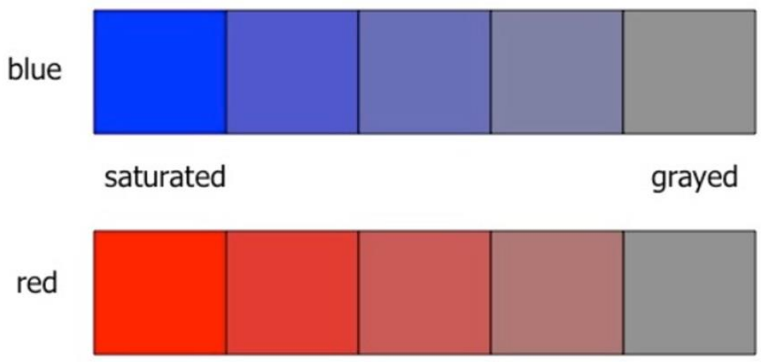
- Một điểm ảnh màu được biểu diễn bằng một vector ba chiều (R, G, B), trong đó R, G, B lần lượt là giá trị màu đỏ, xanh lá, và xanh dương của điểm ảnh đó. Mô hình màu RGB rất phù hợp để biểu diễn ảnh vì các thiết bị hiển thị ảnh như màn hình máy tính, TV, điện thoại di động... thường sử dụng mô hình màu này.
- Ưu điểm:
    - Không cần chuyển đổi để hiển thị thông tin trên màn hình, vì lý do này, nó được coi là không gian màu cơ bản cho các ứng dụng khác nhau
    - Được sử dụng trong hiển thị video nhờ tính chất cộng dồn
- Nhược điểm:
    - Không hữu ích trong việc đặc tả đối tượng và nhận dạng màu sắc
    - Phức tạp trong nhận diện màu cụ thể

1.2.2. Mô hình màu HSV

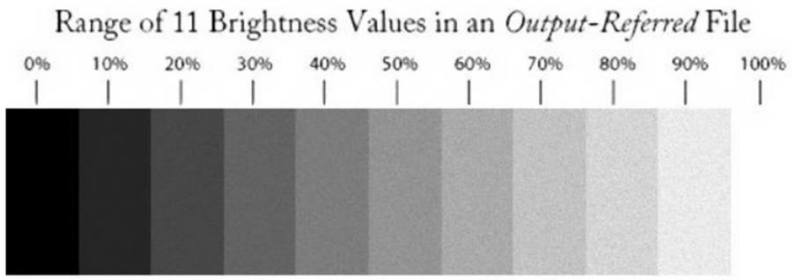
- Mô hình màu HSV là một mô hình màu dựa trên ba thông số: Hue (H), Saturation (S), và Value (V). Trong đó, Hue đại diện cho màu sắc, Saturation đại diện cho độ bão hòa màu sắc, và Value đại diện cho độ sáng của màu sắc.
- H (Hue) là thông số chỉ màu cơ bản, nó biểu diễn dưới dạng góc độ trong một vòng tròn màu, với giá trị từ 0 đến 360 độ . Ví dụ, màu đỏ có giá trị H khoảng từ 0 đến 20 hoặc từ 340 đến 360 độ.

- S (Saturation) là thông số độ bão hòa của màu, nó biểu diễn mức độ màu sắc của một điểm ảnh so với màu xám, là đại lượng đặc trưng cho sắc màu tương đối của vật thể so với màu gốc. Hiểu một cách dễ hiểu là cũng một màu tuy nhiên với cấp độ đậm nhạt không giống nhau hay nói khác đi là chúng ta

đang pha màu với màu xám.Giá trị S từ 0 đến 1, với S = 0 thì điểm ảnh tương ứng là màu xám, còn S = 1 thì màu sắc của điểm ảnh đó đậm nhất.



- V (Value) là thông số độ sáng của màu, nó biểu diễn mức độ sáng tối của một điểm ảnh. Giá trị V từ 0 đến 1, với V = 0 thì điểm ảnh tương ứng là đen nhất, còn V = 1 thì màu sắc của điểm ảnh đó sáng nhất.



- Mô hình màu HSV rất phù hợp để phân tích về mặt màu sắc của ảnh, bởi vì nó cho phép phân biệt các màu sắc một cách dễ dàng hơn so với mô hình màu RGB, và các thông số Saturation và Value cung cấp thông tin về độ bão hòa và độ sáng của màu sắc.

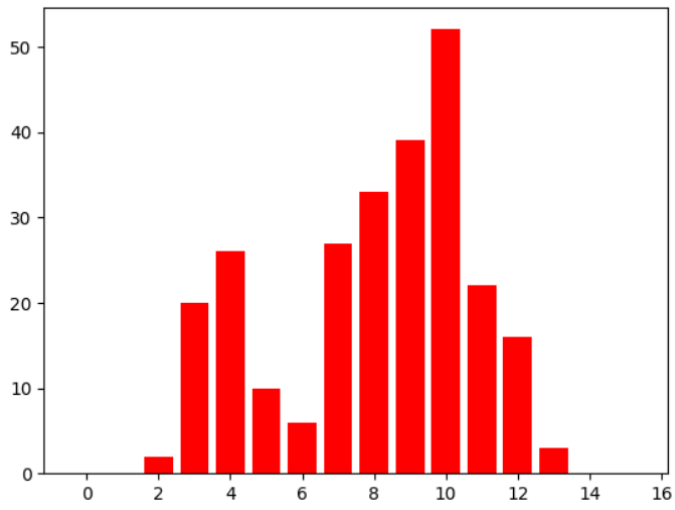
- So sánh giữa RGB và HSV trong trích xuất đặc trưng Color Histogram:
    - Mô hình màu HSV thường được sử dụng hơn trong trích xuất đặc trưng Color Histogram, bởi vì nó phản ánh màu sắc của ảnh một cách trực quan hơn so với mô hình màu RGB.

- Trong mô hình màu HSV, khoảng giá trị màu được chia ra theo đặc tính của màu sắc, trong khi đó, trong mô hình màu RGB, khoảng giá trị màu được chia ra theo các giá trị cơ bản của màu sắc, nên mô hình màu HSV thường cho kết quả tốt hơn trong việc phân biệt các màu sắc.

Tuy nhiên, việc sử dụng mô hình màu RGB trong trích xuất đặc trưng Color Histogram cũng có thể đưa ra kết quả tốt, đặc biệt là khi chúng ta cần một mô hình màu đơn giản và hiệu quả để tính toán.

1.3. Chia bin trong histogram

- Kĩ thuật chia bin trong không gian màu RGB và HSV là quá trình chia khoảng giá trị màu của các kênh màu thành các bin để đếm số lượng pixel trong ảnh có giá trị màu nằm trong từng bin đó. Kỹ thuật này là phương pháp phổ biến để trích xuất đặc trưng Color Histogram của một ảnh.
- Ví dụ: ta cần tìm số pixel nằm trong khoảng từ 0 đến 15, sau đó là 16 đến 31, ..., 240 đến 255. Ta sẽ chỉ cần 16 giá trị để biểu diễn biểu đồ. Vậy những gì cần làm chỉ đơn giản là chia toàn bộ biểu đồ thành 16 phần con và giá trị của mỗi phần con là tổng của tất cả số lượng pixel trong đó. Mỗi phần này được gọi là "BIN". Trong trường hợp đầu tiên, số lượng thùng là 256 trong khi ở trường hợp thứ hai, nó chỉ là 16.



- Khi chia bin trong không gian màu, việc lựa chọn số lượng bin và độ rộng bin phù hợp rất quan trọng để đảm bảo rằng histogram đủ phong phú để mô tả màu sắc của ảnh và đồng thời không quá phức tạp để tính toán. Thông thường, số lượng bin có thể được điều chỉnh để đạt được kết quả tốt nhất cho bài toán cụ thể. Nếu chọn số bin ít quá , thì biểu đồ sẽ có ít thành phần hơn và không thể phân biệt giữa các hình ảnh có sự phân bố màu về cơ bản khác nhau. Tương tự như vậy, nếu chọn số bin nhiều quá, biểu đồ sẽ có nhiều thành phần và hình ảnh có nội dung rất giống nhau có thể bị coi là "không giống" trong thực tế.

1.4. Global Color Histogram and Local Color Histogram

1.4.1. Global Color Histogram(GCL)

- Biểu đồ màu toàn cục (GCL) là biểu đồ màu của toàn bộ bức ảnh mà không có sự phân chia vùng , do đó nó sẽ biểu thị phân bố màu sắc trên toàn bộ bức ảnh. GCH được biết đến là phương pháp truy xuất hình ảnh dựa trên màu sắc truyền thống. GCH sẽ bất biến với phép xoay của ảnh vì khi đó phân bố màu sắc của bức ảnh không thay đổi
- Tuy nhiên nó không bao gồm phân bố màu của các vùng nên khi so sánh hai GCH, nên khi so sánh hai hình ảnh có cùng phân bố màu sắc nhưng khác nhau về hình dáng vật thể hoặc vị trí có thể không đem lại sự chính xác.
- Hai hình ảnh có thể so sánh bằng cách tính khoảng cách giữa 2 histogram, có thể sử dụng công thức Euclid để tính khoảng cách

$$d_{GCH}(Q, I) = \sqrt{\sum_{i=1}^{N}(H_Q[i] - H_I[i])^2}$$

trong đó :

- N là số lượng bin trong histogram
- Hq[i] là giá trị của bin i trong lược đồ màu Hq của ảnh Q
- $H_I[i]$ là giá trị của bin i trong lược đồ màu Hq của ảnh I

1.4.2. Local Color Histogram(LCH)

- Lược đồ màu cục bộ (LCH) là lược đồ màu của các vùng trong bức ảnh. Đầu tiên là phân vùng bức ảnh thành các khối và thu được lược đồ màu cho mỗi khối. Sau đó một ảnh sẽ được biểu diễn bởi các lược đồ này. Khi so sánh hai hai bức ảnh, ta tính khoảng cách lược đồ giữa 2 khối của 2 bức ảnh ở cùng một vị trí. Khoảng cách giữa 2 ảnh sẽ bằng tổng khoảng cách giữa các lược đồ. Nếu chúng ta sử dụng khoảng cách Euclid để tính khoảng cách giữa các lược đồ, khoảng cách giữa 2 ảnh Q và I có thể được xác định:

$$d_{LCH}(Q,I) = \sum_{k=1}^{M} \sqrt{\sum_{i=1}^{N} (H_Q^k[i] - H_I^k[i])^2}$$

  trong đó:

  - M là số các khối trong bức ảnh
  - H_Q^k[i] là giá trị của bin i trong lược đồ màu của khối k của ảnh Q
  - H_I^k[i] là giá trị của bin i trong lược đồ màu của khối k của ảnh I

- LCH có thể khắc phục nhược điểm của GCH khi có thể so sánh được sự khác nhau thông tin về không gian của các đối tượng trong bức ảnh
- Tuy nhiên LCH, đối phép xoay ảnh thì có thể LCH sẽ đưa ra kết quả kém chính xác

1.5. Cách tính Color Histogram

- Bước 1: Chọn ảnh đầu vào và chuyển sang không gian màu sắc RGB (nếu ảnh không phải là RGB).
- Bước 2: Chia ảnh thành các vùng nhỏ (cell) hoặc sử dụng toàn bộ ảnh.
- Bước 3: Đối với mỗi vùng ảnh hoặc toàn bộ ảnh, tách ra ba kênh màu sắc R (red), G (green) và B (blue).

- Bước 4: Đối với mỗi kênh màu sắc, chia khoảng giá trị màu sắc từ 0 đến 255 thành nhiều bin nhỏ hơn, tùy thuộc vào độ phân giải màu sắc mong muốn. Ví dụ, nếu chia thành 16 bin, thì mỗi bin sẽ có độ rộng là 16 (256 / 16 = 16).
- Bước 5: Đối với mỗi kênh màu sắc, đếm số lượng điểm ảnh có giá trị màu sắc nằm trong mỗi bin. Kết quả là một histogram cho mỗi kênh màu sắc.
- Bước 6: Nối các histogram của ba kênh màu sắc lại với nhau để tạo thành vector đặc trưng cho mỗi vùng ảnh hoặc toàn bộ ảnh.
- Bước 7: Chuẩn hóa histogram bằng phương pháp L2-norm. Sau khi tính toán histogram màu, ta có thể chuẩn hóa bằng cách chia histogram cho giá trị L2-norm của vector histogram.

$$||x||_2 = \sqrt{\sum_{i=1}^{N} |x_i|^2}$$

$$x = x \,/\, ||x||_2$$

Trong đó:

- x là vector histogram
- x_i là giá trị của bin thứ i
- ||x|| là giá trị L2-norm

- Bước 8: Sau khi tính histogram của toàn bộ khối, ta nối toàn bộ histogram lại thành 1 vector histogram đặc trưng duy nhất

1.6. Một số nhược điểm của Color Histogram

- Nhạy cảm với các nhiễu như thay đổi độ sáng và lỗi lượng tử hóa
- Không xem xét sự tương đồng màu sắc giữa các khối khác nhau

- Không thể xử lý xoay và dịch chuyển. Điều này có nghĩa là thông tin và vị trí và hình dạng và kết cấu của đối tượng bị bỏ qua.
- Hai hình ảnh khác nhau về mặt tri giác nhưng có phân bố màu tương tự nhau sẽ được coi là giống nhau. Do đó hình ảnh được truy xuất bằng lược đồ màu toàn cục sẽ có thể không liên quan ngữ nghĩa

2. Đặc trưng hình dạng

2.1. Đặc trưng hình dạng là gì?

- Đặc trưng hình dạng trong xử lý ảnh là những thuộc tính liên quan đến hình dạng của các đối tượng trong ảnh, như diện tích, chu vi, độ dài cạnh, góc, đường cong, đối xứng, v.v. Đặc trưng hình dạng có thể giúp phân biệt và nhận dạng các đối tượng trong ảnh.
- Để trích chọn đặc trưng hình dạng, thường cần các bước như phát hiện biên, phân vùng ảnh, rút trích đặc trưng và phân lớp
- Có nhiều phương pháp để rút trích đặc trưng hình dạng, ví dụ như Fourier descriptor, moment invariant, Hough transform, v.v

2.2. Gradient là gì?

- Gradient là một vector chỉ ra hướng mà tại đó giá trị của hàm thay đổi nhiều nhất và trở thành véc tơ 0 khi hàm đạt giá trị cực đại hoặc cực tiểu địa phương.
- Biểu diễn toán học của Gradient:

  Gradient của một hàm f(x1,x2,…,xn) được ký hiệu ∇f là một vector n chiều, mà mỗi thành phần trong vector đó là một đạo hàm riêng phần (partial derivative) theo từng biến của hàm đó:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)^T$$

- Trong xử lý ảnh, độ dốc (tức gradient) đang nói đến ở đây chính là độ dốc về mức sáng. Hay nói cách khác chính là sự thay đổi các giá trị pixel trong ảnh.
- Vùng ảnh trơn (smooth) thì các pixel trong vùng ảnh đó có giá trị xấp xỉ / gần bằng nhau, vì vậy khi tính toán đạo hàm sẽ gần bằng zero. Đạo hàm bằng 0 thể hiện không có biến thiên về giá trị (mức sáng). Điều này có nghĩa là độ dốc của các pixel trong vùng ảnh trơn gần bằng zero. Đạo hàm dương tại một pixel thể hiện rằng biến thiên mức sáng đang ở chiều hướng đi lên, ngược lại đạo hàm âm tại một pixel cho biết biến thiên mức sáng tại đó đang giảm dần. Nói tóm gọn lại gradient của ảnh chính là đạo hàm ảnh.

2.3. Histograms of oriented gradients (HOG)

- Phương pháp rút trích đặc trưng hình ảnh HOG xuất bản ở hội nghị CVPR 2005 được đề xuất bởi tác giả là Dalal và Triggs.
- Là một mô tả tính năng được sử dụng trong thị giác máy tính và xử lý hình ảnh cho mục đích phát hiện đối tượng. Kỹ thuật này đếm số lần xuất hiện của định hướng gradient trong các phần cục bộ của hình ảnh. Phương pháp này tương tự như biểu đồ định hướng cạnh, mô tả biến đổi tính năng bất biến tỷ lệ và bối cảnh hình dạng, nhưng khác ở chỗ nó được tính toán trên một lưới dày đặc các ô cách đều nhau và sử dụng chuẩn hóa độ tương phản cục bộ chồng chéo để cải thiện độ chính xác

2.4. Các bước trích chọn đặc trưng hình dạng ảnh sử dụng HOG

Bước 1: Tiền xử lý dữ liệu

Đây là một bước mà hầu hết các bạn sẽ khá quen thuộc. Tiền xử lý dữ liệu là một bước quan trọng trong bất kỳ dự án học máy nào và điều đó cũng không khác gì khi làm việc với hình ảnh.

Cần xử lý trước hình ảnh và giảm tỷ lệ chiều rộng trên chiều cao xuống. Kích thước hình ảnh được sử dụng là 128 x 128. Điều này là do ta sẽ chia hình ảnh thành các bản

vá 8*8 và 16*16 để trích xuất các tính năng. Có kích thước được chỉ định (128 x 128) sẽ làm cho tất cả các tính toán trở nên khá đơn giản.

Bước 2: Tính gradient

Bước tiếp theo là tính toán độ dốc cho mọi pixel trong ảnh. Độ dốc là sự thay đổi nhỏ theo hướng x và y. Phương pháp phổ biến nhất là áp dụng mặt nạ đạo hàm điểm, trung tâm 1-D theo một hoặc cả hai hướng ngang và dọc. Cụ thể, phương pháp này yêu cầu lọc dữ liệu màu sắc hoặc cường độ của hình ảnh bằng các hạt nhân bộ lọc sau:

$[-1, 0, 1]$ and $[-1, 0, 1]^\top$

Ta sẽ lấy một miếng vá nhỏ từ hình ảnh và tính toán độ dốc trên đó. Giả sử ma trận pixel bên dưới cho bản vá nhất định:

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Vị trí được đánh dấu là giá trị pixel 85. Bây giờ, để xác định độ dốc (hoặc thay đổi) theo hướng x, chúng ta cần trừ giá trị ở bên trái khỏi giá trị pixel ở bên phải. Tương tự, để tính độ dốc theo hướng y, chúng ta sẽ trừ giá trị pixel bên dưới khỏi giá trị pixel bên trên pixel đã chọn.

Do đó, độ dốc kết quả theo hướng x và y cho pixel này là:

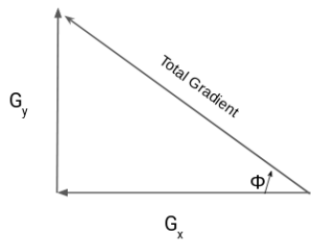
- Thay đổi theo hướng X(G_x) = 89 – 78 = 11
- Thay đổi theo hướng Y(G_y) = 68 – 60 = 8

Quá trình này sẽ cung cấp cho chúng ta hai ma trận mới – một ma trận lưu trữ độ dốc theo hướng x và ma trận lưu trữ độ dốc khác theo hướng y.

- Tính gradient: áp dụng các toán tử lấy đạo hàm theo hai hướng x và y cho ảnh, sau đó tính cường độ và hướng của gradient tại mỗi điểm ảnh.
- Tính vector đặc trưng cho từng ô (cell): chia ảnh thành các block, mỗi block lại chia thành các cell nhỏ hơn, ví dụ 8x8. Với mỗi cell, tính histogram của các hướng gradient cho các điểm ảnh trong cell đó. Số bin của histogram phụ thuộc vào số lượng hướng gradient được chọn, ví dụ 9 bin cho 9 hướng gradient từ 0 đến 180 độ.
- Chuẩn hóa khối (block): để giảm thiểu ảnh hưởng của ánh sáng, các histogram cục bộ của các cell được chuẩn hóa về độ tương phản bằng cách tính một ngưỡng cường độ trong một vùng lớn hơn cell, gọi là khối (block). Một khối có thể gồm nhiều cell, ví dụ 2x2 hoặc 3x3. Các khối có thể chồng lên nhau để tăng tính liên tục của vector đặc trưng.
- Tính toán vector HOG: sau khi chuẩn hóa, các histogram của các cell trong cùng một khối được nối lại với nhau để tạo thành một vector đặc trưng cho khối đó. Sau đó, các vector đặc trưng của các khối được nối lại với nhau để tạo thành vector HOG cho toàn bộ ảnh.

Bước 3: Tính độ lớn và hướng

Sử dụng độ dốc đã tính toán ở bước 2, bây giờ ta sẽ xác định độ lớn và hướng cho từng giá trị pixel. Đối với bước này, chúng ta sẽ sử dụng định lý Pythagoras.



Ở bước 2, ta có Gx và Gy là 11 và 8. Áp dụng định lý Pythagoras để tính tổng độ lớn của gradient:

Tổng Độ lớn Gradient = $\sqrt{Gx^2 + Gy^2} = \sqrt{11^2 + 8^2} \approx 13.6$

Tiếp theo, tính toán hướng cho cùng một pixel.

Ta có:

$\tan(\Phi) = \frac{Gy}{Gx}$

Do đó, giá trị của góc sẽ là:

$\Phi = \arctan(\frac{Gy}{Gx}) = \arctan(\frac{8}{11}) = 36°$

Định hướng xuất hiện là 36. Bây giờ, với mỗi giá trị pixel, chúng ta có tổng độ dốc (độ lớn) và hướng (hướng). Chúng ta cần tạo biểu đồ bằng cách sử dụng các độ dốc và hướng này.

Các phương pháp khác nhau để tạo biểu đồ sử dụng độ dốc và định hướng:

Biểu đồ tần suất là một đồ thị biểu thị phân bố tần suất của một tập hợp dữ liệu liên tục. Chúng ta có biến (ở dạng cột) trên trục x và tần số trên trục y. Ở đây, chúng ta sẽ lấy góc hoặc hướng trên trục x và tần số trên trục y.

- Cách 1: Lấy từng giá trị pixel, tìm hướng của pixel và cập nhật bảng tần số

  Đây là quy trình cho pixel được đánh dấu (85). Vì hướng của pixel này là 36, ta sẽ thêm một số vào giá trị góc 36, biểu thị tần số:

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Frequency						1									
Angle	1	2	3	4 ...	35	36	37	38	39....	175	176	177	178	179	180

Quá trình tương tự được lặp lại cho tất cả các giá trị pixel và kết thúc bằng một bảng tần số biểu thị các góc và sự xuất hiện của các góc này trong ảnh. Bảng tần số này có thể được sử dụng để tạo biểu đồ với các giá trị góc trên trục x và tần số trên trục y.

- Cách 2:

Tạo các tính năng biểu đồ cho các giá trị bin cao hơn

Phương pháp này tương tự như phương pháp trước, ngoại trừ ở đây chúng tôi có kích thước cột là 20. Vì vậy, số lượng cột sẽ nhận được ở đây là 9.

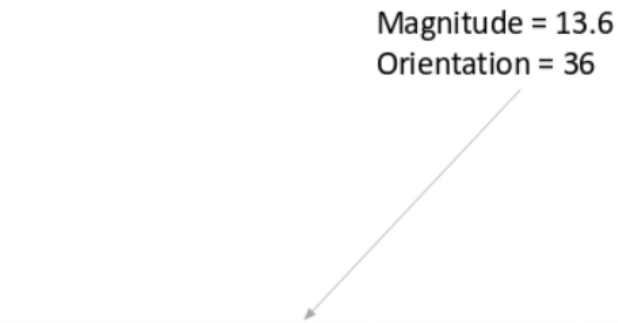
Một lần nữa, đối với mỗi pixel, ta sẽ kiểm tra hướng và lưu trữ tần số của các giá trị hướng ở dạng ma trận 9 x 1. Vẽ sơ đồ này sẽ cho chúng ta biểu đồ:

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Magnitude		1							
Bin	0	20	40	60	80	100	120	140	160

- Cách 3:

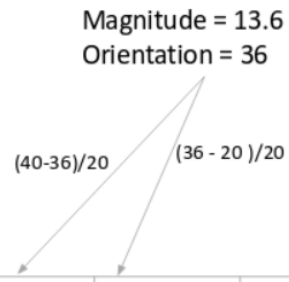
Hai phương pháp trên chỉ sử dụng các giá trị định hướng để tạo biểu đồ và không tính đến giá trị độ dốc. Đây là một cách khác để chúng ta có thể tạo biểu đồ thay vì sử dụng tần số, chúng ta có thể sử dụng cường độ gradient để điền vào các giá trị trong ma trận. Dưới đây là một ví dụ về điều này:



Magnitude		13.6							
Bin	0	20	40	60	80	100	120	140	160

- Cách 4:

Hãy thực hiện một sửa đổi nhỏ cho phương pháp trên. Ở đây, ta sẽ thêm phần đóng góp của độ dốc của pixel vào các thùng ở hai bên của độ dốc pixel. Hãy nhớ rằng, đóng góp cao hơn phải là giá trị bin gần với định hướng hơn.



Magnitude		(4/20)*13.6	(16/20)*13.6						
Bin	0	20	40	60	80	100	120	140	160

Đây chính xác là cách biểu đồ được tạo trong bộ mô tả tính năng HOG.

Bước 4: Tính toán histogram of gradient trong các ô 8x8 (9x1)

Biểu đồ được tạo trong bộ mô tả tính năng HOG không được tạo cho toàn bộ hình ảnh. Thay vào đó, hình ảnh được chia thành các ô 8x8 và biểu đồ của độ dốc định hướng được tính cho mỗi ô.

Bằng cách đó, ta có được các tính năng (hoặc biểu đồ) cho các bản vá nhỏ hơn, lần lượt đại diện cho toàn bộ hình ảnh. Ta có thể thay đổi giá trị này ở đây từ 8 x 8 thành 16 x 16 hoặc 32 x 32.

Nếu chúng ta chia hình ảnh thành 8×8 ô và tạo biểu đồ, chúng ta sẽ nhận được ma trận 9 x 1 cho mỗi ô. Ma trận này được tạo bằng phương pháp 4 trong bước 3.

Bước 5: Chuẩn hóa độ dốc trong ô 16x16 (36x1)

Mặc dù đã có các tính năng HOG được tạo cho các ô 8×8 của hình ảnh, độ dốc của hình ảnh rất nhạy cảm với ánh sáng tổng thể. Điều này có nghĩa là đối với một hình ảnh cụ thể, một số phần của hình ảnh sẽ rất sáng so với các phần khác

Ta không thể loại bỏ hoàn toàn điều này khỏi hình ảnh. Nhưng ta có thể giảm sự thay đổi ánh sáng này bằng cách chuẩn hóa độ dốc bằng cách lấy các khối 16 × 16. Dưới đây là một ví dụ có thể giải thích cách các khối 16×16 được tạo:

Ở đây, ta sẽ kết hợp bốn ô 8×8 để tạo một khối 16×16. Và ta đã biết rằng mỗi ô 8×8 có một ma trận 9×1 cho một biểu đồ. Vì vậy, ta sẽ có bốn ma trận 9×1 hoặc một ma trận 36×1. Để chuẩn hóa ma trận này, ta sẽ chia từng giá trị này cho căn/ bậc hai của tổng bình phương các giá trị. Về mặt toán học, đối với một vectơ V đã cho:

$V = [a_1, a_2, a_3, \ldots a_{36}]$

Ta tính toán gốc của tổng bình phương:

$k = \sqrt{a_1^2 + a_2^2 + a_3^2 + \cdots + a_{36}^2}$

Và chia tất cả giá trị trong vector với giá trị k này:

$$\text{Normalised Vector} = \left(\frac{a_1}{k}, \frac{a_2}{k}, \frac{a_3}{k}, \ldots \frac{a_{36}}{k}\right)$$

Kết quả sẽ là một vectơ chuẩn hóa có kích thước 36 × 1.

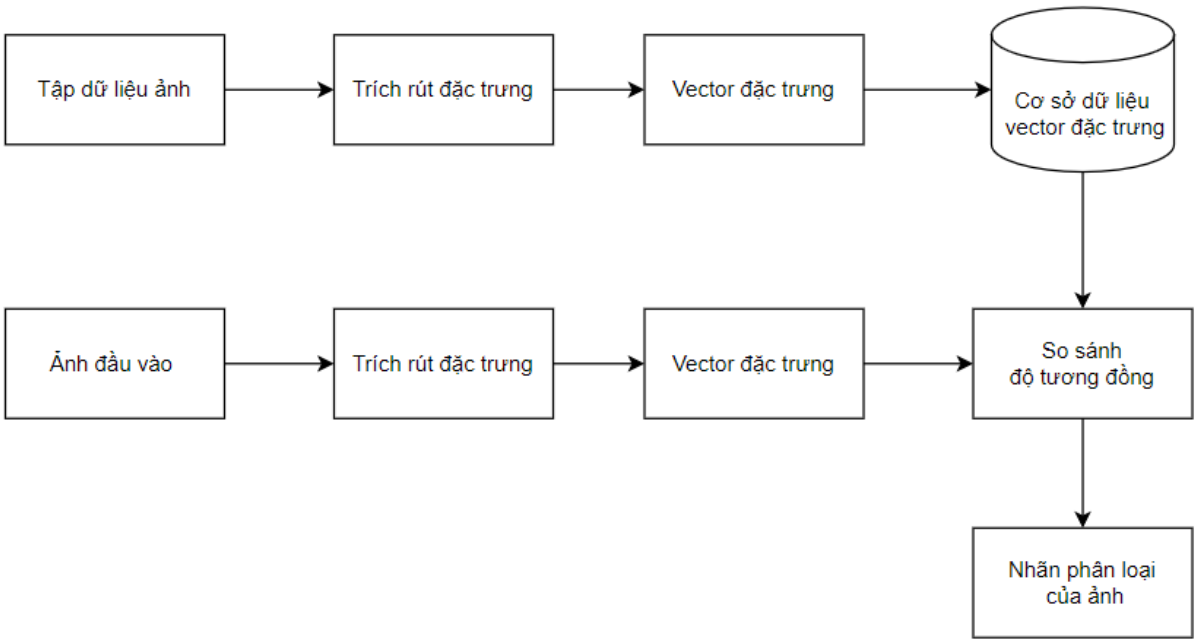
Bước 6: Tính toán Biểu đồ của vectơ đặc trưng Định hướng Gradients

Để tính toán véc-tơ đặc trưng cuối cùng cho toàn bộ bản vá hình ảnh, các véc-tơ 36×1 được nối thành một véc-tơ khổng lồ. Kích thước của vectơ này là bao nhiêu?

- Chúng ta có bao nhiêu vị trí của các khối 16×16? Có 128 / 16 * 2 – 1 = 15 vị trí ngang và 128 / 16 * 2 – 1 = 15 vị trí dọc, tổng cộng là 15 * 15 = 225 vị trí.
- Mỗi khối 16×16 được biểu diễn bằng một vectơ 36×1. Vì vậy, khi chúng ta nối tất cả chúng thành một vectơ khuếch đại, chúng ta thu được một vectơ có kích thước 36 * 225 = 8100 chiều.

III. Xây dựng hệ thống nhận diện ảnh hoa

1. Sơ đồ khối của hệ thống



a. Xây dựng cơ sở dữ liệu các đặc trưng ảnh

B1. Tập dữ liệu ảnh:

Ảnh sau khi được thu thập sẽ được tiến hành tách bỏ nền, cắt ảnh và chỉnh tất cả ảnh về cùng 1 kích thước 128x128

B2: Trích rút đặc trưng ảnh:

- Thực hiện tiền xử lý ảnh để chuẩn bị cho việc trích rút đặc trưng
- Sử dụng các thuật toán trích rút đặc trưng được trình bày ở phần II như Color Histogram, Histogram of Gradient, Local Binary Pattern để thực hiện

B3: Vector đặc trưng:

- Sau khi thực hiện các thuật toán sẽ thu được các vector đặc trưng
- Các vector đặc trưng sẽ được tập hợp lại để lưu vào cơ sở dữ liệu

B4: Cơ sở dữ liệu:

Các vector sẽ được lưu trữ trong 1 tệp để có thể sử dụng phân loại ảnh đầu vào

b. Phân loại ảnh

B1: Ảnh đầu vào đã được tiền xử lý xóa nền, cắt ảnh và chỉnh về kích thước về 128x128

B2: Trích rút đặc trưng ảnh:

- Thực hiện tiền xử lý ảnh để chuẩn bị cho việc trích rút đặc trưng
- Sử dụng các thuật toán trích rút đặc trưng được trình bày ở phần II như Color Histogram, Histogram of Gradient, Local Binary Pattern để thực hiện

B3: Vector đặc trưng: Sau khi thực hiện các thuật toán sẽ thu được các vector đặc trưng, các vector đặc trưng sẽ được dùng để phân loại ảnh

B4: So sánh độ tương đồng:

- Các vector đặc trưng của ảnh đầu vào so sánh với các vector đặc trưng của các ảnh trong cơ sở dữ liệu.

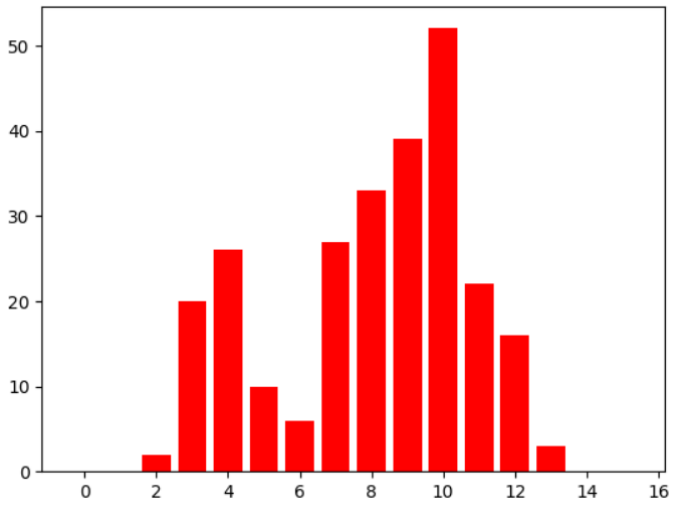
- Chọn ra các ảnh có độ tương đồng cao với ảnh đầu vào

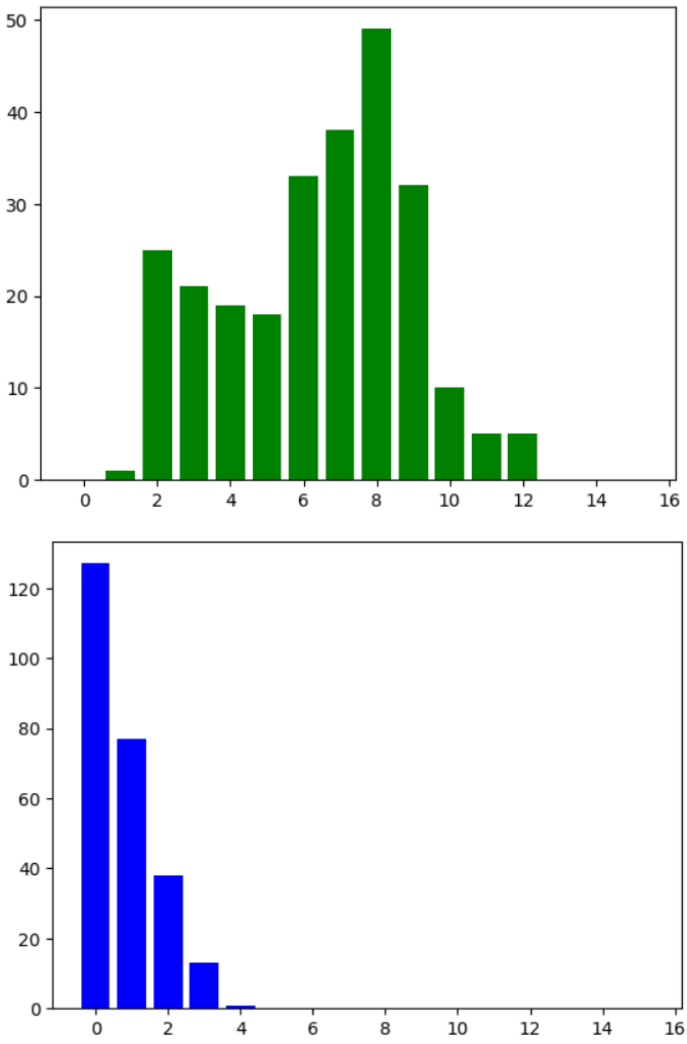
B5: Chọn ra nhãn có kết quả tốt nhất trong các kết quả và trả về

2. Các thuộc tính được sử dụng và các kỹ thuật trích rút

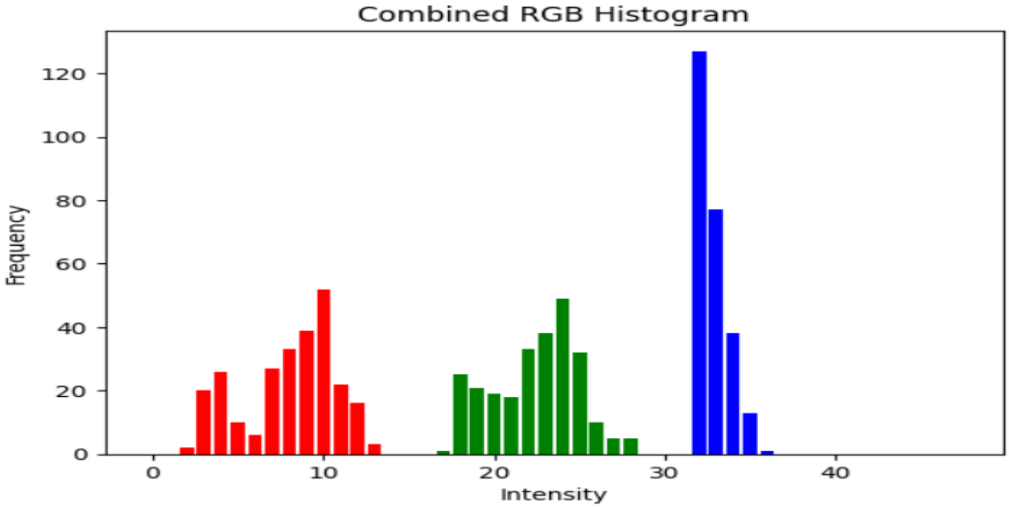
2.1. Màu sắc

- Bước 1: Chọn ảnh đầu vào và chuyển sang không gian màu sắc RGB (nếu ảnh không phải là RGB).
- Bước 2: Chia ảnh thành các vùng nhỏ (cell), mỗi vùng có kích thước 16x16 pixel
- Bước 3: Đối với mỗi vùng ảnh hoặc toàn bộ ảnh, tách ra ba kênh màu sắc R (red), G (green) và B (blue).
- Bước 4: Đối với mỗi kênh màu sắc, chia khoảng giá trị màu sắc từ 0 đến 255 thành nhiều bin nhỏ hơn, tùy thuộc vào độ phân giải màu sắc mong muốn. Ta chia thành 16 bin, thì mỗi bin sẽ có độ rộng là 16 (256 / 16 = 16).
- Bước 5: Đối với mỗi kênh màu sắc, đếm số lượng điểm ảnh có giá trị màu sắc nằm trong mỗi bin. Kết quả là một histogram cho mỗi kênh màu sắc.





- Bước 6: Nối các histogram của ba kênh màu sắc lại với nhau để tạo thành vector đặc trưng cho mỗi vùng ảnh



- Bước 7: Chuẩn hóa histogram bằng phương pháp L2-norm. Sau khi tính toán histogram màu, ta có thể chuẩn hóa bằng cách chia histogram cho giá trị L2-norm của vector histogram.

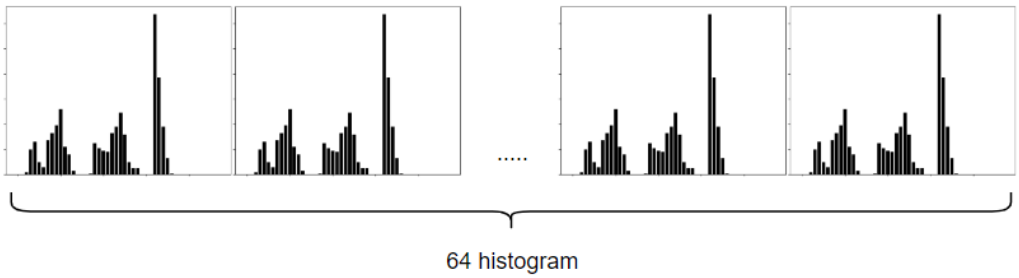
$$||x||_2 = \sqrt{\sum_{i=1}^{N} |x_i|^2}$$

$$x = x \,/\, ||x||_2$$

Trong đó:

- x là vector histogram
- x_i là giá trị của bin thứ i
- ||x|| là giá trị L2-norm

- Bước 8: Sau khi tính histogram của toàn bộ 64 cell, ta nối 64 histogram lại thành 1 histogram có 3072 chiều.



2.2. Hình dạng

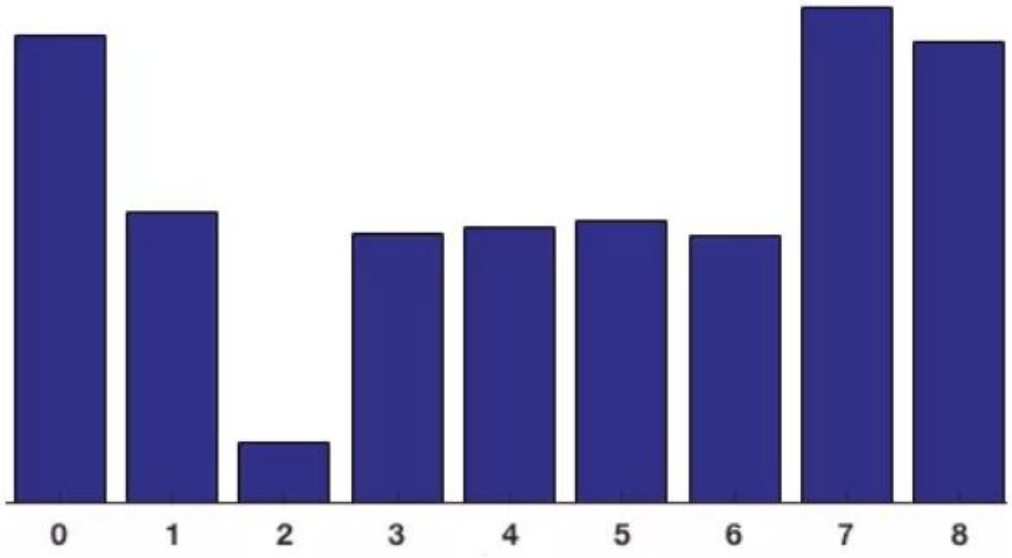
- Bước 1: Chọn ảnh đầu vào và chuyển sang không gian màu sắc RGB (nếu ảnh không phải là RGB).
- Bước 2: Tính Gradient của tất cả các pixel trên ảnh: Đối với mỗi kênh màu tính giá trị cường độ và hướng tại pixel tương ứng, sau đó lấy ra giá trị có cường độ lớn nhất.
- Bước 3: Chia ảnh thành các vùng nhỏ (cell), mỗi vùng có kích thước 8x8 pixel
- Bước 4: Tính độ lớn và hướng trên từng pixel theo công thức:

$$left\_bin = \frac{right\_angle - angle}{\frac{\pi}{oriantations}} * total\_gradient$$

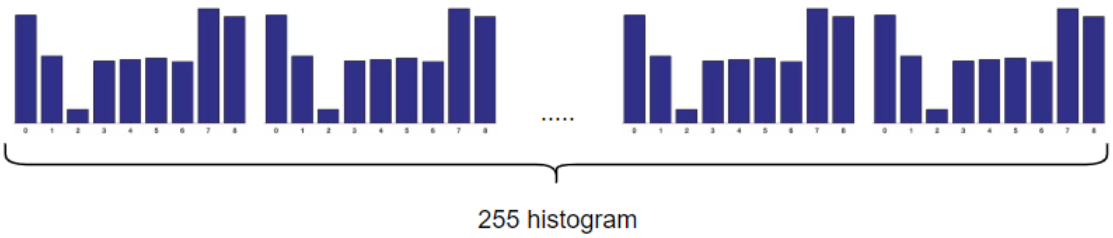
$$right\_bin = \frac{angle - left\_angle}{\frac{\pi}{oriantations}} * total\_gradient$$

ở đây ta chia mỗi cell thành 9 bin tương ứng với oriantations = 9

- Bước 5: Ghép các giá trị được tính ở bước 4 trong mỗi cell lại với nhau để tạo thành 1 histogram ứng với mỗi cell được thể hiện dưới dạng một mảng numpy 9 chiều



- Bước 6: Chuẩn hóa độ dốc trong ô 16x16:
  - Ghép 4 histogram tương ứng của mỗi 4 ô liền kề lại với nhau để tạo thành một numpy 36 chiều.
  - Thực hiện chuẩn hóa bằng cách lấy từng giá trị trên mảng numpy chia cho tổng bình phương tất cả các giá trị
- Bước 7: Ghép các histogram lại tạo thành một numpy duy nhất đại diện cho đặc trưng hình dạng của ảnh. Do ảnh có kích thước 128x128 nên ta sẽ có được 256 cell 8x8. Từ 256 cell đó ta ghép thành $(\sqrt{256} - 1)^2 = 225$ block. Với mỗi block ta thu được một numpy 36 chiều nên vector đặc trưng sẽ có kích thước: 225 * 36 = 8100 chiều.



3. Lưu trữ và nhận dạng

3.1. Cách lưu trữ các thuộc tính ảnh hoa

Các vector đặc trưng của hình ảnh sẽ được lưu trữ vào một file json, cấu trúc của file json như sau:

```
{
  "data": [
    {
      "label": "hoa cúc",
      "color_feature": [0.2, 0.3, 0.5, 0.1, 0.4, ...],
      "hog_feature" :  [0.1, 0.4, 0.3, 0.2, 0.5, ...],
      ]
    },
    {
      "label": "hoa hướng dương",
      "color_feature": [0.1, 0.2, 0.3, 0.4, 0.5, ...],
      "hog_feature" :  [0.2, 0.3, 0.4, 0.1, 0.5, ...],
    },
    ...
  ]
}
```

Trong đó:

- “data” là một mảng các đối tượng ảnh hoa
- Mỗi đối tượng ảnh hoa sẽ có các trường sau:
  - “label”: tên nhãn của ảnh hoa

- “color_feature”: vector đặc trưng màu sắc
- “hog_feature” : vector đặc trưng hog

3.2. Cách nhận dạng ảnh hoa

3.2.1. Thuật toán KNN

- Sử dụng thuật toán KNN(K-Nearest Neighbor) để phân loại ảnh hoa đầu vào. K Nearest Neighbor là một thuật toán đơn giản lưu trữ tất cả các trường hợp có sẵn và phân loại dữ liệu hoặc trường hợp mới dựa trên thước đo độ tương tự. Nó chủ yếu được sử dụng để phân loại một điểm dữ liệu dựa trên cách các điểm lân cận của nó được phân loại.
- Các bước trong KNN
  - Ta có D là tập các điểm dữ liệu đã được gán nhãn và A là dữ liệu chưa được phân loại
  - Đo khoảng cách từ dữ liệu A đến tất cả các dữ liệu khác đã được phân loại trong D
  - Chọn ra k khoảng cách nhỏ nhất
  - Kiểm tra danh sách các lớp có khoảng cách ngắn nhất và đếm số lượng của mỗi lớp xuất hiện
  - Lấy đúng lớp (lớp xuất hiện nhiều lần)

- Làm cách nào để chọn giá trị của k trong thuật toán KNN
  - K trong thuật toán KNN dựa trên tính tương tự của thuộc tính, việc chọn đúng giá trị của K là một quá trình được gọi là điều chỉnh tham số và rất quan trọng để có độ chính xác cao hơn. Tìm giá trị của k không dễ
  - Các ý tưởng về chọn giá trị cho K
    - Không có phương pháp có cấu trúc nào để tìm giá trị tốt nhất cho K. Chúng ta cần tìm ra các giá trị khác nhau bằng cách thử và sai.
    - Việc chọn các giá trị nhỏ hơn cho K có thể gây nhiễu và sẽ có ảnh hưởng hơn đến kết quả
    - Các giá trị lớn hơn của K sẽ có kết quả tốt hơn
    - Cố gắng giữ giá trị của k lẻ để tránh nhầm lẫn giữa 2 lớp

=> Chọn giá trị K = 5. Đây là giá trị k đủ lớn để loại bỏ nhiễu và không gây ra tình trạng quá khớp và cũng đủ nhỏ để không bị ảnh hưởng bởi các điểm dữ liệu xa.

3.2.2. Độ đo khoảng cách

- Sử dụng công thức khoảng cách Euclid để tính toán khoảng cách giữa các điểm dữ liệu

$$D(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

trong đó:

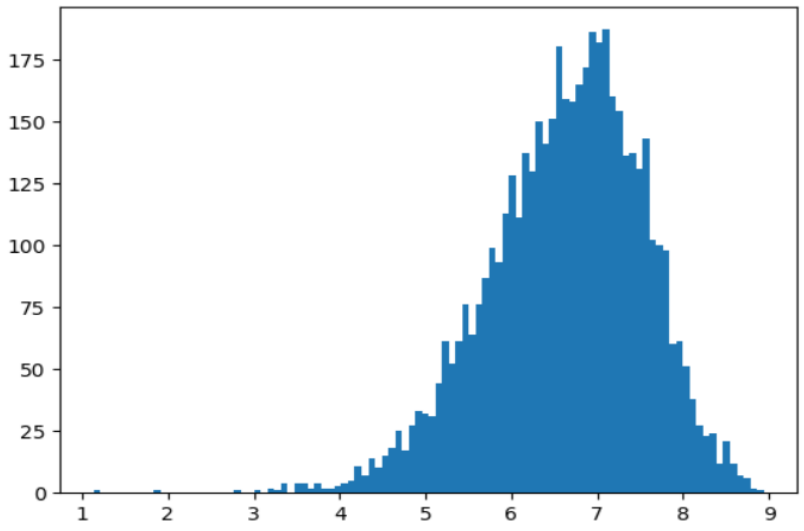
- x là vector đặc trưng của dữ liệu trong cơ sở dữ liệu
- y là vector đặc trưng điểm dữ liệu đầu vào
- xi là giá trị của phần tử thứ i trong vector của x
- yi là giá trị của phần tử thứ i trong vector của y

- Ta sẽ tính toán khoảng cách Euclid cho từng cặp feature vector giữa 2 bức ảnh. Kết quả ta có 2 giá trị khoảng cách cho 2 feature vector
- Tổng hợp hai giá trị khoảng cách thành một giá trị duy nhất để đo độ tương đồng giữa hai bức ảnh. Có nhiều cách để tổng hợp ba giá trị khoảng cách thành một giá trị duy nhất, ví dụ như lấy trung bình, lấy trọng số, lấy min hoặc max, etc. Mỗi cách có ưu nhược điểm và tác động đến kết quả của việc so sánh hai bức ảnh.
  - Nếu lấy trung bình, thì ta sẽ coi ba feature vector là có vai trò như nhau trong việc đánh giá độ tương đồng.
  - Nếu lấy trọng số, thì ta sẽ coi một số feature vector là quan trọng hơn các feature vector khác.

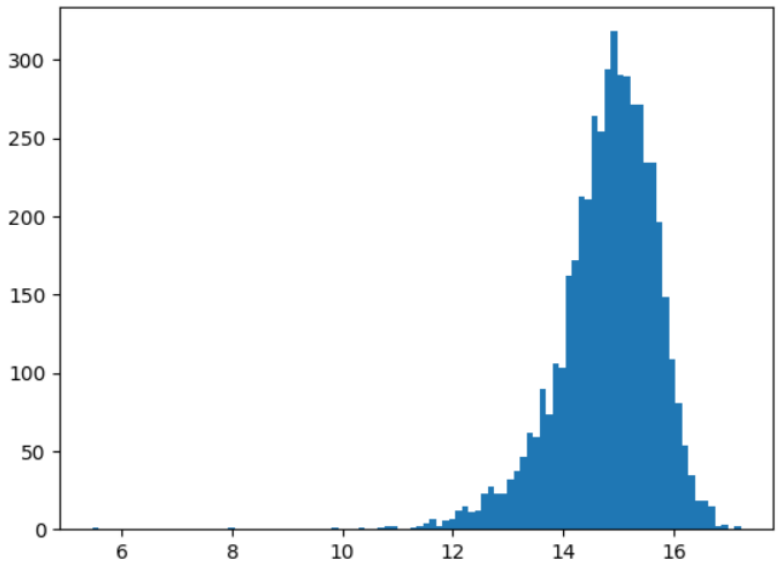
- Nếu lấy min hoặc max, thì ta sẽ coi một feature vector là quyết định cho việc so sánh hai bức ảnh.

3.2.3. Xác định trọng số và ngưỡng dự đoán

- Sau khi trích rút được đặc trưng của toàn bộ ảnh trong tập dữ liệu, ta sẽ xét phân bố khoảng cách các đặc trưng của các ảnh trong dữ liệu.
- Đồ thị phân bố khoảng cách về đặc trưng màu sắc



- Đồ thị phân bố khoảng cách về đặc trưng hình dạng



- Các chỉ số về giá trị trung bình, độ lệch chuẩn, giá trị thấp nhất, giá trị cao nhất

	color_distance	hog_distance
mean	6.630225	14.806848
std	0.909709	0.884638
min	1.131135	5.463728
max	8.937278	17.216010

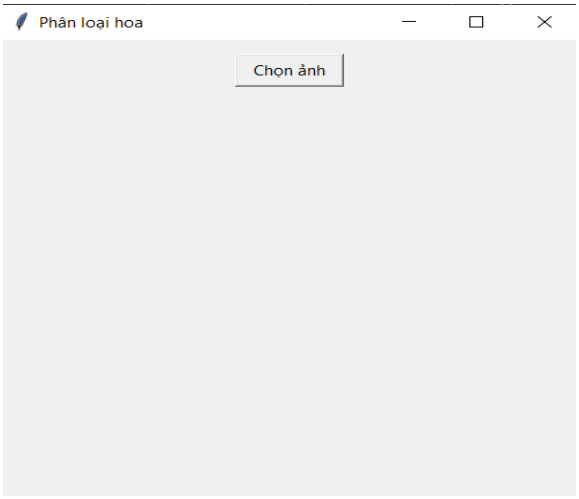
- Từ hai đồ thị và bảng số liệu về phân bố khoảng cách của 2 đặc trưng, ta thấy cả 2 đặc trưng đều có xu hướng phân bố chuẩn, và độ lệch chuẩn của 2 đặc trưng cũng đều ngang nhau (0.91 và 0.88). Như vậy có thể coi rằng độ quan trọng của 2 đặc trưng là như nhau khi tổng hợp khoảng cách giữa 2 điểm dữ liệu. Vậy ta có thể tổng hợp khoảng cách giữa 2 điểm dữ liệu bằng cách cộng 2 giá trị khoảng cách lại với nhau.

$$D(x, y) = Distance\_color + Distance\_shape$$

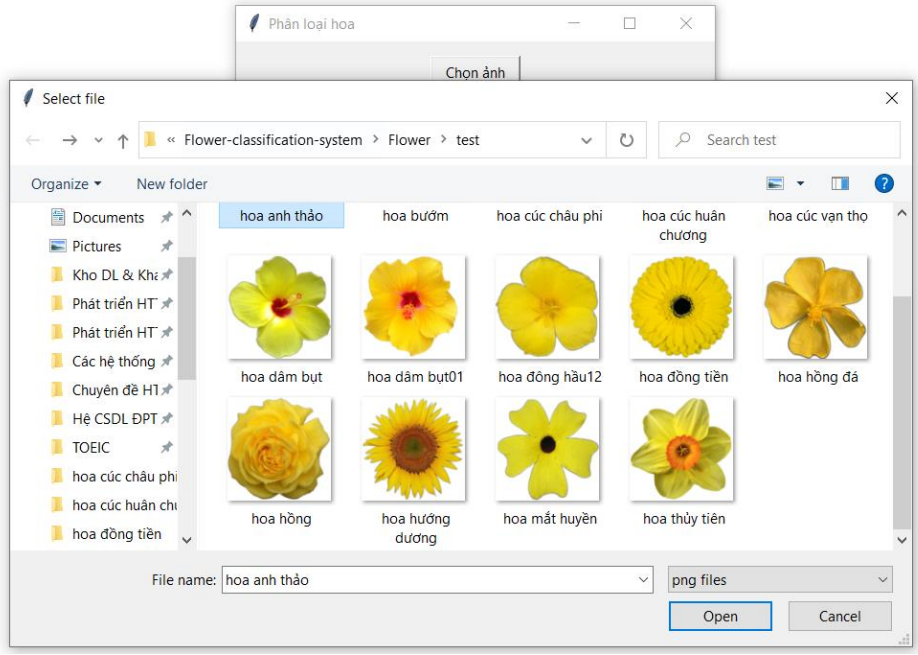
- Xét ngưỡng khoảng cách: Việc xét ngưỡng để xác định xem khoảng cách dữ liệu có nằm trong ngưỡng có thể dự đoán không. Nếu khoảng cách quá xa nằm ngoài ngưỡng thì có thể cho là dữ liệu đầu vào không khớp với dữ liệu nào trong tập dữ liệu. Ta có thể dựa vào giá trị trung bình trong phân phối khoảng cách của 2 đặc trưng để xác định ngưỡng. Giá trị trung bình của đặc trưng màu là 6.6 và của hình dạng là 14.8. Như vậy có thể xác định ngưỡng bằng cách tổng hợp 2 giá trị này lại và chọn ngưỡng là 21.

4. Demo hệ thống

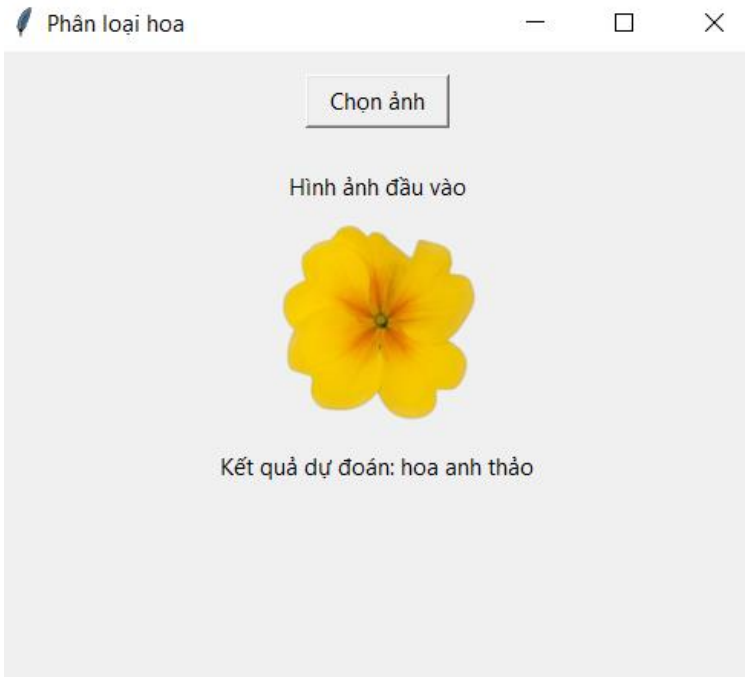
- Giao diện ban đầu của hệ thống

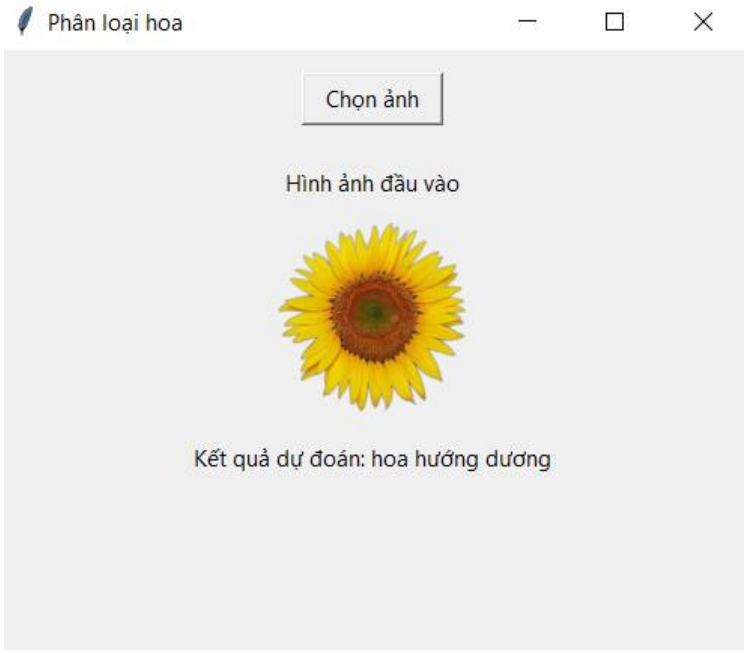


- Bấm nút "Chọn ảnh" để chọn ảnh cần dự đoán trong folder

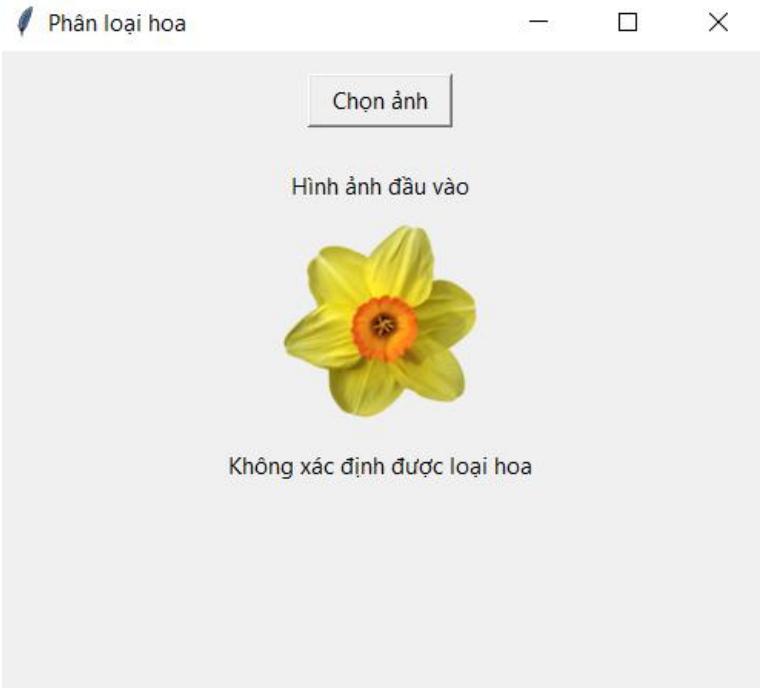


- Chọn ảnh cần dự đoán và bấm Open
    + Trường hợp 1: Ảnh loài hoa đã có trong tập dữ liệu, hệ thống sẽ hiển thị kết quả dự đoán về tên loài hoa

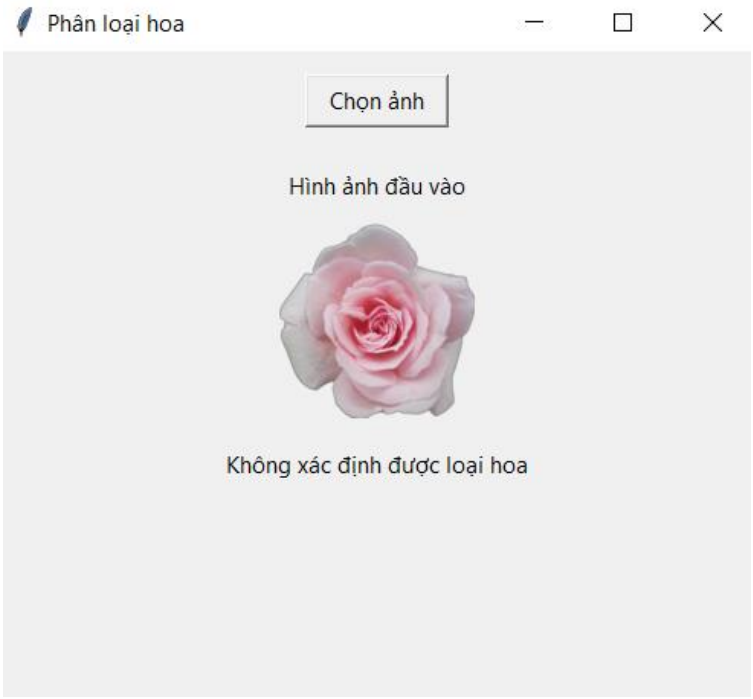

Phân loại hoa
Chọn ảnh
Hình ảnh đầu vào
Kết quả dự đoán: hoa anh thảo


Phân loại hoa
Chọn ảnh
Hình ảnh đầu vào
Kết quả dự đoán: hoa hướng dương

+ Trường hợp 2: Ảnh loài hoa chưa có trong tập dữ liệu, hệ thống sẽ hiển thị kết quả "Không xác định được loài hoa"
    - Ảnh đầu vào là hoa thủy tiên



    - Ảnh đầu vào là hoa hồng



5. Đánh giá kết quả

- Sử dụng bộ ảnh test có 14 ảnh với 11 ảnh là loài hoa có trong bộ dữ liệu và 3 ảnh là loài hoa chưa có trong bộ dữ liệu. Sau khi thực hiện dự đoán thì kết quả cho ra như sau:
  - 11 ảnh loài hoa có trong dữ liệu:
    + Dự đoán đúng nhãn: 10
    + Dự đoán sai nhãn: 1
  - 3 ảnh loài hoa không có trong dữ liệu:
    + Dự đoán đúng (Không xác định được) : 2
    + Dự đoán sai (Nhãn trong tập dữ liệu) : 1
- Dựa vào kết quả trên có thể rút ra một số kết luận:
  - Đối với 11 ảnh loài hoa có trong tập dữ liệu, việc dự đoán sai nhãn chỉ xảy ra trên 1 trong 11 ảnh, cho thấy hệ thống vẫn có độ chính xác khá cao trong việc phân loại các loài hoa thuộc tập dữ liệu
  - Đối với 3 ảnh loài hoa không có trong tập dữ liệu, hệ thống dự đoán sai 1 ảnh trong 3 ảnh. Điều này cho thấy hệ thống chưa thế phân loại các loài hoa mới một cách chính xác, có thể do sự khác biệt về màu sắc hoặc hình dạng khá thấp dẫn đến khác biệt khoảng cách giữa các ảnh không cao.

Tổng quát, hệ thống phân loại ảnh hoa dựa trên trích rút đặc trưng về màu sắc và hình dạng, sử dụng thuật toán KNN và khoảng cách Euclidean để dự đoán, cho thấy độ chính xác khá tốt đối với loài hoa có trong tập dữ liệu, tuy nhiên cần cải thiện để có thể phân loại chính xác các loài hoa mới. Cách tiếp cận này có thể được cải thiện bằng cách xác định ngưỡng tối ưu hơn đối với các loài hoa mới, sử dụng các độ đo khoảng cách khác và sử dụng các phương pháp trích rút đặc trưng phức tạp hơn để đạt được độ chính xác tốt hơn.

IV. Phụ lục

1. Code trích rút đặc trưng màu sắc

```
Class Feature:
    def color_histogram(self, image, num_bins=16, block=(16,16)):
        vector_features = []
        num_block_row = int(image.shape[0] / block[0])
        num_block_col = int(image.shape[1] / block[1])

        for i in range(num_block_row):
            for j in range(num_block_col):
                block_image = image[i*block[0]: (i+1)*block[0], j*block[1]:
    (j+1)*block[1]]
                histogram = self.calculateRGBHistogram(block_image, num_bins)
                vector_features.append(histogram)
        vector_features = np.concatenate(vector_features)
        return vector_features

    def calculateRGBHistogram(self, image, num_bins):
        value_bin = 256 / num_bins
        histogram_of_red    = np.zeros(num_bins)
        histogram_of_green  = np.zeros(num_bins)
        histogram_of_blue   = np.zeros(num_bins)

        for i in range(image.shape[0]):
            for j in range(image.shape[1]):
                histogram_of_red[int(image[i][j][0] / 256 * value_bin)] += 1
                histogram_of_green[int(image[i][j][1] / 256 * value_bin)] += 1
                histogram_of_blue[int(image[i][j][2] / 256 * value_bin)] += 1

        histogram = np.concatenate((histogram_of_red, histogram_of_green,
    histogram_of_blue))
        # histogram = histogram / np.sum(histogram)
        histogram = histogram / np.linalg.norm(histogram)

        return histogram
```

2. Code trích rút đặc trưng hình dạng

```
class Gradient:
    def __init__(self, left=0, right=0, top=0, bottom=0):
        self.Gx = right - left
        self.Gy = top - bottom
        self.total_gradient = math.sqrt(self.Gx * self.Gx + self.Gy * self.Gy)
        angle = np.arctan2(self.Gy, self.Gx)
        self.angle = angle if angle >= 0 else math.pi + angle

    def calculateGradient(image):
        row_size, col_size = image.shape[0], image.shape[1]
        if image.ndim == 2:
            image = image.reshape((row_size, col_size, 1))
            isGrayImage = True
        else:
            isGrayImage = False
        gradient = []
        for row in range(row_size):
            gradient.append([])
            for col in range(col_size):
                gradient[row].append(Gradient())
                row_left = row
                col_left = col - 1
                row_right = row
                col_right = col + 1
                row_top = row - 1
                col_top = col
                row_bottom = row + 1
                col_bottom = col
                if isGrayImage:
                    length = 1
                else:
                    length = 3
                for i in range(length):
                # i = {
                #     0: 'red',
                #     1: 'green',
                #     2: 'blue',
```

```
                # }
                    if Gradient.isExist(row_left, col_left, row_size, col_size):
                        left = image[row_left][col_left][i].astype(np.int32)
                    else:
                        left = 0
                    if Gradient.isExist(row_right, col_right, row_size, col_size):
                        right = image[row_right][col_right][i].astype(np.int32)
                    else:
                        right = 0
                    if Gradient.isExist(row_top, col_top, row_size, col_size):
                        top = image[row_top][col_top][i].astype(np.int32)
                    else:
                        top = 0
                    if Gradient.isExist(row_bottom, col_bottom, row_size, col_size):
                        bottom = image[row_bottom][col_bottom][i].astype(np.int32)
                    else:
                        bottom = 0
                    gradient_by_color = Gradient(left, right, top, bottom)
                    if gradient[row][col].total_gradient <
    gradient_by_color.total_gradient:
                        gradient[row][col].total_gradient =
    gradient_by_color.total_gradient
                        gradient[row][col].angle = gradient_by_color.angle
            return np.array(gradient)


        def isExist(x, y, row_size, col_size):
            if x < 0 or y == col_size or y < 0 or x == row_size:
                return False
            return True

class Feature():
        def calculateFeature(self, histogram, cells_per_block):
            vector_features = []
            row_block_size = histogram.shape[0]
            col_block_size = histogram.shape[1]
            row_cell_start, col_cell_start, row_cell_end, col_cell_end = 0, 0,
    cells_per_block[0], cells_per_block[1]
            while(True):
```

```
            block = histogram[row_cell_start:row_cell_end,
    col_cell_start:col_cell_end]
            k = math.sqrt(np.sum(block * block))
            if k != 0:
                block = block / k
            vector_features.append(block)
            if col_cell_end < col_block_size:
                col_cell_start += 1
                col_cell_end += 1
            else:
                if row_cell_end < row_block_size:
                    row_cell_start += 1
                    row_cell_end += 1
                    col_cell_start = 0
                    col_cell_end = cells_per_block[1]
                else:
                    break
        vector_features = np.array(vector_features)
        return vector_features.reshape(-1)

    def calculateHistogramOfGradient(self, gradient, orientations,
    pixel_per_cell):
        row_size, col_size = gradient.shape[0], gradient.shape[1]
        histogram = []
        histogram_of_row = []
        row_start, col_start, row_end, col_end = 0, 0, pixel_per_cell[0],
    pixel_per_cell[1]
        pi_per_orientations = math.pi / orientations
        while(True):
            histogram_of_cell = [0 for i in range(orientations)]
            cell = gradient[row_start:row_end, col_start:col_end]
            # chuyển thành 1 chiều
            cell = np.ravel(cell)
            # chuyển thành mảng chứa 1 mảng total_gradient và 1 mảng chứa angle
            vfunc = np.vectorize(self.getValue)
            cell = vfunc(cell)
            for i in range(len(cell[0])):
                total_gradient = cell[0][i]
                angle = cell[1][i]
```

```
                angle_per_pi_per_orientations = angle / pi_per_orientations
                if angle_per_pi_per_orientations ==
int(angle_per_pi_per_orientations):
                    left_bin = int(angle_per_pi_per_orientations) - 1
                else:
                    left_bin = int(angle_per_pi_per_orientations)
                left_angle = left_bin * pi_per_orientations
                right_angle = (left_bin + 1) * pi_per_orientations
                if left_bin < orientations - 1:
                    right_bin = left_bin + 1
                else:
                    right_bin = 0
                histogram_of_cell[left_bin] += (right_angle - angle) /
pi_per_orientations * total_gradient
                histogram_of_cell[right_bin] += (angle - left_angle) /
pi_per_orientations * total_gradient
            histogram_of_row.append(histogram_of_cell)
            if col_end < col_size:
                col_start = col_end
                col_end = col_start + pixel_per_cell[1]
            else:
                histogram.append(histogram_of_row)
                histogram_of_row = []
                if row_end < row_size:
                    row_start = row_end
                    row_end = row_start + pixel_per_cell[0]
                    col_start = 0
                    col_end = pixel_per_cell[1]
                else:
                    break
        return np.array(histogram)


    def getValue(self, g):
        return (g.total_gradient, g.angle)

    def hog(self, image, orientations=9, pixel_per_cell=(8,8),
cells_per_block=(2,2)):
        gray_image = self.rbgToGray(image)
```

```
        gradient = Gradient.calculateGradient(gray_image)
        histogram = self.calculateHistogramOfGradient(gradient, orientations,
pixel_per_cell)
        self.shape = self.calculateFeature(histogram, cells_per_block)
        return self.shape
```

3. Trích rút đặc trưng từ tập dữ liệu

```
folder_path = './Flower/'
subfolders = [f.name for f in os.scandir(folder_path) if f.is_dir()]

feature_data = []
feature = Feature()
for subfolder in subfolders:
    # Lấy danh sách các tệp trong thư mục
    files = os.listdir(os.path.join(folder_path, subfolder))

    for file_name in files:
        data = {}
        data['label'] = subfolder
        # Đọc ảnh
        img = Image.open(os.path.join(folder_path, subfolder, file_name))
        img = np.array(img)
        # Tính toán đặc trưng
        color_feature = feature.color_histogram(img)
        hog_feature = feature.hog(img)
        lbp_feature = feature.lbp(img)
        data['color_feature'] = color_feature.tolist()
        data['hog_feature'] = hog_feature.tolist()
        feature_data.append(data)
```

```
data = {}
data['data'] = feature_data


with open('data.json', 'w', encoding='utf-8') as f:
    json.dump(data, f)
```

4. Code giao diện

```
class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Phân loại hoa")
        self.width = 500
        self.height = 500
        screen_width = self.winfo_screenwidth()
        screen_height = self.winfo_screenheight()
        x = (screen_width / 2) - (self.width / 2)
        y = (screen_height / 2) - (self.height / 2)
        self.geometry('{}x{}+{}+{}'.format(self.width, self.height, int(x), int(y)))

        self.open_button = tk.Button(self, text="Chọn ảnh",
command=self.open_file)
        self.open_button.pack(pady=15, ipadx=10,)
        self.label = tk.Label(self)
        self.label.pack(pady=10)

        self.image_input = tk.Label(self,)
        self.image_input.pack()

        self.result = tk.Label(self)
        self.result.pack(pady=15)

        self.feature = Feature()
        self.data = None
        self.load_data()
```

```
    def load_data(self):
        with open('data.json', 'r',encoding='utf-8') as f:
            data = json.load(f)
            self.data = data['data']

    def open_file(self):
        file = filedialog.askopenfile(initialdir=os.path.abspath(os.getcwd() +
'/Flower/test'), title="Select file", filetypes=(("png files", "*.png"), ("jpg files",
"*.jpg")))
        if file:
            print(file.name)
            self.label.configure(text="Hình ảnh đầu vào")
            self.image = tk.PhotoImage(file=file.name)
            self.image_input.config(image=self.image)

            self.predict(file.name)

    def predict(self, image_file):
        image = Image.open(image_file)
        image = np.array(image)

        color_feature = self.feature.color_histogram(image)
        hog_feature = self.feature.hog(image)

        list_distance = []
        list_color_distance = []
        list_hog_distance = []
        for data in self.data:
            label = data['label']
            color_feature2 = np.array(data['color_feature'])
            hog_feature2 = np.array(data['hog_feature'])

            distance_color = self.feature.distanceEuclidean(color_feature,
color_feature2)
            distance_hog = self.feature.distanceEuclidean(hog_feature,
hog_feature2)
            distance = distance_color + distance_hog
            list_distance.append((label, distance))
```

```
            list_color_distance.append(distance_color)
            list_hog_distance.append(distance_hog)


        list_distance.sort(key=lambda x: x[1])
        print(list_distance[4][1])


        if list_distance[5][1] > 21:
            self.result.configure(text="Không xác định được loại hoa")
            return

        list_distance = list_distance[:5]
        label_count = {}
        for label, distance in list_distance:
            if label in label_count:
                label_count[label]['count'] += 1
                if distance < label_count[label]['distance']:
                    label_count[label]['distance'] = distance
            else:
                label_count[label] = {
                    'count': 1,
                    'distance': distance
                }

        label_count = sorted(label_count.items(), key=lambda x: (-
x[1]['count'],x[1]['distance']))

        most_common_label = label_count[0][0]

        self.result.configure(text="Kết quả dự đoán: " + most_common_label)


if __name__ == "__main__":
    app = App()
    app.mainloop()
```