**TRƯỜNG ĐẠI HỌC ĐIỆN LỰC**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO MÔN HỌC**

**NGÔN NGỮ LẬP TRÌNH PYTHON**

ĐỀ TÀI:

NHẬN DIỆN KHUÔN MẶT BẰNG OPENCV

**Giảng viên hướng dẫn : CÙ VIỆT DŨNG**

**Sinh viên thực hiện : NGUYỄN ĐỨC NAM**

**Mã sinh viên : 1681310044**

**Lớp : D11CNPM**

Hà Nội, 31 tháng 05 năm 2019.

**PHIẾU CHẤM ĐIỂM**

|  |  |  |
| --- | --- | --- |
| Họ và tên | Chữ ký | Ghi chú |
|  |  |  |
|  |  |  |
|  |  |  |

**Sinh viên thực hiện:**

**Giảng viên chấm:**

|  |  |  |
| --- | --- | --- |
| Họ và tên | Chữ ký | Ghi chú |
| Giảng viên chấm 1: |  |  |
| Giảng viên chấm 2: |  |  |

**MỤC LỤC.**

[CHƯƠNG I: CƠ SỞ LÝ THUYẾT 2](#_Toc422583646)

[**I.** **Mở đầu về phương pháp phát hiện khuôn mặt Haar-like – Adaboost.** 2](#_Toc422583647)

[**II.** **Đặc trưng Haar-like.** 3](#_Toc422583648)

[**III.** **Ảnh tích hợp.** 6](#_Toc422583649)

[**IV.** **Adaboost** 8](#_Toc422583650)

[**V.** **Cascade of classifier.** 13](#_Toc422583651)

[CHƯƠNG II: ỨNG DỤNG THỰC NGHIỆM 16](#_Toc422583652)

[**I.** **Sử dụng Opencv và chương trình đơn giản phát hiện khuôn mặt bằng python** 16](#_Toc422583653)

[CHƯƠNG III: KẾT LUẬN 22](#_Toc422583654)

**MỞ ĐẦU**

Công nghệ thông tin đang được ứng dụng trong mọi lĩnh vực của cuộc sống. Với một hệ thống máy tính, chúng ta có thể làm được rất nhiều việc, tiết kiện thời gian và công sức. Điển hình như công việc nhận dạng mặt người. Ngày xưa, muốn tìm kiếm một kẻ tình nghi trong siêu thị hay sân bay, các nhân viên an ninh phải tìm kiếm trên từng màn hình camera theo dõi. Ngày nay, công việc đấy đã được làm tự động nhờ các hệ thống nhận dạng mặt người. Phát hiện mặt người trong ảnh là một phần quan trọng của hệ thống nhận dạng mặt người đó, giải quyết tốt việc phát hiên mặt người sẽ giúp tiết kiệm thời gian và nâng cao độ chính xác của việc nhận dạng khuôn mặt.

Phát hiện mặt người cũng là một bài toán nhận dạng đơn giản, hệ thống chỉ cần phân loại đối tượng đưa vào có phải mặt người hay không phải mặt người. Ở mức độ cao hơn, sau khi đã phát hiện được khuôn mặt, các khuôn mặt đó sẽ được so sánh với các khuôn mặt có trong dữ liệu để nhận dạng xem khuôn mặt đấy là của ai (thường áp dụng trong nhận dạng khuôn mặt của người nổi tiếng hoặc của tội phạm đang bị truy nã).

Với mục tiêu chính là tìm hiểu thuật toán Adaboost, các đặc trưng haar-like, mô hình Cascade of Classifiers, đồng thời áp dụng vào bài toán phát hiện mặt người trong ảnh, khóa luân được trình bày trong ba chương với bố cục như sau:

**Chương I: Cơ sở lý thuyết:** tìm hiểu phương pháp học máy Adaboost. Giới thiệu về các đặc trưng Haar-like và cách tính giá trị đặc trưng. Tiếp theo là giới thiệu về mô hình *cascade of classifiers* và cách áp dụng các công cụ trên vào bài toán phát hiện mặt người trong ảnh.

**Chương II: Ứng dụng thực nghiệm:** trên cơ sở thư viện mã nguồn mở OpenCV, xây dựng chương trình phát hiện mặt người trong ảnh và trong video, cùng với những phân tích chương trình.

**Chương III: Kết luận và hướng phát triển:** đánh giá kết quả và hướng phát triển nghiên cứu.

# CHƯƠNG I: CƠ SỞ LÝ THUYẾT

1. **Mở đầu về phương pháp phát hiện khuôn mặt Haar-like – Adaboost.**
2. Tổng quan.

Có rất nhiều phương pháp để giải quyết bài toán xác định khuôn mặt người trên ảnh 2D dựa trên các hướng tiếp cận khác nhau. Phương pháp Haar-like – Adaboost(viết tắt HA) của hai tác giả Paul Viola và Michael J.Jones là phương pháp xác định mặt người dựa theo hướng tiếp cận trên diện mạo.

Hướng tiếp cận trên diện mạo tức là hệ thống nhận dạng khuôn mặt sẽ được học các mẫu của khuôn mặt từ một tập ảnh mẫu. Sau khi quá trình học hay quá trình huấn luyện này (training) thực hiện xong, hệ thống sẽ rút ra được những tham số để phục vụ cho quá trình nhận dạng.Vì vậy tiếp cận trên diện mạo còn được biết đến với tên gọi tiếp cận theo phương pháp học máy. Bài báo cáo tập trung chủ yếu vào quá trình nhận dạng(sau khi đã thực hiện quá trình học).

Về tổng quan, phương pháp HA được xây dựng dựa trên sự kết hợp, lắp ghép của 4 thành phần, đó là:

* Các đặc trưng Haar-like: các đặc trưng được đặt vào các vùng ảnh để tính toán các giá trị của đặc trưng, từ những giá trị đặc trưng này đưa vào bộ phân loại Adaboost ta sẽ xác định được ảnh có khuôn mặt hay không.
* Ảnh tích hợp(Integral Image): thực ra đây là một công cụ giúp việc tính toán các giá trị đặc trưng Haar-like nhanh hơn.
* Adaboost(Adaptive Boost): bộ phân loại (bộ lọc) hoạt động dựa trên nguyên tắc kết hợp các bộ phân loại yếu để tạo lên bộ phân loại mạnh. Adaboost sử dụng giá trị đặc trưng Haar-like để phân loại ảnh là mặt hay không phải mặt.
* Cascade of Classifiers: bộ phân loại tầng với mỗi tâng là một bộ phân loại Adaboost, có tác dụng tăng tốc độ phân loại.

Như vậy bài toán xác định mặt người trong ảnh cũng chính là bài toán phân loại ảnh thành hai lớp mặt hoặc không phải mặt.

1. Tiền xử lý.

Một lưu ý nhỏ là phương pháp HA thực hiện trên ảnh xám (gray image). Mỗi điểm ảnh(pixel) sẽ có giá trị mức xám từ 0 đến 255(không gian màu 8 bit). Như vậy phương pháp HA sẽ không khai thác những đặc điểm về màu sắc khuôn mặt để nhận dạng song vẫn rất hiệu quả. Ảnh màu sẽ được chuyển về ảnh xám để nhận dạng, việc chuyển đổi này khá đơn giản, thực hiện bằng một hàm chuyển đổi và sử dụng chỉ với một câu lệnh trong Opencv nên báo cáo chưa đề cập tới.

Sau khi chuyển thành ảnh xám, ảnh lại tiếp tục được chuyển thành “ảnh tích hợp” (sẽ trình bày ở phần sau) và trong bước đầu tiên của quá trình nhận dạng, các đặc trưng Haar-like sẽ làm việc trực tiếp trên ảnh tích hợp.

1. **Đặc trưng Haar-like.**
2. Đối tượng nhận dạng.

Trên ảnh, vùng khuôn mặt là tập hợp các điểm ảnh có nhưng mối quan hệ khác biệt so với các vùng ảnh khác, những mối quan hệ này tạo lên các đặc trưng riêng của khuôn mặt. Tất cả khuôn mặt người đều có chung những đặc điểm sau khi đã chuyển qua ảnh xám, ví dụ như:

* Vùng hai mắt sẽ tối hơn vùng má và vùng chán, tức mức xám của vùng này cao hơn vượt trội so với hai vùng còn lại.
* Vùng giữa sống mũi cũng tối hơn vùng hai bên mũi.
* …

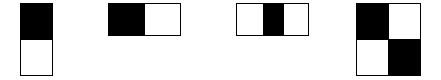
Và còn rất nhiều những đặc điểm khác của khuôn mặt và các đặc trưng Haar like dựa vào các đặc điểm này để nhận dạng.

Về tổng quát, các đặc trưng Haar like không chỉ được sử dụng để nhận dạng khuôn mặt mà có thể dùng để nhận dạng bất kì một đối tượng nào trên ảnh( thân người, tay, chân, ô tô, đồ vật,….). Bởi vì cũng giống như khuôn mặt, mỗi đối tượng có những đặc điểm riêng biệt đặc trưng bởi các vùng điểm ảnh, công việc của đặc trưng Haar-like là tính toán các giá trị tương quan giữa các vùng ảnh đó.

Đây chỉ là bước ban đầu về cách áp dụng đặc trưng để phân loại cửa sổ con, chi tiết cách phân loại của bộ Adaboost và Cascade of Classifiers sẽ trình bày ở mục sau.

1. Các đặc trưng Haar-like

Mỗi đặc trưng Haar-like là một miền hình chữ nhật được chia thành 2,3 hoặc 4 hình chữ nhật nhỏ phân biệt quy ước bằng màu trắng và đen.

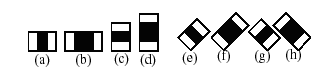


Như hình vẽ trên là 3 loại đặc trưng Haar-like cơ bản gồm 2,3 và 4 hình chữ nhật hợp thành. Từ những đặc trưng cơ bản mở rộng ra thành tập các đặc trưng:

* + Đặc trưng cạnh:



* + Đặc trưng đường:



* + Đặc trưng tâm – xung quanh:



Giá trị của một đặc trưng Haar-like là sự khác biệt giữa tổng các giá trị xám của các pixel trong vùng “đen” với tổng các giá trị xám của các pixel trong vùng “trắng”:

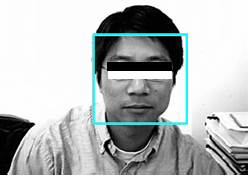
*f(x) = Tổngvùng đen(pixel) - Tổngvùng trắng(pixel) (1)*

Vậy khi được đặt lên một vùng ảnh, đặc trưng Haar-like sẽ tính toán và đưa ra giá trị đặc trưng *h(x)* của vùng ảnh đó.

1. Cách áp dụng đặc trưng Haar-like để phát hiện khuôn mặt trong ảnh.

Như đã trình bày ở bài báo cáo 1, để phát hiện khuôn mặt, 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 khuôn mặt hay không.

Hình dưới là một ví dụ: khung màu xanh là cửa sổ con, một đặc trưng Haar-like với kích thước và vị trí đặt như trong hình vẽ.



Ứng với mỗi một đặc trưng như trên, một bộ phân lớp yếu(weak classifier) hk(x) được định nghĩa như sau:

(2)

Trong đó:

X : cửa sổ con cần xét

: ngưỡng

fk: giá trị của đặc trưng Haar-like

pk: 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:*fk tại cửa sổ con *x* vượt qua một ngưỡng thì bộ phân lớp hk(x) sẽ kết luận cửa sổ con *x* là khuôn mặt(hk(x)=1), còn fk không vượt qua ngưỡng đó thì không là khuôn mặt.

Ngưỡng là giá trị đã rút ra sau qua trình huấn luyện bộ phân lớp, sẽ trình bày sau.

Như trong phần đối tượng nhận dạng, ta thấy đặc trưng Haar-like *k* có kích thước và vị trí đặt trong cửa sổ con *x* trên hình sẽ có giá trị fk rất lớn(vì mức xám ở vùng mắt lớn hơn rất nhiều vùng má ). Giá trị fk này lớn hơn nhiều với hầu hết các giá trịfk khác khi ta di chuyển cửa sổ con x sang các vị trí khác(quét trên toàn ảnh đầu vào), và với nhưỡng phù hợp, kết quả bộ phân lớp hk(x) sẽ cho cửa sổ con ở vị trí như trên là khuôn mặt còn ở vị trí khác thì không.

Đương nhiên sẽ có những vị trí khác trên ảnh đầu vào vô tình cho ra fk vượt ngưỡng và bộ phân lớp yếu hk(x) kết luận đấy là khuôn mặt, nhưng ta không chỉ dùng một đặc trưng Haar-like mà dùng rất nhiều đặc trưng ở những vị trí và kích thước khác nhau trong cửa sổ con làm giảm đi sự trùng hợp ngẫu nhiên.

1. Số lượng đặc trưng Haar-like

Việc xác định một cửa sổ con có phải là mặt hay không phải sử dụng rất nhiều đặc trưng Haar-like. Ứng với mỗi một kích thước, kiểu đặc trưng và vị trí đặt trong cửa sổ con cho ta một đặc trưng ứng với một bộ phân lớp yếu *x*. Vì vậy tập hợp đầy đủ số lượng các đặc trưng trong một cửa sổ con là rất lớn. Theo như sự tính toán của hai tác giả, với một cửa sổ con ở kích thước cơ bản 2424 pixel số lượng đầy đủ các đặc trưng Haar-like là hơn 160000.Có hai vấn đề đặt ra ở đây:

Thứ nhất là giá trị của mỗi một đặc trưng Haar-like được tính bằng tổng giá trị các điểm ảnh vùng đen trừ đi tổng các điểm ảnh vùng trắng, như vậy là với số lượng lớn các đặc trưng sẽ tạo ra một khối lượng tính toán khổng lồ. Điều này là không phù hợp để đáp ứng thời gian thực vì thời gian xử lý rất lâu.

Thứ hai là trong số hàng trăm nghìn đặc trưng đó thì không phải đặc trưng nào cũng có ích cho công việc phân loại. Nếu chúng ta không tìm cách loại bổ những đặc trưng không có ích để tập trung vào những đặc trưng có hiệu quả cao thì cũng sẽ mất thời gian xử lý một cách vô ích.

Những phần sau sẽ lần lượt tìm cách giải quyết các vấn đề này.

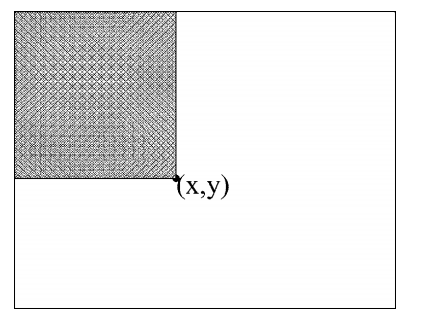
1. **Ảnh tích hợp.**
2. Định nghĩa ảnh tích hợp( Integral image).

Như đã trình bày ở phần trên, số lượng đặc trưng Haar-like là rất nhiều và khối lượng tính toán giá trị các đặc trưng này là rất lớn. Vì vậy ảnh tích hợp được đưa ra nhằm tính toán nhanh chóng các đặc trưng, giảm thời gian xử lý.

Ảnh tích hợp được định nghĩa theo công thức:

 (3)

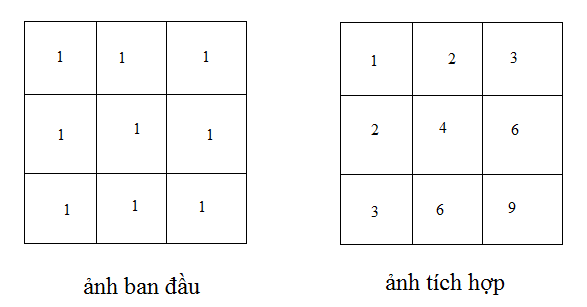
Giá trị của ảnh tích hợp tại vị trí (x,y) là tổng các điểm ảnh thuộc hình chữ nhật xác định bởi góc trái trên là (0,0) và góc phải dưới (x,y).



Trên thực tế khi chuyển một ảnh thành ảnh tích hợp ta dùng công thức truy hồi sau:

 (4)

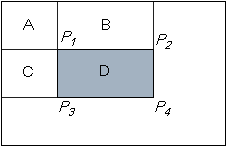
Ví dụ chuyển một ảnh 33 có giá trị xám như bên dưới thành ảnh tích hợp:



Sau khi chuyển ảnh cần nhận dạng thành ảnh tích hợp, viêc tính toán giá trị các đặc trưng Haar-like sẽ rất đơn giản.

1. Áp dụng tính toán đặc trưng Haar-like.

Để tính giá trị đặc trưng Haar-like,ta phải tính được tổng giá trị điểm ảnh trong một vùng hình chữ nhật trên ảnh. Ví dụ như vùng D trong hình vẽ:



Với :A,B,C,D là tổng giá trị các điểm ảnh trong từng vùng.

P1,P2,P3,P4 là giá trị của ảnh tích hợp tại 4 đỉnh của D.

Nếu như là ảnh xám bình thường thì để tính D ta phải tính tổng tất cả các giá trị điểm ảnh trong D, miền D càng lớn thì số phép cộng càng nhiều. Nhưng với ảnh tích hợp dù miền D có kích thước như thế nào thì D cũng chỉ cần tính thông qua 4 giá trị tại 4 đỉnh.

Ta có: P1 = A, P2 = A + B, P3 = A + C, P4 = A + B + C + D

Suy ra:

P1+ P4–P2–P3= A + (A + B + C + D) – (A + B) – (A + C) = D

Vậy ta có: **D = P1 + P4 – P2 – P3.**

Khi áp dụng vào tính toán các giá trị đặc trưng ta có thể thấy:

Đặc trưng hai hình chữ nhật(đặc trưng cạnh) được tính thông qua 6 giá trị điểm ảnh tích hợp.

Đặc trưng ba hình chữ nhật(đặc trưng đường) và đặc trưng tâm-xung quanh được tính thông qua 8 giá trị điểm ảnh tích hợp.

Đặc trưng 4 hình chữ nhật(đặc trưng chéo) được tính thông qua 9 giá trị điểm ảnh tích hợp.

Trong khi đó nếu tính như định nghĩa thì các giá trị cần tính toán lên tới hàng trăm. Điều này làm tăng tốc độ xử lý một cách đáng kể.

1. **Adaboost**
2. Khái quát về Adaboost

Kỹ thuật Boosting: nguyên tắc cơ bản của Boosting là kết hợp các bộ phân lớp yếu (hay các bộ phân lớp cơ sở) để tạo nên một bộ phân lớp mạnh. Các bộ phân lớp yếu này thậm chí chỉ cần nhỉnh hơn phương pháp random một chút. Bằng cách này, ta nói bộ phân lớp đã được “boost.”

Adaboost là một thuật toán sử dụng kỹ thuật Boosting được đưa ra bởi Freund và Schapire vào 1996. Thuật toán này được sử dụng rất phổ biến cho bài toán phát hiện khuôn mặt.

Điểm cải tiến của Adaboost là ta sẽ **gán cho mỗi mẫu một trọng số**. Ý nghĩa của việc gán trọng số như sau:

Ở mỗi vòng lặp của quá trình huấn luyện, khi một bộ phân lớp yếu yi đã được xây dựng, ta sẽ tiến hành cập nhật trọng số cho các mẫu. Việc cập nhật này được tiến hành như sau: ta sẽ tăng trọng số của các mẫu bị phân lớp sai bởi bộ phân lớp yếu yi và giảm trọng số của các mẫu được phân lớp đúng bởi yi. Bằng cách này, ở vòng lặp kế, ta sẽ xây dựng bộ phân lớp yếu yi+1 theo hướng: **tập trung vào các mẫu bị phân lớp sai bởi bộ phân lớp yếu trước đó**.

Cuối cùng, để có được bộ phân lớp mạnh, ta sẽ kết hợp tuyến tính các bộ phân lớp yếu đã tìm được lại với nhau. Mỗi bộ phân lớp yếu sẽ được đánh một trọng số tương ứng với độ tốt của bộ phân lớp yếu đó.

1. Chi tiết thuật toán.

Cho tập ảnh huấn luyện (x1, t1), …, (xn, tn) với ti { 0, 1}

1. Khởi tạo hệ số trọng cho mỗi mẫu huấn luyện: với n = 1, 2, …, N.
2. For m = 1, …, M:

(a)Xây dựng bộ phân lớp yếu hm:

+ Với mỗi đặc trưng j, xây dựng một bộ phân lớp hjvới độ lỗi:

(1.0)

với nếu và = 0 nếu ngược lại.

+ Chọn bộ phân lớp hj với độ lỗi nhỏ nhất ta được hm.

(b) Cập nhật trọng số:

+ Tính:

(1.1)

và:

(1.2)

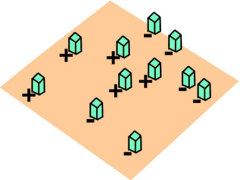
+ Cập nhật trọng số:

(1.3)

1. Bộ phân lớp mạnh cuối cùng:

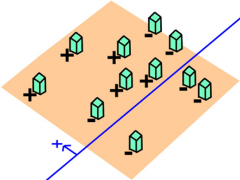
(1.4)

Thuật toán bắt đầu bằng việc khởi tạo trọng số cho các mẫu huấn luyện. Các trọng số này được khởi tạo bằng nhau. Các trọng số này cho thuật toán biết độ quan trọng của mẫu.



Ở mỗi vòng lặp, ta làm 2 việc:

* Thứ 1: tìm bộ phân lớp yếu ymdựa vào độ lỗi nhỏ nhất.

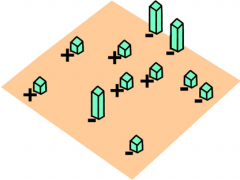
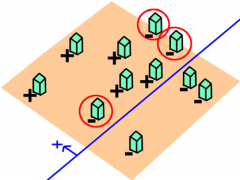


Công thức tính độ lỗi:

với nếu và = 0 nếu ngược lại.

Công thức này đơn giản chỉ là tính tổng các trọng số của các mẫu bị phân lớp sai.

* Thứ 2: cập nhật trọng số theo nguyên tắc: ta sẽ tăng trọng số cho các mẫu hiện đang bị phân lớp sai và giảm trọng số cho các mẫu hiện đang được phân lớp đúng. Bằng cách này, ở vòng lặp kế ta sẽ xây dựng bộ phân lớp yếu theo hướng tập trung giải quyết các mẫu bị phân lớp sai bởi bộ phân lớp yếu trước đó.



Trong đó:

với:

Ta nhận thấy nếu mẫu được phân lớp đúng thì trọng số không thay đổi; nếu mẫu bị phân lớp sai thì:

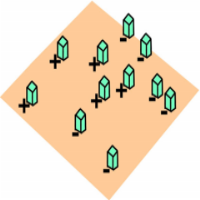
Thấy ngay

Thật vậy, giả sử , nghĩa là tỉ lệ phần sai lớn hơn phần đúng. Điều này mâu thuẫn với điều kiện bộ phân lớp yếu là tối ưu (có độ lỗi nhỏ nhất) vì rằng chỉ cần đổi chiều của mặt phẳng phân lớp lại là ta có ngay một bộ phân lớp yếu tốt hơn.

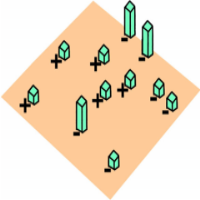
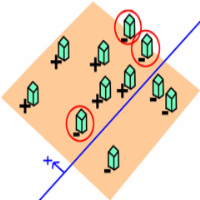
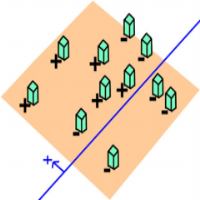
Và vì nên : trọng số của các mẫu bị phân lớp sai được tăng lên.

Để trực quan, ta hãy quan sát thuật toán thông qua chuỗi các hình vẽ dưới đây:

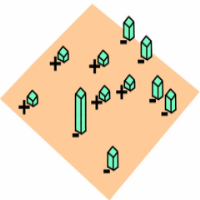
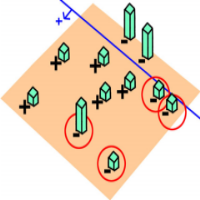
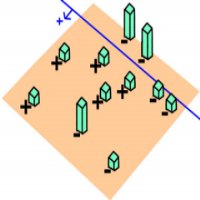
* Khởi tạo trọng số cho các mẫu:



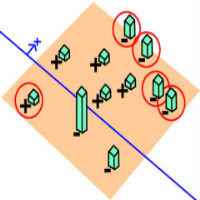
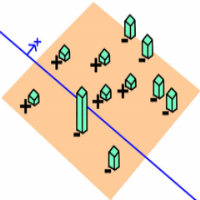
* Vòng lặp thứ 1:



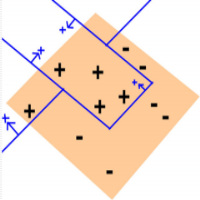
* Vòng lặp thứ 2:



* Vòng lặp thứ 3:



* Kết hợp các bộ phân lớp yếu lại:



Cuối cùng, kết hợp tuyến tính các bộ phân lớp yếu lại ta được một bộ phân lớp mạnh:

Nhận thấy mức độ đóng góp của mỗi bộ phân lớp yếu vào bộ phân lớp mạnh được quyết định bởi :

Dễ thấy tỉ lệ nghịch với mà thể hiện độ lỗi ứng với bộ phân lớp hm. Như vậy, độ lỗi càng thấp, tức bộ phân lớp càng tốt thì mức độ đóng góp của nó vào bộ phân lớp mạnh cuối cùng sẽ càng nhiều.

Nói thêm về điều kiện dừng: trong thực tế cài đặt, người ta sử dụng một giá trị ngưỡng là tỉ lệ nhận dạng sai tối đa (max false positive) để làm điều kiện dừng. Qua các vòng lặp, tỉ lệ nhận dạng sai của bộ phân lớp mạnh sẽ giảm dần. Đến một lúc nào đó, tỉ lệ này nhỏ hơn tỉ lệ nhận dạng sai tối đa và ta sẽ dừng thuật toán.

1. **Cascade of classifier.**
2. Đặt vấn đề

Sau khi đã hiểu được thuật toán Adaboost, suy nghĩ đầu tiên của ta là dùng thuật toán Adaboost để huấn luyện một bộ phân lớp mạnh. Sau đó, ta rải các cửa sổ con (sub window), chứa bộ phân lớp mạnh này, khắp một tấm ảnh đưa vào để phát hiện khuôn mặt.

*Tuy nhiên, cách làm này tốn chi phí*. Giả sử, bằng Adaboost ta có được một bộ phân lớp mạnh gồm 10 bộ phân lớp yếu. Nếu làm như trên, tại tất cả các cửa sổ con trên tấm ảnh ta đều phải dùng cả 10 bộ phân lớp yếu. Trong khi đó, ta thấy: những cửa sổ thật sự là khuôn mặt rất ít và tại những cửa sổ không phải là khuôn mặt ta có thể loại bỏ mà chỉ cần dùng một bộ phân lớp mạnh gồm ít hơn 10 bộ phân lớp yếu.

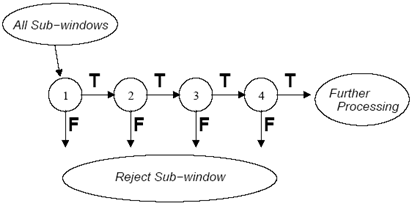
🡪Người ta giải quyết vấn đề này bằng chuỗi các bộ phân lớp (Cascade of classifier)

1. Khái quát Cascade of classifier.

Ta sẽ có một chuỗi các bộ phân lớp, trong đó mỗi bộ phân lớp được xây dựng bằng thuật toán Adaboost.

Bây giờ, ta đưa tất cả các cửa sổ con đi qua chuỗi các bộ phân lớp này:

* Bộ phân lớp đầu tiên sẽ loại bỏ phần lớn các cửa sổ không phải khuôn mặt (nagative sub window) và cho đi qua các cửa sổ được cho là khuôn mặt (positive sub window). Ở đây, bộ phân lớp này rất đơn giản và do đó, độ phức tạp tính toán cũng rất thấp. Tất nhiên, vì rằng nó đơn giản nên trong số các cửa sổ được nhận dạng là khuôn mặt sẽ có một số lượng lớn cửa sổ bị nhận dạng sai (không phải là khuôn mặt.)
* Những cửa chỗ được cho đi qua bởi bộ phân lớp đầu sẽ được xem xét bởi bộ phân lớp sau đó: nếu bộ phân lớp cho rằng đó không phải là khuôn mặt thì ta loại bỏ; nếu bộ phân lớp cho rằng đó là khuôn mặt thì ta lại cho đi qua và chuyển đến bộ phân lớp phía sau.
* Những bộ phân lớp càng về sau thì càng phức tạp hơn, đòi hỏi sự tính toán nhiều hơn. Người ta gọi những cửa sổ con (mẫu) mà bộ phân lớp không loại bỏ được là những mẫu khó nhận dạng. Những mẫu này càng đi sâu vào trong chuỗi các bộ phân lớp thì càng khó nhận dạng. Chỉ những cửa sổ đi qua được tất cả các bộ phân lớp thì ta mới quyết định đó là khuôn mặt.



Tóm lại, chuỗi các bộ phân lớp sẽ xử lý các mẫu (cửa sổ con) đi vào theo nguyên tắc sau: nếu một bộ phân lớp nào đó cho rằng đó không phải là mặt người thì ta loại bỏ ngay; còn nếu bộ phân lớp cho rằng đó là khuôn mặt thì ta chuyển đến bộ phân lớp sau. Nếu một mẫu trót lọt hết tất cả các bộ phân lớp thì ta mới quyết định đó là khuôn mặt.

# CHƯƠNG II: ỨNG DỤNG THỰC NGHIỆM

1. **Sử dụng Opencv và chương trình đơn giản phát hiện khuôn mặt bằng python**

**1.Phương pháp Haar like- Adaboost trong Opencv**

Phương pháp nhận diện khuôn mặt dựa vào đặc trưng Haar like kết hợp Adaboost được cài sẵn trong bộ thư viện Opencv. Để sử dụng phương pháp này trong Opencv ta sử dụng hàm detectMultiScale.

Phát hiện khuôn mặt trong một bức ảnh, cách thức là chương trình sẽ tạo một khung cửa sổ có kích thước xác định chứa một phần của bức ảnh, khung cửa sổ này chính là “ảnh đầu vào” mà phương pháp Haar like dùng để tính toán. Khi quét khung cửa sổ này trên toàn bộ bức ảnh ta sẽ nhận được vô số “ảnh đầu vào”, với mỗi ảnh đầu vào chương trình sẽ xác nhận xem đó có phải là mặt người hay không với những bước như trong phần I. Bằng cách thay đổi tỉ lệ và vị trí của khung cửa sổ này chúng ta có thể phát hiện được tất cả các khuôn mặt có trong bức ảnh( tất nhiên còn phụ thuộc vào độ chính xác của phương pháp).

Minh họa cho cách thức phát hiện mặt người của Opencv qua video:

https://vimeo.com/12774628

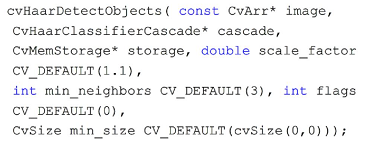
Chi tiết cách thức này được trinh bày trong mục sau.

**2. Chương trình đơn giản phát hiện khuôn mặt bằng python**

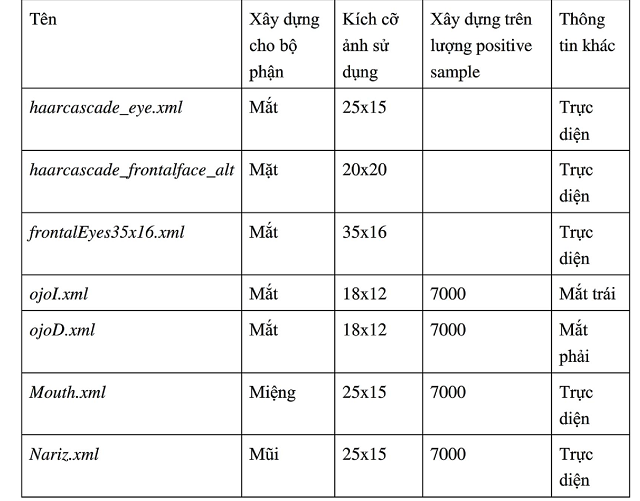
\*Đây là một chương trình phát hiện khuôn mặt đơn giản với đối tượng là một ảnh ABC.jpg:

1. import numpy as np
2. import cv2
3. face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')
4. eye\_cascade = cv2.CascadeClassifier('haarcascade\_eye.xml')
5. img = cv2.imread('ABC.jpg')
6. gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)
7. faces = face\_cascade.detectMultiScale(gray, 1.3, 5)
8. for (x,y,w,h) in faces:
9. img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
10. roi\_gray = gray[y:y+h, x:x+w]
11. roi\_color = img[y:y+h, x:x+w]
12. eyes = eye\_cascade.detectMultiScale(roi\_gray, 1.3, 5)
13. for (ex,ey,ew,eh) in eyes:
14. cv2.rectangle(roi\_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
15. cv2.imshow('ABC',img)
16. cv2.waitKey(0)
17. cv2.destroyAllWindows()

Trước khi phân tích chương trình ta tìm hiểu hàm detectMultiScale là phần tìm kiếm khuôn mặt, hàm(hay phương thức) này thuộc lớp CascadeClassifier(lớp phục vụ tìm kiếm đối tượng của Opencv), hàm có nguyên mẫu với những tham số như sau:



Tham số **cascade** chính bộ phân lớp đề cập trong phần I, bộ phân lớp này được lưu vào các file .xml được xây dựng sẵn trên các bộ cơ sở dữ liệu chuẩn. Dưới đây là một số bộ phân loại trong thư viện Opencv.



Như trong bảng khi cần phát hiện các đối tượng khác nhau ta sẽ sử dụng các bộ phân lớp khác nhau, tức là load các file .xml tương ứng vào chương trình. Opencv cũng hỗ trợ chúng ta tự xây dựng một bộ phân lớp riêng cho các đối tượng khác bằng hàm cvCreateHidHaarClassifierCascade, chúng ta có thể xây dưng các bộ phân loại để phát hiện các đối tượng khác như ôtô, chó, mèo….

Quay lại hàm detectMultiScale, tham số tiếp theo là **image** chính là bức ảnh cần phát hiện khuôn mặt.

**scale\_factor** là tỉ lệ tăng kích thước của khung cửa sổ tìm kiếm. Ví dụ nếu scale\_factor=1.1 thì sau khi quét hết bức ảnh 1 lần, khung cửa sổ sẽ tăng kich thước lên 10% 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 khuôn mặt 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 khuôn mặt nào nhưng chương trình sẽ tốn thời gian hơn vì tăng số lần quét lên.

**min\_neighbors** giá trị tối thiểusố hình chữ nhật lân cận được gộp lại sau khi quá trình quét đã xong, tham số này sẽ được giải thích rõ hơn trong phần sau.

**min\_size** là kích thước nhỏ nhất của khung cửa sổ phát hiện khuôn mặt, khung sẽ bắt đầu quét từ kích thước này, mọi khuôn mặt có diện tích vượt qua kích thước khung này đều không được phát hiện.

Đó những tham số quan trọng của hàm.

Chuyển sang phân tích chương trình:

Dòng 1,2 là phần khai báo chương trình sẽ sử dụng 2 bộ thư viện là Numpy và Opencv bản 2.x

Dòng 3,4 ta tạo ra 2 đối tượng thuộc lớp CascadeClassifier là face\_cascade và eye\_cascade để tìm kiếm mặt và mắt, đồng thời load 2 flie .xml cho 2 đối tượng này.

Dòng 5 đọc ảnh cần tìm khuôn mặt vào biến img.

Dòng 6 tạo một biến ảnh gray là ảnh vào chuyển sang dạng xám để thực hiện tìm kiếm.

Dòng 7 sử dụng hàm detectMultiScale để tìm kiếm các khuôn mặt trong ảnh xám gray với các tham số **scale\_factor=1.3** và **min\_neighbors=5** bằng bộ phân lớp của face\_cascade.

Dòng 8,9 hàm detectMultiScale sau khi tìm kiếm xong sẽ trả về bộ giá trị gồm tọa độ gốc của khung chứa khuôn mặt x,y ; chiều dài, rộng của khung w,h. Các giá trị này nằm trong mảng faces. Cấu trúc for….in sẽ duyệt qua toàn bộ các bộ giá trị này, với mỗi bộ giá trị ta dùng hàm rectangle để vẽ một hình chữ nhật lên ảnh ban đầu img với tọa độ 2 điểm trái trên và phải dưới: (x,y),(x+w,y+h). (0,255,0) là màu sẽ vẽ hình chữ nhật.

Như vậy là đã xác định xong các khuôn mặt có trong ảnh.

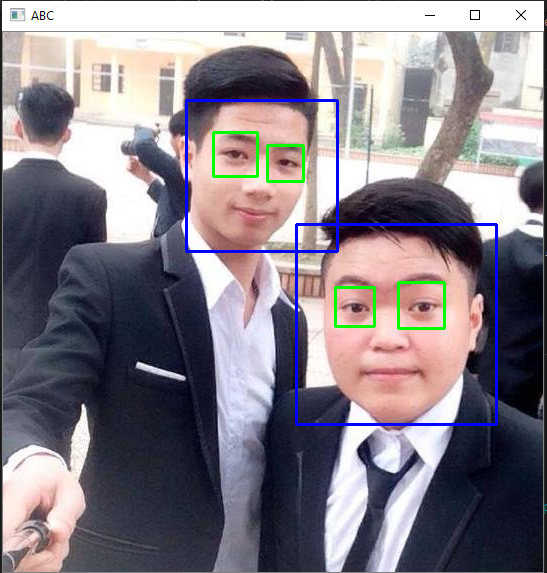
Dòng 10,11 ta tạo ra roi\_gray = gray[y:y+h, x:x+w] và roi\_color = img[y:y+h, x:x+w] là 2 biến ảnh màu và ảnh xám các khuôn mặt đã tìm thấy. Và việc tìm kiếm mắt được thực hiện trên các khuôn mặt đã tìm thấy chứ không phải quét toàn bộ ảnh ban đầu như tìm kiếm mặt.

Dòng 12,13,14 tương tự là tìm kiếm và vẽ khung hình chữ nhật chứa mắt với một màu khác.

Dòng 15 dùng hàm hiển thị ảnh đã vẽ xong lên một cửa sổ window có tên ABC

Dòng 16,17 đợi ấn nút bất kì để thoát chương trình, xóa cửa sổ hiển thị.

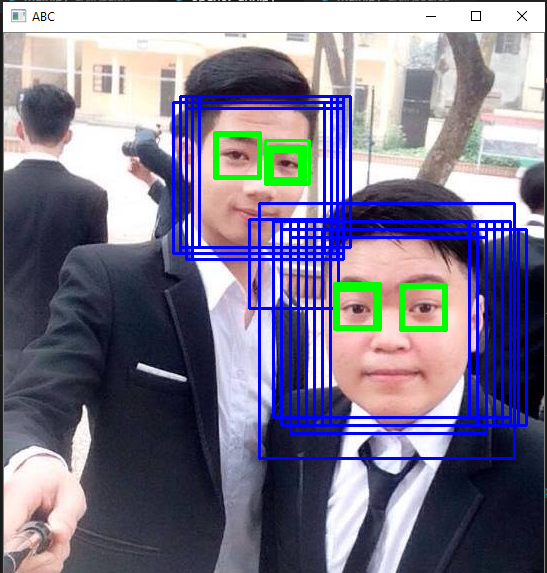
Kết quả chương trình:



Quay lại với tham số **min\_neighbors:** 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 khuôn mặt chương trình sẽ tìm được nhiều những khung hình chữ nhật chứa khuôn mặt cho dù đó chỉ là một khuôn mặt và có nhưng trường hợp nhận diện nhầm.

Nếu ta để tham số min\_neighbors=0 cho hàm tìm khuôn mặt tức là để nguyên những gì tìm được sau khi quét thì sẽ được kết quả:



Như vậy tham số này sẽ gộp lại những hình chữ nhật chứa cùng một khuôn mặt để chỉ cho ra một hình chữ nhật cho 1 khuôn mặt. Đồng thời nó sẽ loại bỏ bớt đi những kết quả sai vì những kết quả sai chỉ cho ra 1 khung hình chữ nhật còn kết quả đúng thì có nhiều khung hình lân cận bao lấy. Ví dụ như những hình chữ nhật nhỏ trên tường như trong hình trên.

Việc phát hiện khuôn mặt trên video hay camera cũng tương tự như trên một bức ảnh, vì video là chuỗi liên tiếp những hình ảnh được hiển thị (frame hình). Ta làm một vòng lặp để thực hiện phát hiện khuôn mặt trên những frame hình liên tiếp và cũng đưa kết quả ra liên tiếp.

Dưới đây là một chương trình phát hiện khuôn mặt lấy đối tượng là hình ảnh từ camera máy tính và hiển thị:

import numpy as np

import cv2

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

video\_capture = cv2.VideoCapture(0)

while True:

q ret, frame = video\_capture.read()

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

faces =face\_cascade.detectMultiScale(gray, 1.3, 5)

for (x,y,w,h) in faces:

cv2.rectangle(frame, (x,y),(x+w,y+h),(255,0,0),2)

cv2.imshow('video',frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

video\_capture.release()

cv2.destroyAllWindows()

Trong chương trình này, lớp video\_capture = cv2.VideoCapture(0) là lớp xử lý những frame hình lấy từ webcam mặc định của máy tính.

Trong mỗi một vòng lặp, thực hiện lấy ảnh đầu vào là frame hình từ webcam bằng hàm frame = video\_capture.read() . Sau đó mọi việc phát hiện khuôn mặt trên frame hình này thực hiện tương tự như trên một ảnh bình thường ở chương trình đầu tiên. Sau khi kết quả được hiển thị, chương trình tiếp tục nhận frame hình tiếp theo, phát hiện rồi hiển thị.Vòng lặp được thực hiện liên tục cho ra một video với khuôn mặt được phát hiện.

# CHƯƠNG III: KẾT LUẬN

A, Kết quả.

Trên cơ sở nghiên cứu về bài toán phát hiện mặt người trong ảnh. Đặc biệt là thuật toán Adaboost, các đặc trưng *haar-like* và mô hình *Cascade of Classifier*, áp dụng thành công mô hình *Cascade of Classfier* vào bài toán phát hiện mặt người trong ảnh. Tuy kết quả đạt được chưa cao, nhưng đây là bước đầu để phát triển bài toán nhận dạng mặt (*face recognition*) người sau này.

B, Hướng phát triển.

Có nhiều hướng phát triển cho chương trình này, có thể phát triển cả về mặt ứng dụng và mặt thuật toán (để cải thiện hiệu quả phát hiện mặt người). Có thể xây dựng một ứng dụng chỉ cần đến phát hiện mặt người mà không cần nhận dạng. Ví dụ như một hệ thống ghép hình, ghép khuôn mặt phát hiện được vào trong một bức ảnh khác (chẳng hạn như ghép khuôn mặt của người sử dụng cho khuôn mặt của người nổi tiếng).

Ngoài ra có thể phát triển chương trình theo hướng nhận dạng khuôn mặt, xây dựng một hệ thống để học các đặc trưng của những người cần nhận dạng. Khi thực hiện, đầu tiên ta đưa qua bức ảnh qua chương trình phát hiện mặt người để phát hiện nhanh các khuôn mặt có trong ảnh, sau đấy so sách các khuôn mặt đó với các khuôn mặt mà chương trình đã được “học” từ trước, so sánh các đặc trưng của hai khuôn mặt, nếu trùng thì đưa ra thông tin về khuôn mặt được nhận dạng.