

Đại học Bách Khoa Hà Nội  
Viện Công nghệ thông tin và Truyền thông

# **Nghiên cứu tốt nghiệp 1**

**Đề tài : Nhận diện gương mặt**

Sinh viên thực hiện : Vũ Công Duy

MSSV : 20150636

Lớp : Việt Nhật AS - K60

Giáo viên hướng dẫn : **PGS.TS. Phạm Văn Hải**

# Mục lục

Mở đầu	3
Chương I : Mô tả bài toán	4
1. Tổng quan:	4
2. Môi trường và mẫu thử	4
Chương II : Phát hiện gương mặt xuất hiện trong ảnh	5
1. Tổng quan:	5
2. Tiền xử lý:	5
3. Đặc trưng Haar-like:	6
a. Đối tượng nhận dạng:	6
b. Các đặc trưng Haar-like	6
c. Áp dụng đặc trưng Haar-like trong nhận dạng gương mặt	7
4. Ảnh tích hợp (integral image):	7
5. Thuật toán Adaboost	8
a. Cơ sở lý thuyết	8
b. Chi tiết thuật toán	9
6. Cascade of classifiers	11
a. Đặt vấn đề	11
b. Khái quát	11
Chương III : Phân loại và nhận dạng gương mặt	12
Chương IV : Thực nghiệm và đánh giá kết quả	12
1. Face detection	12
a. Source code	12
b. Thực nghiệm	13
2. Face recognizer	14
a. Cấu trúc thư mục	14
b. Source code	15
b. Thực nghiệm	18
Source code :	18
Reference	18

## 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 kiệm thời gian và công sức. Điển hình như công việc nhận dạng mặt người

Nhận dạng mặt người (Face recognition) là một lĩnh vực nghiên cứu của ngành Computer Vision, và cũng được xem là một lĩnh vực nghiên cứu của ngành Biometrics (tương tự như nhận dạng vân tay – Fingerprint recognition, hay nhận dạng mống mắt – Iris recognition).

Việc xây dựng một hệ thống nhận dạng gương mặt chia ra thành 2 bài toán:

1. Phát hiện gương mặt xuất hiện trong ảnh (**face detect**)
2. Phân loại và nhận dạng gương mặt (**classification face**)

Với mục tiêu xây dựng ứng dụng nhận diện gương mặt, đề án được trình bày trong ba chương với bố cục như sau:

**Chương I : Mô tả bài toán:** đặc tả yêu cầu bài toán, dữ liệu đầu vào, đầu ra, công cụ và môi trường phát triển

**Chương II : Phát hiện gương mặt xuất hiện trong ảnh:** 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 III: Phân loại và nhận dạng gương mặt:** tìm hiểu bộ mô tả kết cấu (texture descriptor) Local Binary Patterns (LBP) và ứng dụng vào bài toán phân loại gương mặt

**Chương IV: Thực nghiệm và đánh giá kết quả:** trên cơ sở thư viện mã nguồn mở OpenCV, xây dựng chương trình phát hiện mặt người trong ảnh

## Chương I : Mô tả bài toán

### 1. Tổng quan:

Xây dựng ứng dụng phát hiện gương mặt một số ca sỹ nổi tiếng của Việt Nam như : Đàm Vĩnh Hưng, Đan Trường, Mỹ Tâm, Hà Anh Tuấn, Hương Tràm.

### 2. Môi trường và mẫu thử

Đầu vào :



Đầu ra :



<b>Dataset</b>	<b>:</b>	<b>Training</b>	: 690 images (96 x 96px)
		<b>Testing</b>	: 301 images (96 x 96px)

**Môi trường** : Hệ thống nhận dạng này được cài đặt bằng ngôn ngữ lập trình Python 3.5, sử dụng thư viện mã nguồn mở OpenCV 4.0 trên một máy tính cá nhân chạy hệ điều hành Linux.

## Chương II : Phát hiện gương mặt xuất hiện trong ảnh

### 1. 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) được xây dựng bởi hai tác giả Paul Viola và Michael J. Jones.

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

#### 1. 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.

#### 2. Ảnh tích hợp (Integral Image):

Chuyển cách mã hóa từ dạng ảnh thông thường sang ảnh tích hợp giúp việc tính toán các giá trị đặc trưng Haar-like nhanh hơn.

#### 3. 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.

#### 4. Cascade of Classifiers:

Bộ phân loại tầng với mỗi tầng là một bộ phân loại Adaboost, có tác dụng tăng tốc độ phân loại.

### 2. Tiền xử lý:

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.

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.

### 3. Đặc trưng Haar-like:

#### a. Đối tượng nhận dạng:

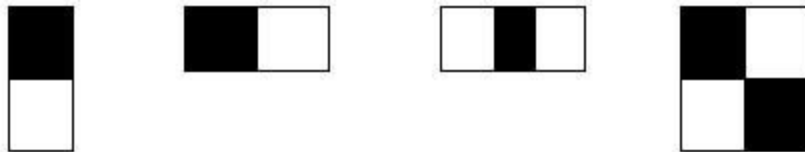
Trên ảnh, **vùng khuôn mặt** là tập hợp các điểm ảnh có những mối quan hệ khác biệt so với các vùng ảnh khác, những mối quan hệ này tạo lên các đặc trưng riêng của khuôn mặt. Tất cả khuôn mặt người đều có chung những đặc điểm sau khi đã chuyển qua ảnh xám, ví dụ như:

- Vùng hai mắt sẽ tối hơn vùng má và vùng trán, 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.

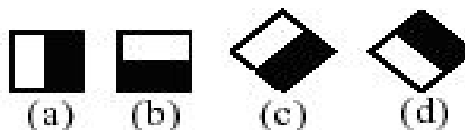
#### b. Các đặc trưng Haar-like

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. Hình dưới đây mô tả 4 đặc trưng Haar Like cơ bản như sau:

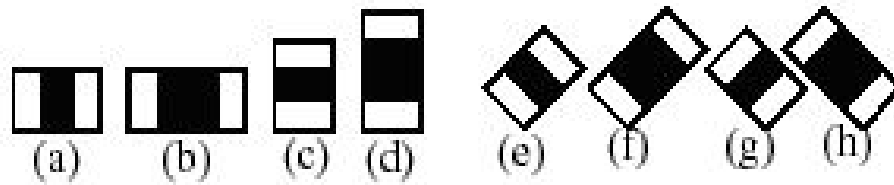


Từ những đặc trưng cơ bản mở rộng ra thành tập các đặc trưng:

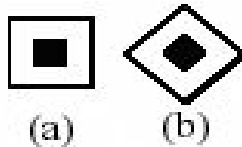
**Đặc trưng cạnh:**



**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”:

$$f(x) = \text{Tổng}_{\text{vùng đen}}(\text{pixel}) - \text{Tổng}_{\text{vùng trắng}}(\text{pixel})$$

### c. Áp dụng đặc trưng Haar-like trong nhận dạng gương mặt

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 đặt 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.

Như vậy, giả sử với image 24x24 pixel, bộ đầy đủ các sub-window cần kiểm tra là khá lớn, hơn 180000. Do đó Viola và Jones đưa ra một khái niệm gọi là **Integral Image**.

## 4. Ảnh tích hợp (integral image):

Là 1 mảng 2 chiều cho phép tính toán nhanh giá trị **chênh lệch giữa tổng các điểm ảnh** trong khu vực đang xét. Từ đó giảm lượng phép tính toán từ  $O(n^2)$  xuống  $O(1)$ .

Với mỗi phần tử của mảng này được tính bằng cách tính tổng của điểm ảnh phía trên (dòng-1) và bên trái (cột-1) của nó

**Ví dụ :**

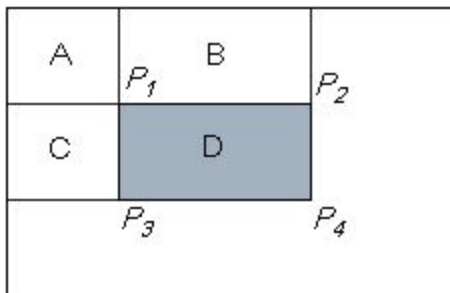
0.1	0.1	0.2	0.1	0.7	0.1
0.2	0.3	0.2	0.7	0.8	0.2
0.1	0.4	0.3	0.3	0.1	0.3
0.1	0.5	0.1	0.1	0.2	0.8
0.1	0.4	0.8	0.5	0.6	0.5

original image

→

0.1	0.2	0.4	0.5	1.2	1.3
0.3	0.7	1.1	1.9	3.4	3.7
0.4	1.2	1.9	3.0	4.6	5.2
0.5	1.7	2.5	3.7	5.3	6.7
0.6	2.3	3.9	5.6	8.0	9.9

integral image



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.

Chênh lệch giữa tổng các điểm ảnh trong hình chữ nhật D có thể tính nhanh bằng:  $P4 + P1 - (P2 + P3)$

- Ví dụ

0.1	0.1	0.2	0.1	0.7	0.1
0.2	0.3	0.2	0.7	0.8	0.2
0.1	0.4	0.3	0.3	0.1	0.3
0.1	0.5	0.1	0.1	0.2	0.8
0.1	0.4	0.8	0.5	0.6	0.5

original image

→

0.1	0.2	0.4	0.5	1.2	1.3
0.3	0.7	1.1	1.9	3.4	3.7
0.4	1.2	1.9	3.0	4.6	5.2
0.5	1.7	2.5	3.7	5.3	6.7
0.6	2.3	3.9	5.6	8.0	9.9

integral image

$$\text{SUM} = 3.7 - 0.5 + 0.2 - 1.7 = 1.7$$



## 5. Thuật toán Adaboost

### a. Cơ sở lý thuyết

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. Trong bài toán face recognition, ta sẽ sử dụng thuật toán Adaboost

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  $h_t$  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 và giảm trọng số của các mẫu được phân lớp đúng bởi  $y_i$ . Bằng cách này, ở vòng lặp kế, ta sẽ xây dựng bộ phân lớp yếu 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.

### b. Chi tiết thuật toán

**Bước 1:** Cho tập huấn luyện  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$  với  $y_i = 0, 1$  tương ứng nhãn  $P_-, P_+$

Khởi tạo trọng số (weights)  $w_{t,i}$  ( $t = 1$ ) cho các mẫu huấn luyện:

$$w_{1,i} = \frac{1}{2M}, \frac{1}{2L}$$

với  $M, L$  là tổng số nhãn  $P_-$ , và  $P_+$  trong tập huấn luyện

Từ cách khởi tạo trên, ta nhận thấy  $\sum_{i=1}^n w_{t,i} = 1$  và luôn được duy trì trong quá trình huấn luyện nhờ thực hiện **Normalize weights** cho mỗi lần huấn luyện  $t$

**Bước 2:** Tiến hành huấn luyện For :  $t = 1, 2, \dots, T$

#### 2.1 Cân bằng trọng số (Normalize weights)

$$w_{t, i} = \frac{w_{t, i}}{\sum_{j=1}^n w_{t, j}}$$

**2.2** Với mỗi đặc trưng  $j$ , huấn luyện các bộ phân lớp yếu (**weak classifiers**)  $h_j$ . Lỗi của mỗi bộ phân lớp  $h_j$  sinh ra được tính toán theo công thức:

$$e_j = \sum_{i=1}^n w_i |h_j(x_i) - y_i|$$

**2.3** Chọn bộ phân lớp  $h_j$  với giá trị lỗi  $e_j$  nhỏ nhất

**2.4** Dựa vào kết quả từ bộ phân lớp  $h_j$  đã chọn, tiến hành **tăng trọng số các mẫu huấn luyện dự đoán sai** và **giảm trọng số các mẫu huấn**

**luyện dự đoán đúng theo cơ số** :  $\beta_t = \frac{e_j}{1-e_j}$

$$w_{t+1, i} = w_{t, i} * \beta_t^{1-\mu}$$

với  $\mu = 0$  nếu  $h_j$  phân loại đúng  $h_j(x_i) = y_i$

$\mu = 1$  nếu ngược lại

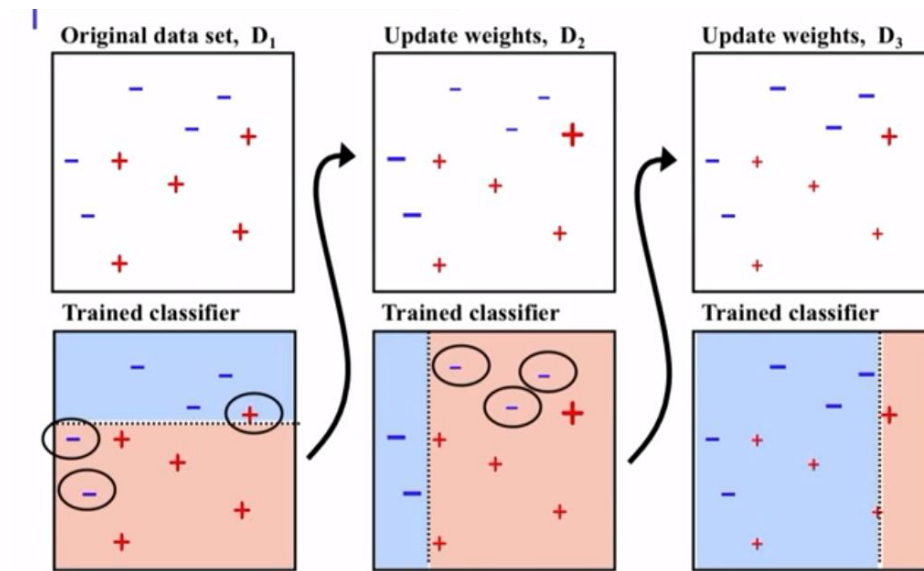
**Nhận xét** : việc **tăng trọng số các mẫu huấn luyện sai** nhằm tập trung vào phân loại những mẫu này trong lần huấn luyện kế tiếp. Bởi vì các lớp phân loại  $h_j$  nếu phân loại sai ở những mẫu thử có trọng số  $w_i$  lớn dễ làm tăng giá trị lỗi  $e_j$  và ko được chọn vào tập các lớp phân loại tốt nhất.

**Bước 3:** Lớp phân loại tổng hợp (**strong classifiers**)

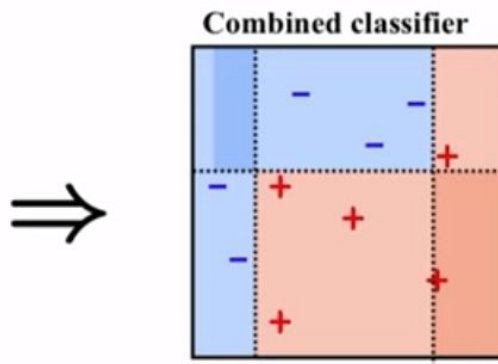
$$h(x) = 1 \text{ nếu } \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \text{ và } 0 \text{ nếu ngược lại}$$

$$\text{với } \alpha_t = \log \log \frac{1}{\beta_t}$$

**Ví dụ:**



## 6. Cascade of classifiers



### a. Đặt vấn đề

Áp dụng thuật toán Adaboost vào phân loại gương mặt, ta tiến hành sử dụng bộ phân lớp mạnh cho toàn bộ sub-window trên ảnh. Cách làm này gây tốn chi phí rất lớn.

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.

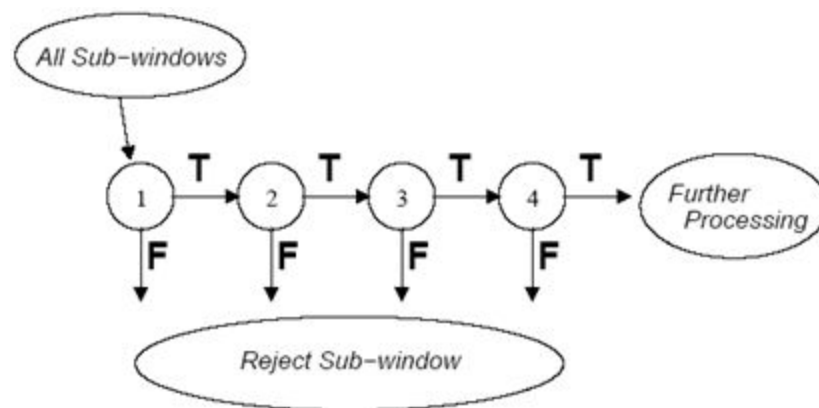
## b. Khái quát

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

Bộ phân lớp đầu tiên sẽ loại bỏ phần lớn các cửa sổ không phải khuôn mặt (negative 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 sổ 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.



## Chương III : Phân loại và nhận dạng gương mặt

## Chương IV : Thực nghiệm và đánh giá kết quả

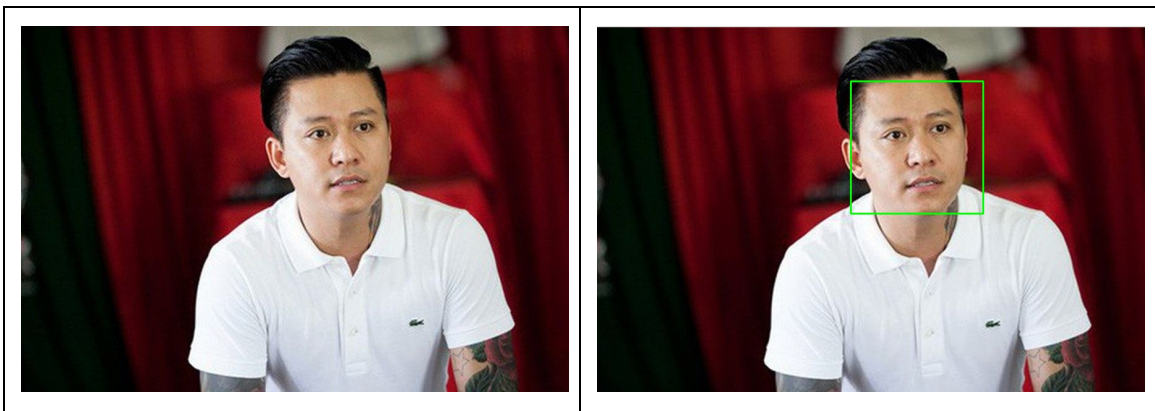
### 1. Face detection

#### a. Source code

```
import numpy as np
import cv2
Cac1 = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

input = cv2.imread("input_img1.png")
gray = cv2.cvtColor(input, cv2.COLOR_BGR2GRAY)
faces = Cac1.detectMultiScale(gray, 1.1, 5)
for (x,y,w,h) in faces:
    output = cv2.rectangle(input,(x,y),(x+w,y+h),(0,255,0),2)
cv2.imshow("output",output)
cv2.waitKey(0)
cv2.imwrite("output_img1.png", output)
cv2.destroyAllWindows()
```

#### b. Thực nghiệm





#### a. Phân tích source code

Sử dụng lớp CascadeClassifier (lớp phục vụ tìm kiếm đối tượng của Opencv),

**Nguyên mẫu hàm:** `CascadeClassifier` (const `String` &filename)

Tham số **filename** chính bộ phân lớp 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.

Tên	Xây dựng cho bộ phân	Kích cỡ ảnh sử dụng	Xây dựng trên lượng positive sample	Thông tin khác
<i>haarcascade_eye.xml</i>	Mắt	25x15		Trực diện
<i>haarcascade_frontalface_alt</i>	Mặt	20x20		Trực diện
<i>frontalEyes35x16.xml</i>	Mắt	35x16		Trực diện
<i>ojoL.xml</i>	Mắt	18x12	7000	Mắt trái
<i>ojoD.xml</i>	Mắt	18x12	7000	Mắt phải
<i>Mouth.xml</i>	Miệng	25x15	7000	Trực diện
<i>Nariz.xml</i>	Mũi	25x15	7000	Trực diện

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.

Hàm **detectMultiScale** để detect được các gương mặt xuất hiện trong ảnh

**Nguyên mẫu hàm :** `detectMultiScale (InputArray image, std::vector< Rect > &objects, double scaleFactor=1.1, int minNeighbors=3, int flags=0, Size minSize=Size(), Size maxSize=Size())`

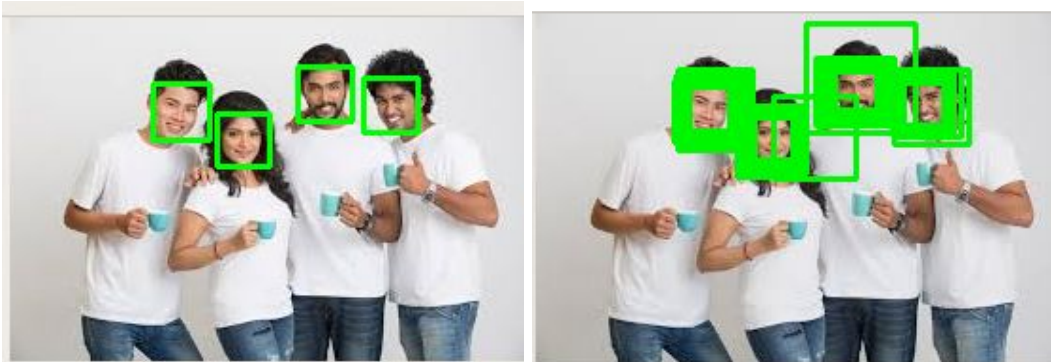
**image** là bức ảnh đầu vào

**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 kích 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 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 những 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:

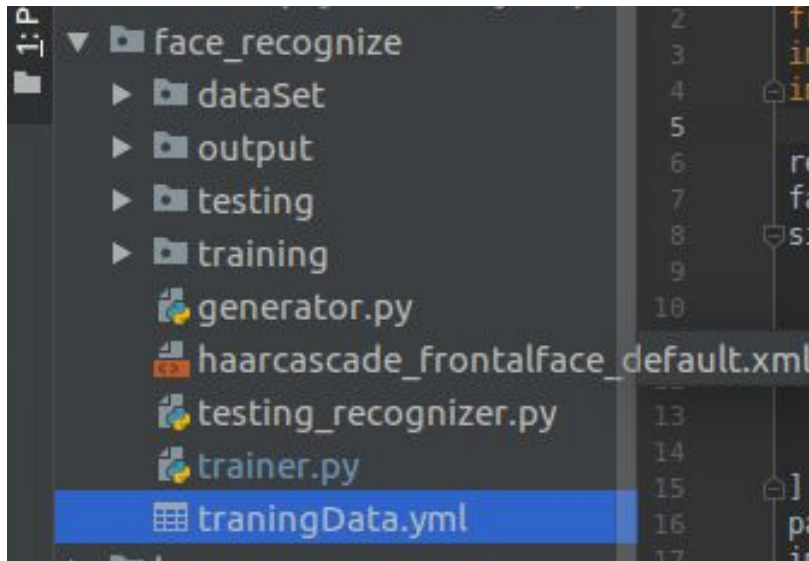
**Ví dụ :**



**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.

## 2. Face recognizer

### a. Cấu trúc thư mục



dataSet.py	tập ảnh một số ca sỹ Việt Nam
generator.py	chia dataSet vào folder training, testing (7:3)
haarcascade_frontalface_default.xml	bộ phân lớp nhận diện gương mặt trực diện
trainer.py	xây dựng model nhận diện mặt trainingData.yml
testing_recognizer	áp dụng model kiểm thử bộ test
trainingData.yml	
training	tập ảnh huấn luyện
testing	tập ảnh kiểm thử
output	tập ảnh được gắn thêm nhãn



## b. Source code

```
-----trainer.py-----  
  
import glob, os, shutil  
from PIL import Image  
import numpy as np  
import cv2  
  
recognizer = cv2.face.LBPHFaceRecognizer_create()  
faceCascade =  
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')  
singers = [ 'dam vinh hung', 'dan truong', 'ha anh tuan', 'my tam',  
            'huong tram', 'son tung' ]  
path ='training'  
imagePaths = [os.path.join(path, f) for f in os.listdir(path)]  
  
faces = []  
IDs = []  
def get_images_and_labels(path):  
    for imagePath in imagePaths:  
        faceImg = Image.open(imagePath).convert('L')  
        faceNp = np.array(faceImg, 'uint8')  
        singer = os.path.split(imagePath)[-1].split('.')[0]  
        ID = singers.index(singer) + 1  
        temp = faceCascade.detectMultiScale(faceNp, 1.1, 5)  
        for (x, y, w, h) in faceCascade.detectMultiScale(faceNp):  
            faces.append(faceNp[y:y + h, x:x + w])  
            IDs.append(ID)  
  
        cv2.imshow('Traning', faceNp)  
        cv2.waitKey(50)  
  
    return np.array(IDs), faces
```

```
IDs, faces = get_images_and_labels(path)
recognizer.train(faces, np.array(IDs))
recognizer.save('trainingData.yml')
cv2.destroyAllWindows()
```

-----  
-----testing\_recognizer.py-----

```
import os
import cv2
import numpy as np

faceDetect = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainingData.yml')

path = testing/
correct = 0
fail = 0
unknown = 0

singers = ['dam vinh hung', 'dan truong', 'ha anh tuan', 'my tam',
           'huong tram', 'son tung' ]

for filename in os.listdir(path):
    img_result = cv2.imread(path + filename)
    img = cv2.imread(path + filename, 0)
    faces = faceDetect.detectMultiScale(img, 1.1, 5)
    id = "unknown"
    for (x, y, w, h) in faces:
```

```

cv2.rectangle(img_result, (x, y), (x + w, y + h), (0, 0, 255), 2)
id, conf = recognizer.predict(img[y:y+h, x:x+w])
id = singers[id - 1]
if (conf < 150):
    if (id in filename):
        correct += 1
    else:
        fail += 1
    cv2.putText(img_result, id + ' - ' + filename, (x, h),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), lineType=cv2.LINE_AA)

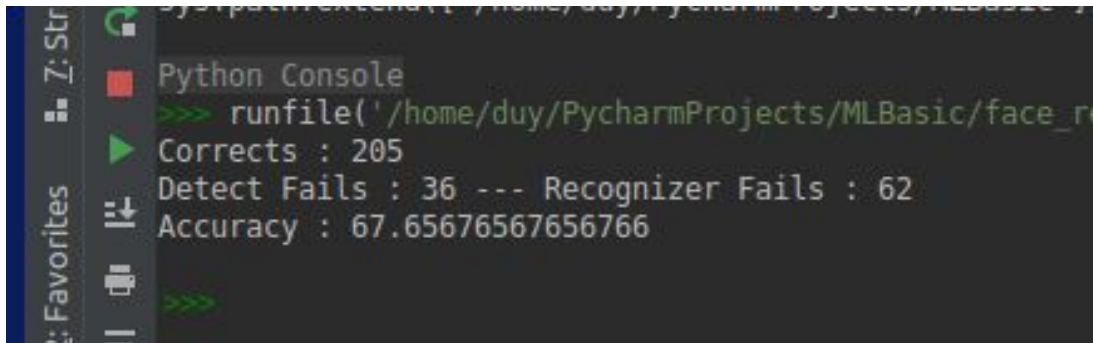
if (id == "unknown"):
    cv2.putText(img_result, id + ' - ' + filename, (x, h),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), lineType=cv2.LINE_AA)
    unknown += 1
cv2.imshow('Testing', img_result)
cv2.imwrite("output/" + filename, img_result)
cv2.waitKey(100)
if (cv2.waitKey(1) == ord('q')):
    break;

cv2.destroyAllWindows()

print ('Corrects : ' + str(correct))
print ('Detect Fails : ' + str(unknown))
print (' Recognizer Fails : ' + str(fail))
print ("Accuracy : " + str(correct/(correct+unknown+fail)*100))

```

## b. Thực nghiệm



```
Python Console
>>> runfile('/home/duy/PycharmProjects/MLBasic/face_re
Corrects : 205
Detect Fails : 36 --- Recognizer Fails : 62
Accuracy : 67.65676567656766
>>>
```

Hiệu suất đạt được : 67.66%

Ca sỹ	Hiệu suất
2	80.23255813953489
3	
4	

## Source code :

<https://github.com/vucongduy192/face-recognition>

## Reference

[1] Rapid Object Detection using a Boosted Cascade of Simple Features By Paul Viola and Michael Jones