ĐẠI HỌC THÁI NGUYÊN
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐÀO QUANG TOÀN

ĐỀ TÀI
NHẬN DẠNG KHUÔN MẶT TRONG HỖ TRỢ CÔNG TÁC QUẢN LÝ TIẾP DÂN

LUẬN VĂN THẠC SĨ KHOA HỌC MÁY TÍNH

THÁI NGUYÊN, 2017

ĐẠI HỌC THÁI NGUYÊN
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐÀO QUANG TOÀN

NHẬN DẠNG KHUÔN MẶT TRONG HỖ TRỢ CÔNG TÁC QUẢN LÝ TIẾP DÂN

Chuyên ngành: Khoa học máy tính

Mã số: 60480101

LUẬN VĂN THẠC SĨ KHOA HỌC MÁY TÍNH

Người hướng dẫn khoa học: TS. VŨ DUY LINH

THÁI NGUYÊN, 2017

LỜI CẢM ƠN

Sau một thời gian tìm hiểu dưới sự giúp đỡ của giáo viên hướng dẫn TS. Vũ Duy Linh, tôi đã hoàn thành đề tài luận văn "Nhận dạng khuôn mặt trong hỗ trợ công tác quản lý tiếp dân".

Tôi xin bày tỏ lòng biết ơn đến:

Thầy giáo hướng dẫn TS. Vũ Duy Linh đã tận tình chỉ dẫn, giúp đỡ tôi hoàn thành luận văn này.

Tôi xin chân thành cảm ơn tới các thầy cô giáo, gia đình và bạn bè đã động viên khích lệ, tạo điều kiện giúp đỡ tôi trong suốt quá trình học tập và hoàn thiện luận văn này

Học viên

Đào Quang Toàn

MỤC LỤC

LỜI MỞ ĐẦU

Nhận dạng khuôn mặt là một khái niệm được phát triển vào những năm 60 của thế kỷ trước. Khi đó, người ta phải dùng tới những phương pháp tính toán thủ công để xác định vị trí, khoảng cách và các bộ phận trên khuôn mặt. Về sau, vào cuối thập niên 80, kỹ thuật nhận diện khuôn mặt dần được cải thiện khi M. Kirby và L. Sirovich phát triển phương pháp tìm mặt riêng (eigenface) sử dụng phương pháp phân tích thành phần chính (PCA), một cột mốc mới trong ngành công nghệ nhận diện khuôn mặt. Ngày nay, có thể dễ dàng nhận ra ứng dụng công nghệ nhận diện khuôn mặt trong việc điều tra tội phạm, kiểm tra hành khách ở sân bay, và xác thực truy cập vào hệ thống. Hệ thống nhận diện khuôn mặt được triển khai khá rộng rãi ở Mỹ, vốn trước kia chỉ dành cho các cơ quan thực thi pháp luật của nước này. Chính phủ Mỹ hiện đang ứng dụng công nghệ này để đảm bảo an ninh quốc gia thông qua việc nhận diện khuôn mặt kẻ tội phạm ngay khi chúng xuống sân bay và loại bỏ những lá phiếu gian lận thông qua việc xác định khuôn mặt người đi bầu cử...

Tuy nhiên ở VN chúng ta việc ứng dụng của công nghệ nhận dạng khuôn mặt vào thực tế còn hạn chế. Mục tiêu của luận văn này là tìm hiểu một số phương pháp nhận diện khuôn mặt và ứng dụng vào thực tế cho phù hợp với thực tế nước nhà.

Chương 1: KHÁI QUÁT VỀ QUẢN LÝ TIẾP DÂN VÀ BÀI TOÁN NHẬN DIỆN KHUÔN MẶT

1.1. Khái quát về quản lý tiếp dân

- Trong các vụ việc tiếp dân (tại các phòng tiếp dân của UBND xã, quận / huyện) chia ra làm 3 loại:
    + Khiếu nại
    + Tố cáo
    + Kiến nghị, phản ánh
- Có 3 loại hình thức tiếp dân
    + Tiếp dân thường xuyên: tiếp dân hàng ngày trong giờ hành chính theo quy định của nhà nước, do cán bộ phòng tiếp dân chịu trách nhiệm ghi chép, báo cáo.
    + Tiếp dân theo định kỳ: theo định kỳ làm việc lãnh đạo của cơ quan, đơn vị phải tiến hành các cuộc tiếp dân trực tiếp.
    + Tiếp dân đột xuất: Khi tiến hành giải quyết các vụ việc phát sinh và lãnh đạo cơ quan đơn vị cần gặp công dân phải tiến hành các cuộc tiếp dân đột xuất
- Mỗi khi có công dân tới phòng tiếp dân thì cán bộ tiếp dân phải ghi chép vào sổ tiếp dân. Nếu phát sinh vụ việc trong thẩm quyền giải quyết thì phải trình lãnh đạo tiến hành thụ lý vụ việc và giao về bộ phận có trách nhiệm giải quyết. Trong quá trình giải quyết này thông thường sẽ phải tiến hành mời gặp, tiếp công dân nhiều lần để bổ sung thêm thông tin, bằng chứng, thông báo kết quả của quá trình giải quyết. Đồng thời việc tiếp dân này do nhiều cán bộ tiếp dân hoặc lãnh đạo, hoặc cán bộ thuộc sở ban ngành có liên quan tiến hành tiếp dân.

- Trong quá trình giải quyết vụ việc thông thường sẽ phải tiến hành mời gặp, tiếp công dân nhiều lần để bổ sung thêm thông tin, bằng chứng, thông báo kết quả của quá trình giải quyết, đồng thời việc tiếp dân này do nhiều cán bộ tiếp dân hoặc lãnh đạo, hoặc cán bộ thuộc sở ban ngành có liên quan tiến hành tiếp dân. Mỗi lần công dân tới phòng tiếp dân phải trình giấy tờ chứng minh thân phận. Phiền toái này có thể khắc phục bằng cách ứng dụng nhận dạng công dân vào quản lý tiếp dân.

1.2. Bài toán nhận diện khuôn mặt

Hệ thống nhận dạng mặt người là một hệ thống nhận vào là một ảnh hoặc một đoạn video (một chuỗi các ảnh). Qua xử lý, tính toán hệ thống xác định được vị trí mặt người trong ảnh nếu có và xác định là người nào trong số những người hệ thống đã biết (qua quá trình học) hoặc là người lạ

Thuật toán nhận diện khuôn mặt hiện chia làm hai loại là hình học (geometric) và trắc quang (photometric). Hình học nhận diện khuôn mặt dựa trên các đặc trưng trên khuôn mặt như mắt, mũi, miệng, gò má; trong khi trắc quang là phương pháp biến hình ảnh thành các giá trị và so sánh với giá trị mẫu để nhận diện. Các nhà nghiên cứu ngày nay đã phát triển những kỹ thuật nhận diện khuôn mặt riêng, nhưng phổ biến nhất hiện có ba loại chính là phân tích thành phần chính (PCA), phân tích phân lớp tuyến tính (LDA) và phương pháp đồ thị đàn hồi (EBGM).

Cách nhận diện khuôn mặt sử dụng phương pháp PCA phụ thuộc rất nhiều vào cơ sở dữ liệu ban đầu chứa các ảnh mẫu và góc quay camera cũng như ánh sáng. Sử dụng các thuật toán đại số để tìm giá trị mặt riêng và vector riêng rồi so sánh với giá trị mẫu, ta thu được khuôn mặt cần nhận diện. Đặc điểm của phương pháp này là giảm thiểu được dữ liệu cần sử dụng làm mẫu. Trong khi đó, phương

pháp LDA lại phân loại các lớp chưa biết thành các lớp đã biết, mà ở đó các khuôn mặt tạo thành một lớp và sự khác biệt giữa các khuôn mặt trong một lớp là rất nhỏ. Cả PCA và LDA đều chọn cách thống kê lấy mẫu, chọn lọc để nhận diện khuôn mặt.

Phương pháp còn lại EBGM chia mặt thành mạng lưới gồm các nút với mỗi khuôn mặt có khoảng 80 điểm nút. Vị trí của các nút giúp xác định khoảng cách giữa hai mắt, độ dài của sống mũi, độ sâu của hốc mắt, hình dạng của gò má… Điểm khó của phương pháp này là cần tính toán chính xác khoảng cách giữa các điểm nút, và do đó đôi khi nó phải dùng kết hợp với các phương pháp như PCA hay LDA.

Với những hạn chế khi sử dụng công nghệ nhận diện khuôn mặt truyền thống, phương pháp nhận diện 3D đã trở thành hướng đi mới trong việc ứng dụng công nghệ nhận diện khuôn mặt. Phương pháp này lưu lại hình ảnh 3D của khuôn mặt với các điểm đặc trưng như độ cong của cằm, mũi, hốc mắt… Ưu điểm của nó là có thể nhận diện khuôn mặt ở nhiều góc độ khác nhau, không bị ảnh hưởng bởi ánh sáng.

Cũng như những phương pháp truyền thống, phương pháp nhận diện khuôn mặt 3D cũng dựa trên các thuật toán. Nó tính toán các đường cong, những điểm đặc trưng trên khuôn mặt để tạo thành những dòng lệnh duy nhất và so sánh với cơ sở dữ liệu. Chúng ta có thể dễ dàng bắt gặp quá trình so sánh này trong các bộ phim hành động của Mỹ, khi hình ảnh của một người được camera ghi lại và ngay lập tức nó được so sánh liên tục với hàng triệu khuôn mặt trong cơ sở dữ liệu của cảnh sát.

Ngày nay, các công ty của Mỹ đã cải tiến phương pháp nhận diện 3D bằng việc bổ sung thêm nhận diện mẫu da, được gọi là phương pháp phân tích vân bề mặt. Phương pháp này cũng sử dụng các thuật toán chia nhỏ vùng da thành các

không gian có thể đo đếm được, giúp xác định danh tính của cả những cặp sinh đôi.

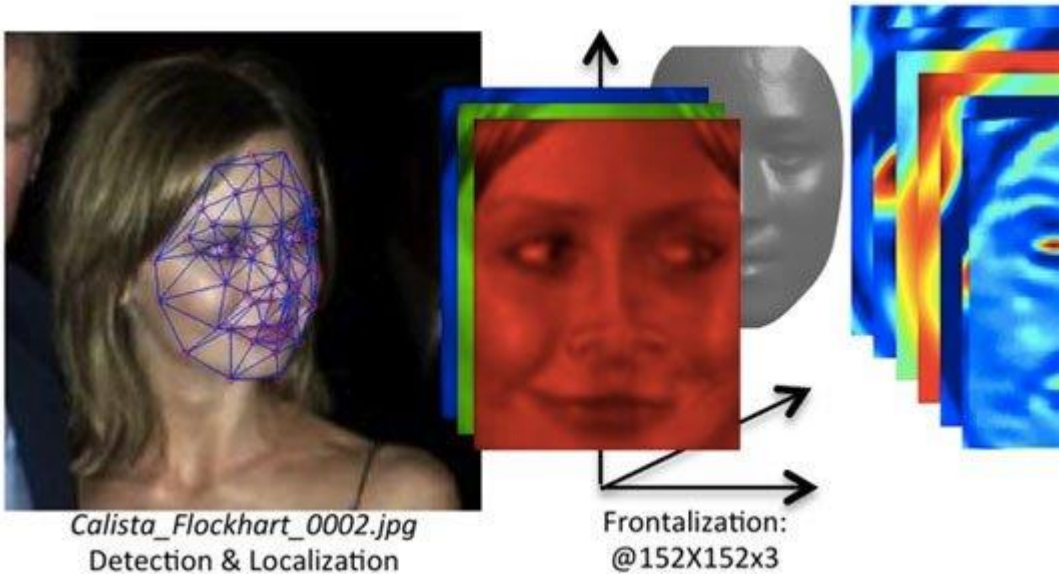
Nhận diện khuôn mặt 3D vẫn chưa hoàn hảo, nó vẫn bị hạn chế đáng kể bởi các yếu tố bên ngoài như khuôn mặt bị tóc che phủ, đeo kính, hình ảnh quá mờ. Các công ty của Mỹ hiện vẫn đang liên tục tìm cách cải tiến để tăng độ chính xác cho công nghệ nhận diện khuôn mặt mà không gây khó chịu cho người bị nhận diện.

Tại Trung Quốc, đất nước này sử dụng công nghệ nhận diện khuôn mặt thay vé tham quan. Du khách tới thị trấn Wuzhen, Trung Quốc sẽ không phải mua vé nhờ hệ thống kiểm soát bằng công nghệ nhận diện khuôn mặt tại cổng. Khi du khách đến thị trấn, ảnh chân dung của họ được chụp lại và tải lên cơ sở dữ liệu. Máy tính bảng gắn trên cổng ra vào của mỗi khu vực nhất định sẽ quay hình người vào cổng, rồi gửi hình ảnh đến kho dữ liệu. Tại đây, trí thông minh nhân tạo sẽ nhận diện du khách và quyết định người đó có được phép vào cửa hay không. Toàn bộ quá trình diễn ra trong 0,6 giây với tỷ lệ chính xác là 99,77%. Nó giúp rút ngắn thời gian chờ và tăng sự thuận tiện cho du khách. Theo Baidu và công ty du lịch Wuzhen, công nghệ này đang nhận được nhiều phản hồi tích cực. Ngoài ra, Baidu cũng cho phép các đối tác của họ sử dụng công nghệ nhận diện khuôn mặt. Một đối tác đã áp dụng nó để thay thế chìa khóa và mật mã trên cửa nhà. Cánh cửa tự động mở ra khi một người được phép vào nhà tới gần.

Các nhà vệ sinh công cộng ở thủ đô Bắc Kinh - Trung Quốc bắt đầu sử dụng công nghệ nhận diện khuôn mặt nhằm ngăn nạn trộm giấy vệ sinh đang diễn ra tràn lan. Tại Thiên Đàn, một trong những địa điểm du lịch thu hút nhiều du khách ở thủ đô Bắc Kinh và từng là điểm nóng của nạn trộm giấy vệ sinh, người dùng nhà vệ sinh cần khăn giấy phải đứng trước một chiếc máy gắn trên tường có camera độ phân giải cao để được phát giấy vệ sinh. Một người chỉ được lấy từ 60-70cm giấy

vệ sinh trong mỗi 9 phút. Thiết bị có thể ghi nhớ nhiều khuôn mặt trong thời gian gần và nếu một người nào đó quay lại nhà vệ sinh trong một khoảng thời gian nhất định, nó sẽ từ chối cung cấp giấy vệ sinh. Camera nhận diện khuôn mặt yêu cầu người dùng giấy vệ sinh phải tháo mũ và kính mát.

Facebook có công nghệ nhận diện khuôn mặt như mắt người. Facebook ra tuyên bố cho biết đã phát triển công nghệ để máy tính có thể xác nhận liệu 2 khuôn mặt trong 2 bức ảnh có phải là cùng một người hay không. Hiện tại, dự án này (có tên gọi DeepFace) của Facebook đã đạt tới mức độ chính xác 97,25%, tức tương đương với mức 97,5% của mắt người trong các thử nghiệm chuẩn.



Hình 1.1 Nhận dạng khuôn mặt của Facebook

Để thực hiện thành công kì tích này, DeepFace sẽ tạo ra các sơ đồ 3D cho chi tiết khuôn mặt, sau đó chuyển mô hình này thành định dạng phẳng với các bộ lọc màu để nhận diện các chi tiết trên mặt. Hiện tại, Facebook đang sử dụng dữ liệu khuôn mặt của 4,4 triệu người trên mạng xã hội của mình nhằm cải tiến cho DeepFace.

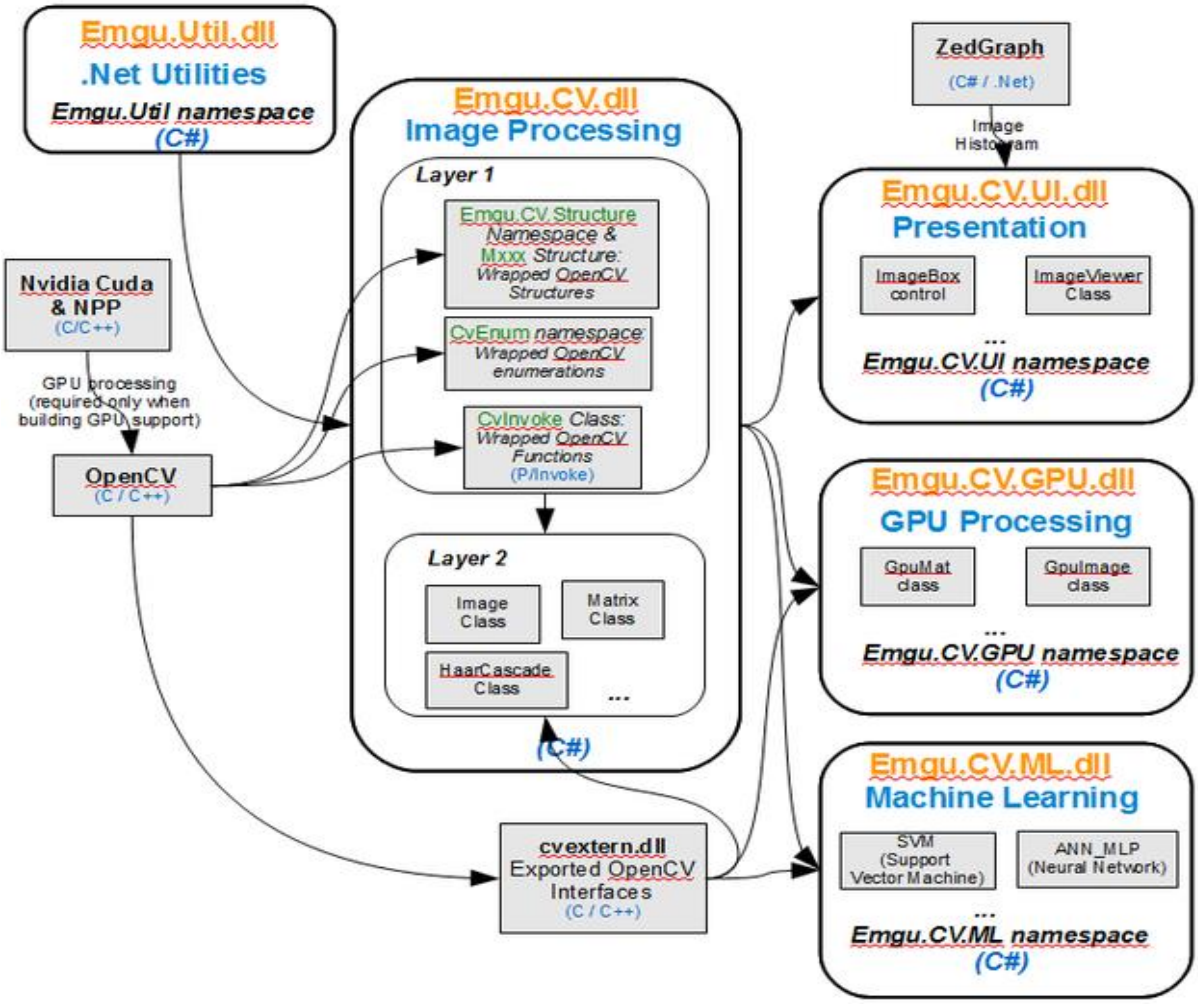
1.3. Thư viện xử lý hình ảnh và thị giác máy tính EmguCV

Chương trình sử dụng thư viện thư viện xử lý hình ảnh và thị giác máy tính EmguCV. EmguCV là một cross flatform .NET, một thư viện xử lý hình ảnh mạnh dành riêng cho ngôn ngữ C#. Cho phép gọi được chức năng của OpenCV là từ .NET

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

Lợi thế của EmguCV

- EmguCV được viết hoàn toàn bằng C#. Có thể chạy trên bất kỳ nền tảng hỗ trợ bao gồm iOS, Android, Windows Phone, Hệ điều hành Mac OS X và Linux.
- EmguCV có thể được sử dụng từ nhiều ngôn ngữ khác nhau, bao gồm C#, VB.NET, C ++ và Iron Python.
- Nhận dạng ảnh: nhận dạng khuôn mặt, các vật thể …
- Xử lý ảnh: khử nhiễu, điều chỉnh độ sáng …
- Nhận dạng cử chỉ.
- Hỗ trợ tài liệu XML và intellisense.
- Sự lựa chọn để sử dụng hình ảnh lớp hoặc trực tiếp gọi chức năng từ OpenCV.



Hình 1.2 Tổng quan về kiến trúc EmguCV

Emgu CV có hai lớp :

- Lớp cơ bản (***layer 1***) có chứa function, structure và ánh xạ bản đồ trực tiếp gọi chức năng từ OpenCV.
- Lớp thứ hai (***layer 2***) có chứa các lớp trộn từ .NET.

1.4. Những khó khăn của nhận diện khuôn mặt

Bài toán nhận diện khuôn mặt là một bài toán khó, các phương pháp nhận diện hiện nay vẫn tồn tại những nhược điểm, độ tin cậy vẫn chưa được như mong muốn, nên vấn đề này vẫn được nhiều nhóm quan tâm nghiên cứu. Khó khăn của bài toán có thể kể ra:

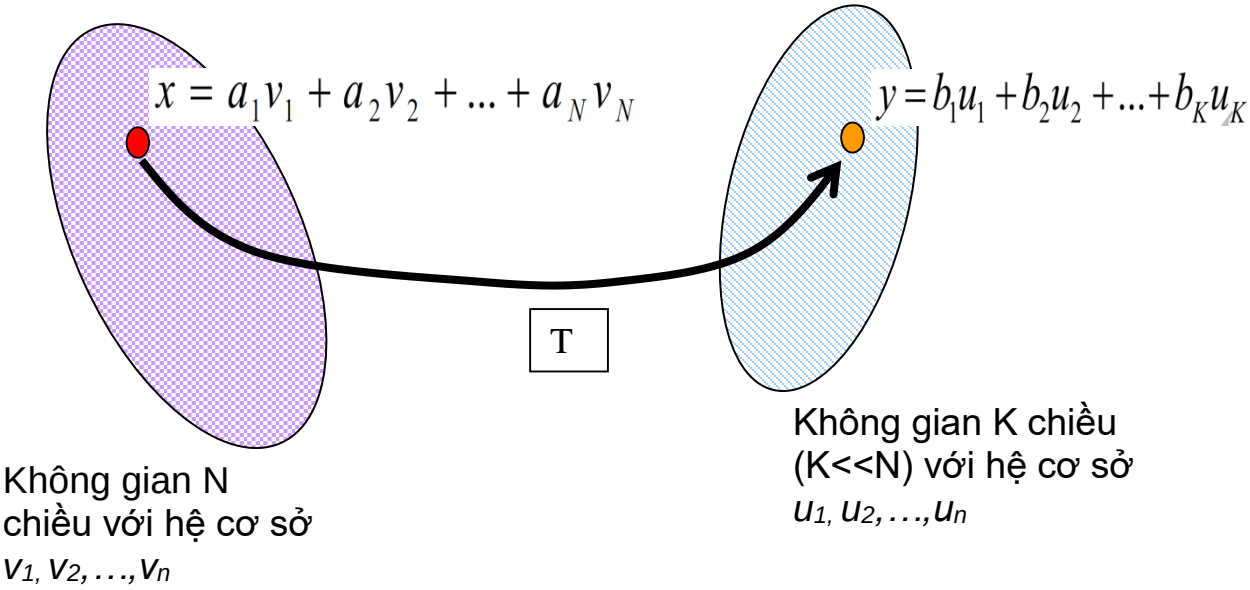
- Góc chụp ảnh
- Mặt bị che khuất một phần
- Biểu cảm của khuôn mặt
- Điều kiện ánh sáng

Chương 2: MỘT SỐ PHƯƠNG PHÁP NHẬN DIỆN KHUÔN MẶT

2.1 Phương pháp PCA (*Principal Component Analysis – Phân tích thành phần chính*)

Ý tưởng chính

Mục tiêu của phương pháp PCA là giảm số chiều của 1 tập vector sao cho vẫn đảm bảo được tối đa thông tin quan trọng nhất . Tức Feature extraction (giữ k thuộc tính mới) chứ không phải Feature selection (giữ lại k thuộc tính nguyên gốc ban đầu).



Hình 2.1 Mô hình ý tưởng chính phương pháp PCA

Vector *x* ban đầu có N chiều khi biến đổi tuyến tính *T* sẽ được vector *y* chỉ còn K chiều. Khi loại bỏ 1 số thành phần của x để thu được y thì sẽ gây ra sai số

→ Phương pháp PCA sẽ cố gắng tìm phép biến đổi tuyến tính *T* thỏa:

y = *T.x* sao cho trung bình bình phương lỗi (MSE) là bé nhất.

Cách để tìm được T:

Gọi M là vector trung bình của tất cả các vector *x* trong tập mẫu.

Gọi ma trận hiệp phương sai của các phần tử *x* trong tập mẫu là C. C được tính theo công thức sau:

$$C = \frac{1}{M-1}\sum_{k=1}^{M}(x_k - M)(x_k - M)^T$$

Người ta chứng minh được rằng:

"Nếu T là ma trận *m* hàng, mỗi hàng là 1 vector riêng của C, đồng thời *m* vector riêng này phải ứng với *m* trị riêng lớn nhất. Khi đó T chính là phép biến đổi thỏa MSE nhỏ nhất".

→ Tóm lại, phương pháp PCA quy về việc đi tìm trị riêng (eigenvalues) và vector riêng (eigenvectors) của ma trận hiệp phương sai C của tập mẫu X. Sau đó, ta chỉ giữ lại K vector riêng ứng với K trị riêng lớn nhất để làm cơ sở cho không gian mới này [1].

Tính toán các Eigenfaces

Kohonen đã đưa ra phương pháp dùng vector riêng để nhận dạng khuôn mặt, ông dùng một mạng neural đơn giản để chứng tỏ khả năng của phương pháp này trên các ảnh đã được chuẩn hóa. Mạng neural tính một mô tả của khuôn mặt bằng cách xấp xỉ các vector riêng của ma trận tương quan của ảnh. Các vector riêng sau này được biết đến với cái tên Eigenface.

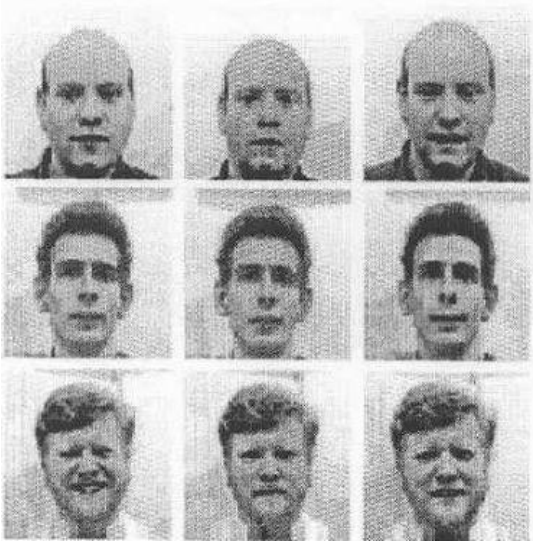
Kirby và Sirovich chứng tỏ các ảnh có các khuôn mặt có thể được mã hóa tuyến tính bằng một số lượng vừa phải các ảnh cơ sở. Tính chất này dựa trên biến đổi Karhunen-Lòeve, mà còn được gọi dưới một cái tên khác là PCA và biến đổi Hotelling. Ý tưởng này được xem là của Pearson trình bày đầu tiên vào năm 1901 và sau đó là Hotelling vào năm 1933. Cho một tập các ảnh huấn luyện có kích thước *n* x *m* được mô tả bởi các vector có kích thước m x m, các vector cở sở cho một không gian con tối ưu được xác định thông qua lỗi bình phương trung bình khi chiếu các ảnh huấn luyện vào không gian con này. Các tác giả gọi tập các vector cơ sở tối ưu này là ảnh riêng sau đó gọi cho đơn giản là

vector riêng của ma trận hiệp phương sai được tính từ các ảnh khuôn mặt đã vector hóa trong tập huấn luyện. Nếu cho 100 ảnh, mà mỗi khuôn mặt có kích thước 91x50 thì có thể chỉ dùng 50 ảnh riêng, trong khi vẫn duy trì được một khả năng giống nhau hợp lý (giữ được 95% tính chất).

Turk và Pentland áp dụng PCA để xác định và nhận dạng khuôn mặt. Tương tự, dùng PCA trên tập huấn luyện ảnh các khuôn mặt để sinh các ảnh riêng (còn gọi là eigenface) để tìm một không gian con (không gian khuôn mặt) trong không gian ảnh. Các ảnh khuôn mặt được chiếu vào không gian con này và được gom nhóm lại. Tương tự các ảnh không có khuôn mặt dùng để huấn luyện cũng được chiếu vào cùng không gian con và gom nhóm lại. Các ảnh khi chiếu vào không gian khuôn mặt thì không bị thay đổi tính chất cơ bản, trong khi chiếu các ảnh không có khuôn mặt thì xuất hiện sự khác nhau cũng không ít. Xác định sự có mặt của một khuôn mặt trong ảnh thông qua tất cả khoảng cách giữa các vị trí trong ảnh và không gian ảnh. Khoảng cách này dùng để xem xét có hay không có khuôn mặt người, kết quả khi tính toán các khoảng cách sẽ cho ta một bản đồ về khuôn mặt. Có thể xác định được từ cực tiểu địa phương của bản đồ này. Có nhiều nghiên cứu về xác định khuôn mặt, nhận dạng, và trích đặc trưng từ ý tưởng vector riêng, phân rã, và gom nhóm. Sau đó Kim phát triển cho ảnh màu, bằng cách phân đoạn ảnh để tìm ứng để không gian tìm kiếm bớt đi [2].

Bước 1:

Sử dụng các ảnh khuôn mặt $I_1, I_2, \ldots I_n$ (tập các khuôn mặt huấn luyện) với khuôn mặt phải chính diện và tất cả ảnh phải cùng kích thước.



Hình 2.2 Các khuôn mặt mẫu

Bước 2:

Biểu diễn mọi ảnh I_i thành vector Γ_i

Ví dụ: Để đơn giản ta giả sử chỉ có 4 *ảnh* trong tập huấn luyện (kích thước 3x3). Ta tính toán được:

$$\Gamma_1 = \begin{bmatrix} 225 \\ 229 \\ 48 \\ 251 \\ 33 \\ 238 \\ 0 \\ 255 \\ 217 \end{bmatrix} \quad \Gamma_2 = \begin{bmatrix} 10 \\ 219 \\ 24 \\ 255 \\ 18 \\ 247 \\ 17 \\ 255 \\ 2 \end{bmatrix} \quad \Gamma_3 = \begin{bmatrix} 196 \\ 35 \\ 234 \\ 232 \\ 59 \\ 244 \\ 243 \\ 57 \\ 226 \end{bmatrix} \quad \Gamma_4 = \begin{bmatrix} 255 \\ 223 \\ 224 \\ 255 \\ 0 \\ 255 \\ 249 \\ 255 \\ 235 \end{bmatrix}$$

Bước 3:

$$\Psi = \frac{1}{M} \sum_{i=1}^{M} \Gamma$$

Tính vector khuôn mặt trung bình Ψ theo công thức:

Cụ thể ta có:

$$\Psi = \begin{bmatrix} 171.50 \\ 176.50 \\ 135.5 \\ 248.25 \\ 27.50 \\ 246.00 \\ 127.25 \\ 205.50 \\ 170.00 \end{bmatrix}$$

Bước 4:

Trừ vector khuôn mặt trung bình: $\Phi_i = \Gamma_i - \Psi$

Cụ thể ta có:

$$\bar{x}^1 = \begin{bmatrix} 53.50 \\ 52.50 \\ -84.50 \\ 2.75 \\ 5.50 \\ -8.00 \\ 127.25 \\ 49.50 \\ 47.00 \end{bmatrix} \quad \bar{x}^2 = \begin{bmatrix} -161.50 \\ 42.50 \\ -108.50 \\ 6.75 \\ -9.50 \\ 1.00 \\ -110.25 \\ 49.50 \\ -168.00 \end{bmatrix} \quad \bar{x}^3 = \begin{bmatrix} 24.50 \\ -141.50 \\ 101.50 \\ -16.25 \\ 31.50 \\ -2.00 \\ 115.75 \\ -148.50 \\ 56.00 \end{bmatrix} \quad \bar{x}^4 = \begin{bmatrix} 83.50 \\ 46.50 \\ 91.50 \\ 6.75 \\ -27.50 \\ 9.00 \\ 121.75 \\ 9.49.50 \\ 65.00 \end{bmatrix}$$

Bước 5:

Tính ma trận hiệp phương sai (covariance) C:

C sẽ có kích thước N^2xN^2

$$C = \frac{1}{M} \sum_{n=1}^{M} \Phi_n \Phi_n^T \overset{1/M}{=} AA^T$$

$$A = [\Phi_1 \ \Phi_2 \cdots \Phi_M]$$

Trong đó: A sẽ có kích thước là N^2xM

Cụ thể ta có:

$$A = \begin{bmatrix} 53.50 & -161.50 & 24.50 & 83.50 \\ 52.50 & 42.50 & -141.50 & 46.50 \\ -84.50 & -108.50 & 101.50 & 91.50 \\ 2.75 & 6.75 & -16.25 & 6.75 \\ 5.50 & -9.50 & 31.50 & -27.50 \\ -8.00 & 1.00 & -2.00 & 9.00 \\ -127.25 & -110.25 & 115.75 & 121.75 \\ 49.50 & 49.50 & -148.50 & 49.50 \\ 47.00 & -168.00 & 56.00 & 65.00 \end{bmatrix}$$

Ma trận A kích thước N^2xM *(9x4)* tạo bởi ghép các Φ_i làm các cột

Từ đó ta dễ dàng tính được ma trận hiệp phương sai C, kết quả như sau:

$$C = \begin{bmatrix} 36517 & -3639 & 23129 & -778 & 304 & 113 & 24000 & -4851 & 36446 \\ -3639 & 26747 & -19155 & 3045 & -5851 & 324 & -22083 & 28017 & -9574 \\ 23129 & -19155 & 37587 & -1997 & 1247 & 1188 & 45603 & -20097 & 25888 \\ -778 & 3045 & -1996 & 363 & -746.5 & 78 & -2153 & 3217 & -1476 \\ 304 & -5851 & 1247 & -747 & 1869 & -364 & 645 & -6237 & 1831 \\ 113 & 324 & 1188 & 78 & -364 & 150 & 1772 & 396 & -71 \\ 24000 & -22083 & 45603 & -2153 & 645.5 & 1772 & 56569 & -22919 & 26937 \\ -4851 & 28017 & -20097 & 3218 & -6237 & 396 & -22919 & 29403 & -11088 \\ 36446 & -9574 & 25888 & -1476 & 1831 & -71 & 26937 & -11088 & 37794 \end{bmatrix}$$

Ma trận C kích thước N^2x N^2 *(9x9)*.

Bước 6:

Tính các Eigenvector u_i (*"vector riêng"*) của ma trận vuông $A.A^T$ (C có kích thước N^2xN^2).

Ma trận này có kích thước quá lớn → không khả thi → phải khôn khéo đi đường vòng.

- **Bước 6.1:**

Xét ma trận $A^T.A$ (chú ý ma trận này chỉ có kích thước là MxM)

Cụ thể ta có:

$$A^T.A = \begin{bmatrix} 33712 & 11301 & -33998 & -115015 \\ 11301 & 82627 & -50914 & -43014 \\ -33998 & -50914 & 70771 & 14141 \\ -11015 & -43014 & 14141 & 39888 \end{bmatrix}$$

Kích thước MxM *(4x4)*.

- **Bước 6.2:**

Tính các vector riêng v_i (eigenvectors) của ma trận vuông $A^T.A$ này.

Tips: Về cách tìm trị riêng (eigenvalues) và vector riêng (eigenvectors) có thể xem lại "Toán cao cấp – Tập 1 – Đại số và Hình học Giải tích". Tuy nhiên, cách này không khả thi khi lập trình. Phải dùng "Phương pháp lặp" (thuật toán QR) để tìm.

Ở đây ta sẽ tìm được 4 trị riêng của ma trận $A^T.A$, tuy nhiên ta sẽ sắp xếp lại theo thứ tự giảm dần, và chỉ lấy các trị riêng "non-zero". Kết quả ta được 3 trị riêng (từ đó tính ra 3 vector riêng tương ứng):

$$v_1 = \begin{bmatrix} -0.263 \\ -0.679 \\ 0.586 \\ 0.355 \end{bmatrix} \quad v_2 = \begin{bmatrix} 0.521 \\ -0.437 \\ -0.559 \\ 0.475 \end{bmatrix} \quad v_3 = \begin{bmatrix} -0.640 \\ 0.314 \\ -0.306 \\ 0.631 \end{bmatrix}$$

$$\lambda_1 = 153520 \qquad \lambda_2 = 50696 \qquad \lambda_3 = 22781$$

Các eigenvector của $A^T.A$ tương ứng với các eigenvalues. Kích thước Mx*1* và các eigenvalues của $A^T.A$.

Sau khi đã tính được các vector v_i (có kích thước là $Mx1$), ta sẽ dễ dàng suy ra được các vector riêng u_i (kích thước N^2x1) mong muốn cần tìm, theo công thức:

$$\boxed{u_i = Av_i} \quad (*)$$

$$u_1 = \begin{bmatrix} 139.5734 \\ -109.108 \\ 187.906 \\ -12.435 \\ 13.699 \\ 3.452 \\ 219.448 \\ -116.108 \\ 157.593 \end{bmatrix} \quad u_2 = \begin{bmatrix} 124.311 \\ 109.995 \\ -9.981 \\ 10.777 \\ -23.655 \\ 0.781 \\ -25.098 \\ 11.715 \\ 97.366 \end{bmatrix} \quad u_3 = \begin{bmatrix} -39.787 \\ 52.380 \\ 46.675 \\ 9.591 \\ -33.493 \\ 11.725 \\ 88.213 \\ 60.533 \\ -58.978 \end{bmatrix}$$

Các eigenvector của ma trận hiệp phương sai $C(A^T.A)$ cần phải tìm. Kích thước $N^2 * 1$

Chú ý nên chuẩn hóa các vector u_i ($\|u_i\| = 1$), nghĩa là: $u_i = \dfrac{u_i}{\|u_i\|}$

Sau khi chuẩn hóa ta thu được các vector u_i cuối cùng như sau:

$$u_1 \begin{bmatrix} 0.356 \\ -0.279 \\ 0.480 \\ -0.032 \\ 0.035 \\ 0.009 \\ 0.560 \\ -0.296 \\ 0.402 \end{bmatrix} \quad u_2 \begin{bmatrix} -0.552 \\ -0.489 \\ 0.044 \\ -0.048 \\ 0.105 \\ -0.004 \\ 0.112 \\ 0.492 \\ -0.432 \end{bmatrix} \quad u_3 \begin{bmatrix} -0.264 \\ 0.347 \\ 0.309 \\ 0.064 \\ -0.222 \\ 0.078 \\ 0.585 \\ 0.401 \\ -0.391 \end{bmatrix}$$

Đây là các vector cơ sở của không gian mới. Ta gọi các vector này là các eigenfaces

- ***Ta rút ra được các chú ý sau (theo TOÁN HỌC):***
  - Ma trận $A.A^T$ có thể có đến N^2 trị riêng (mỗi trị riêng sẽ ứng với vô số vector riêng – nó được gọi là *"không gian riêng ứng với trị riêng nào đó"*).
  - Ma trận $A^T.A$ có thể có đến M trị riêng.
  - M trị riêng của ma trận $A^T.A$ (kèm với các vector riêng tương ứng), sẽ ứng với M trị riêng lớn nhất của $A.A^T$ (cái này quan trọng bởi ta đang đi tìm các hướng biến thiên quan trọng nhất).
- **Bước 6.3:**

Tính M vector riêng u_i tốt nhất của $A.A^T$ theo công thức

$$u_i = Av_i$$

Bước 7:

Chỉ giữ lại K vector riêng trong số M vector nói trên (ứng với K trị riêng *lớn nhất*), tất nhiên $K<N^2$.

Có 2 cách để xác định K.

Cách 1:

- Sắp xếp theo thứ tự dãy *giảm dần* các eigenvalues tìm được.
- Theo dõi sự biến thiên của dãy trên, khi không còn biến thiên (hoặc xấp xỉ bằng không) thì lúc đó ta đã chọn đủ K.

Cách 2:

Chọn K theo công thức sau:

$$\frac{\sum_{i=1}^{K} \lambda_i}{\sum_{i=1}^{N} \lambda_i} > Threshold \quad \text{(e.g., 0.9 or 0.95)}$$

Trong đó N là Alpaydin

Biểu diễn các khuôn mặt (tập huấn luyện) vào trong không gian vector

Mỗi khuôn mặt Φ_i trong tập huấn luyện có thể được biểu diễn lại là 1 tổ hợp tuyến tính của K vector riêng lớn nhất:

$$\sum_{j=1}^{K} w_j u_j$$

Trong đó:

$$(w_j = u_j^T \Phi_i)$$

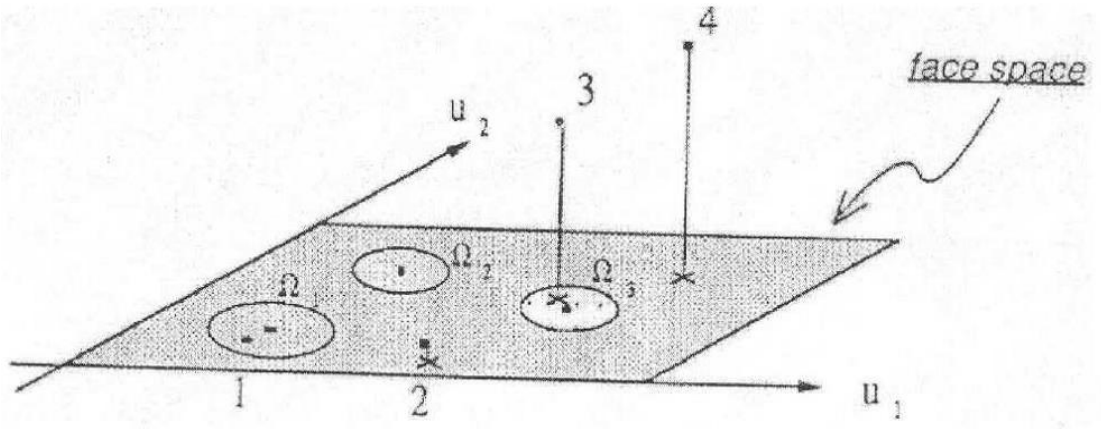
Ta thống nhất gọi các ***"vector riêng u_i"*** là các **"EIGENFACES"** (khuôn mặt riêng). Như vậy, bây giờ, mỗi khuôn mặt huấn luyện Φ_i sẽ được biểu diễn trong không gian mới là 1 vector như sau:

$$\Omega_i = \begin{bmatrix} w_1^i \\ w_2^i \\ \dots \\ w_K^i \end{bmatrix}, \quad i = 1, 2, \dots, M$$

Đây là tọa độ của véc tơ $\hat{\Phi}$

$$\Omega_i = \begin{bmatrix} u_1^T.\Phi_i \\ u_2^T.\Phi_i \\ u_3^T.\Phi_i \\ \dots\dots \\ \dots\dots \\ u_K^T.\Phi_i \end{bmatrix}$$

Kích thước vector này chỉ còn là Kx1



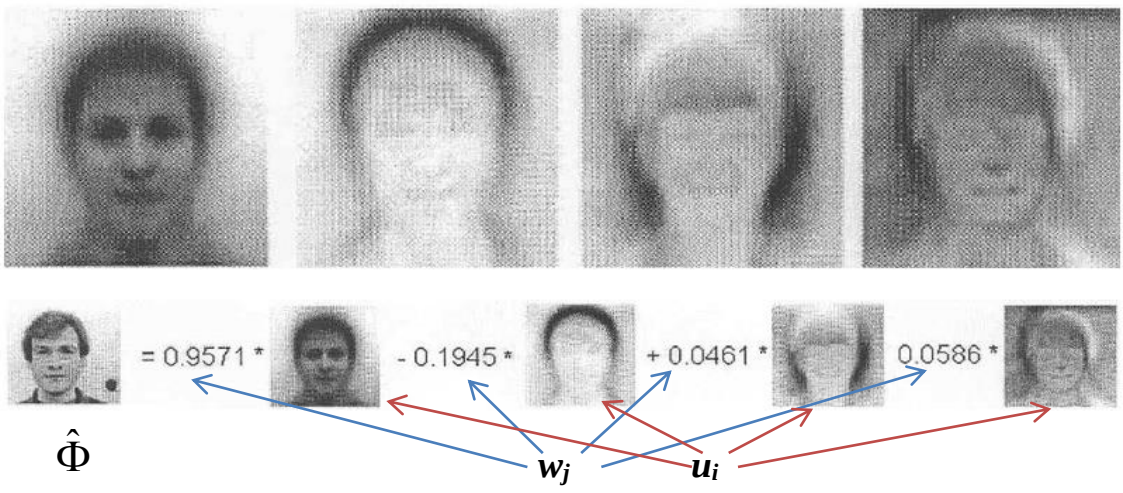
Hình 2.3 Mô hình vector hình chiếu khuôn mặt

u1, u2: là các vector cơ sở u_i của không gian khuôn mặt (được gọi là *"EIGENFACE")*

Các vector Φ_i là ảnh khuôn mặt "gốc" ban đầu

Chiếu các Φ_i vào trong *"không gian khuôn mặt"* ta được các vector $\hat{\Phi}$. Tọa độ của $\hat{\Phi}$ trong không gian này là vector Ω_i

Hình này không thật sát với tổ chức dữ liệu, chỉ là mô tả cho dễ hình dung. *Ví dụ: Eigenface có kthước* N^2x*1, hình của* $\hat{\Phi}$ *, hình eigenfaces….*

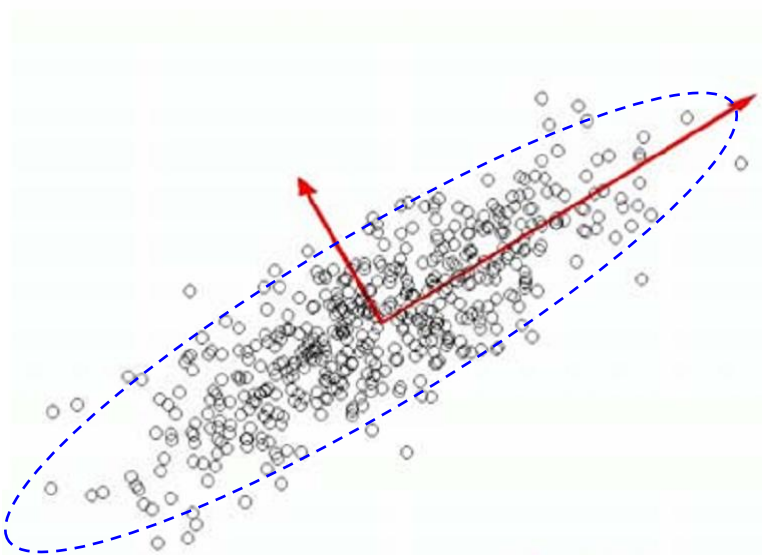


Hình 2.4 Mô hình vector hình chiếu khuôn mặt

Khuôn mặt $\hat{\Phi}$ có thể biểu diễn là tổ hợp tuyến tính của các vector cơ sở
Các w_j của vector Ω_i (đây chính là tọa độ trong không gian khuôn mặt)
Các Eigenface (vector cơ sở u_i của không gian khuôn mặt)

Diễn giải hình học của phương pháp PCA (tìm các eigenvalue/vectors):

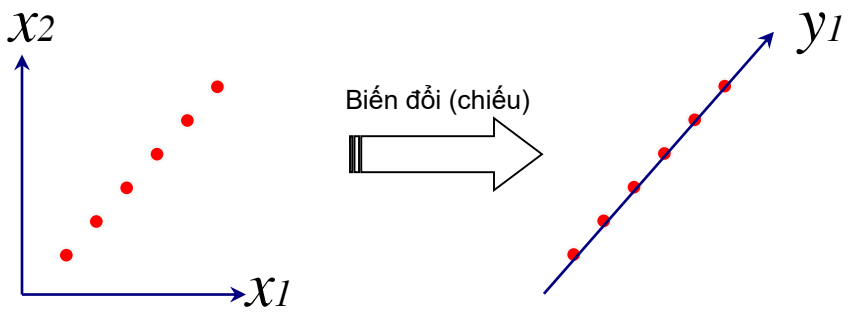
- PCA chiếu dữ liệu theo hướng mà ở đó dữ liệu khác nhau nhiều nhất.
- Các hướng này được xác định bằng các eigenvectors của ma trận hiệp phương sai *(covariance matrix)*.



Hình 2.5 Diễn giải hình học của phương pháp PCA

Gốc tọa độ chính là Ψ
Mặt Ellipsoid định nghĩa bởi covariance matrix *(các sample vector tập trung chủ yếu ở trong này)*
Trục dài nhất ám chỉ hướng biến thiên quan trọng nhất của dữ liệu (có variance – *"phương sai"* - cao nhất). Đây chính là hướng của Eigenvector ứng với eigenvalues lớn nhất.

Ví dụ về khả năng tại sao PCA lại có thể *"giảm được chiều dữ liệu"*:



Hình 2.6 Khả năng *"giảm được chiều dữ liệu"* PCA

Vector ***2 chiều***, trong không gian với các cơ sở là x_1, x_2

Vector chỉ còn 1 chiều, trong không gian với cơ sở là y_1

Nhận dạng khuôn mặt bằng các EIGENFACES

Cho 1 ảnh khuôn mặt chưa biết là ai Γ (tất nhiên cũng phải giống tập mẫu – *chính diện & cùng kích thước*). Chú ý, giai đoạn nhận dạng này giống hệt giai đoạn biểu diễn ảnh khuôn mặt trong tập mẫu. Ta thực hiện lần lượt các bước sau:

Bước 1:

Chuẩn hóa $\Gamma : \Phi = \Gamma - \Psi$

Bước 2:

Biểu diễn Φ thành Ω như sau: $\Omega = \begin{bmatrix} u_1^T.\Phi \\ u_2^T.\Phi \\ u_3^T.\Phi \\ \ldots\ldots \\ \ldots\ldots \\ u_K^T.\Phi \end{bmatrix}$

Bước 3:

Tìm: $e_r = \min_l \|\Omega - \Omega^l$

Tức ta tìm khuôn mặt thứ *l* trong tập mẫu có khoảng cách gần nhất với khuôn mặt cần nhận dạng.

Bước 4:

Nếu $e_r < T_r$ (T_r là 1 ngưỡng chấp nhận được nào đó):

Tức ảnh khuôn mặt cần xác định "đủ gần" với ảnh của người thứ *l* trong tập mẫu. Khi đó, ta kết luận đó chính là khuôn mặt của người thứ *l*.

- **Chú ý:**

Ta có thể sử dụng khoảng cách Euclid để tính e_r. Tuy nhiên, người ta chứng mình được rằng, kết quả sẽ tốt hơn nếu dùng khoảng cách *"Mahalanobis"*:

$$\|\Omega - \Omega^k\| = \sum_{i=1}^{K} \frac{1}{\lambda_i}(w_i - w_i^k)^2$$

Ứng dụng dò tìm – *"DETECTION"*

Thật ra, đây không đúng nghĩa là "detection", mà chỉ là xác định xem có phải là ảnh mặt người (hay chỉ là ảnh phong cảnh) hay không mà thôi.

Bước 1:

Tính $\Phi = \Gamma - \Psi$

Bước 2:

Tính $$\hat{\Phi} = \sum_{i=1}^{K} w_i u_i \quad (w_i = u_i^T \Phi)$$

Bước 3:

Tính $$e_d = \|\Phi - \hat{\Phi}\|$$

Bước 4:

Nếu $e_d < T_d$ (1 ngưỡng nào đó chấp nhận được):

Tức ảnh cần xác định có khoảng cách "đủ gần" với không gian khuôn mặt. Khi đó, ta kết luận đó chính là ảnh khuôn mặt người.

Có 1 tài liệu *(Slides của Pradeep Buddharaju)* đề cập đến việc tính ngưỡng như sau:

$$\theta_c = ½ \max_{j,k} \{||p_j - p_k||\}; j,k = 1,\ldots,m$$

Tức là θ_c bằng ½ khoảng cách lớn nhất giữa 2 khuôn mặt bất kỳ.

Ngưỡng này sẽ được dùng thay cho cả T_d và T_r

Nhược điểm của Eigenfaces

Trong những trường hợp sau, PCA Eigenfaces sẽ nhận dạng sai:

- Khác nhau về điều kiện ánh sáng
- Khác nhau về điệu bộ (nghiêng đầu, quay trái, quay phải ...)
- Cảm xúc (cười to, há miệng, nhe răng …)

2.2 Phương pháp EBGM (*Elastic Bunch Graph Matching - So khớp đồ thị đàn hồi*)

Ý tưởng chính

So khớp đồ thị đàn hồi (EBGM) là một thuật toán bắt nguồn từ sinh học được sử dụng cho các bài toán nhận dạng đối tượng trong lĩnh vực thị giác máy. Nguồn gốc sinh học của thuật toán này gồm hai phần

- Các đặc điểm trực quan được sử dụng dựa trên biến đổi Gabor wavelet. Biến đổi này đã được chứng minh là một mô hình tốt về xử lý hình ảnh trong não bộ con người, chính xác hơn là các tế bào đơn giản trong vỏ não thị giác chính.

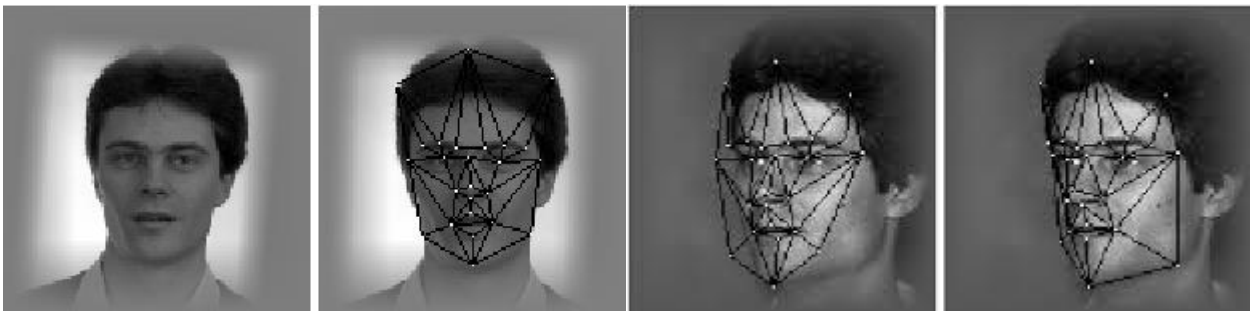
- Các thuật toán tự so khớp: Đây là một phiên bản của thuật toán liên kết động phù hợp (DLM) do Wiskott và von der Malsburg đề xuất năm 1996, đó là một mô hình của nhận dạng các đối tượng bất biến trong não.

Các đối tượng trực quan trong EGM được biểu diễn như là một đồ thị có nhãn, nơi các nút đại diện cho đặc điểm riêng biệt dựa trên Gabor wavelet và các cạnh đại diện cho khoảng cách giữa các địa điểm nút trên một hình ảnh. Vì vậy, một hình ảnh của một đối tượng được biểu diễn như là một tập hợp đặc điểm riêng trong một tổ chức không gian

nhất định. Nếu một đối tượng mới trong một hình ảnh được nhận dạng, các đồ thị được dán nhãn của các đối tượng được lưu trữ (vì vậy được gọi là đồ thị mô hình) và được so khớp trên hình ảnh. Đối với mỗi đồ thị mô hình, các vị trí của các nút trong các hình ảnh được tối ưu hóa. Nhờ vậy mà đặc điểm riêng biệt của hình ảnh phù hợp với giữa các nút của mô hình. Đồ thị mô hình phù hợp nhất với đối tượng được nhận dạng và với các vị trí nút của nó trong hình ảnh, một đồ thị hình ảnh có thể được tạo ra.

Elastic Bunch Graph Matching (EBGM) là một mở rộng của EGM phù hợp cho các lớp đối tượng có một cấu trúc phổ chung, chẳng hạn như những khuôn mặt trong tư thế giống hệt nhau. Tất cả các trường hợp có thể của một lớp được biểu diễn bởi cùng một loại đồ thị. Từ các đồ thị, một bó các đồ thị có cùng một cấu trúc được tạo ra với các nút đại diện cho kết cấu địa phương của bất kỳ đối tượng trong các lớp học (ví dụ: tất cả các biến thể của một con mắt bên trái) và các cạnh đại diện cho khoảng cách trung bình giữa các địa điểm nút (ví dụ: khoảng cách trung bình giữa hai mắt). Điều này cho phép tận dụng lợi thế của các tổ hợp của các đặc điểm riêng biệt đại diện cho các trường hợp của các lớp đối tượng không nhìn thấy trước. Ví dụ, các kết cấu của mắt có thể được lấy từ một khuôn mặt và kết cấu của miệng từ mặt khác đại diện cho một khuôn mặt mới mà chia sẻ các tính năng với hai gương mặt được lưu trữ. Như vậy, một biểu đồ bó là một khái niệm trừu tượng đại diện cho lớp đối tượng chứ không phải là đối tượng cá nhân.

Như vậy, phép biến đổi Gabor wavelet tạo ra một mô hình khuôn mặt dạng cấu trúc lưới đàn hồi liên kết động. Như trên Hình 1.10, các nút trên lưới đàn hồi được gọi là các Gabor Jet, được ký hiệu bởi các vòng tròn mô tả đặc điểm của khuôn mặt. Đó là kết quả tích chập của hình ảnh đầu vào với một bộ lọc Gabor, được sử dụng để phát hiện và trích xuất các các đặc trưng của khuôn mặt cho quá trình nhận dạng. Phương pháp này có thể nhận được khuôn mặt ở mọi tư thế, nhưng phương pháp này khó thực hiện bởi nó đòi hỏi vị trí chính xác của các nút trên mô hình lưới.

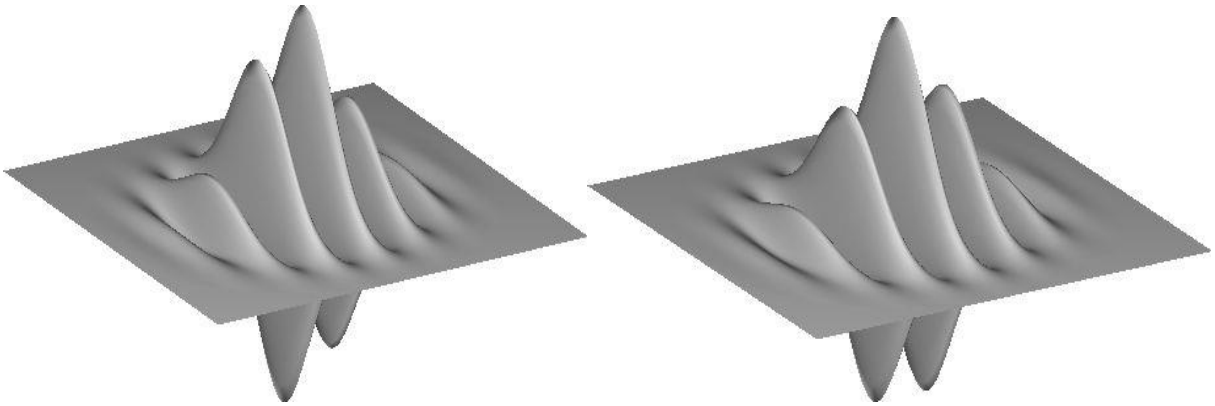


Hình 2.7 Các nút trên mô hình lưới

Nội dung thuật toán

Các đặc trưng cơ bản thường được sử dụng nhất trong EBGM được dựa trên biến đổi Gabor wavelet, có hình dạng của một sóng sin hoặc cos nhân với một hàm đường biên Gaussian:

$$\psi_{\vec{k}}(\vec{x}) = \frac{\vec{k}^2}{\sigma^2}\exp\left(-\frac{\vec{k}^2\vec{x}^2}{2\sigma^2}\right)\left(\exp\left(i\vec{k}\vec{x}\right)-\exp\left(-\frac{\sigma^2}{2}\right)\right)$$



Hình 2.8 Cosine Gabor wavelet và Sin Cosine Gabor wavelet đại diện cho phần thực và phần ảo của biến đổi Wavelet

Thành phần mũ đầu tiên là hàm đường bao Gaussian với σ/k xác định chiều rộng của nó. Hàm mũ thứ hai là một hàm phức bao gồm một sóng cosin trong phần thực và một sóng sin trong phần ảo với vector sóng $\vec{k}$ xác định phương hướng và tần số không

gian của sóng. Thành phần mũ thứ thứ ba là một điều chỉnh nhỏ để đảm bảo rằng các wavelet có giá trị trung bình bằng không. Hệ số k_2/σ_2 để chuẩn hóa wavelet. Cách biểu diễn các wavelet như trên nhằm đảm bảo rằng đối với mỗi giá trị σ cố định các wavelets có hướng và tần số khác nhau (tức là $\vec{k}$ khác nhau) sẽ trông giống nhau ngoại trừ tỷ lệ và hướng.

Về mặt toán học các wavelets được định nghĩa với các biến liên tục, nhưng trong thực tế, thuật toán được rời rạc hóa với độ phân giải hình ảnh, và $\vec{k}$ rời rạc của nó như sau:

$$\vec{k}_{m,l} = k_{\max}\alpha^{-m}\begin{pmatrix}\cos\left(\frac{\pi l}{L}\right)\\ \sin\left(\frac{\pi l}{L}\right)\end{pmatrix} \quad m = 0,\ldots,M-1,\ l = 0,\ldots,L-1$$

Các thông số thường được sử dụng là L = 8, M = 5, α = 2, tức là, có 8 định hướng và 5 tần số. Do vậy, tổng số bộ lọc phức tạp là 40. Chiều rộng tương đối của các hàm đường bao Gaussian thường được chọn là $\sigma = 2\pi$.

Biến đổi Gabor Wavelets

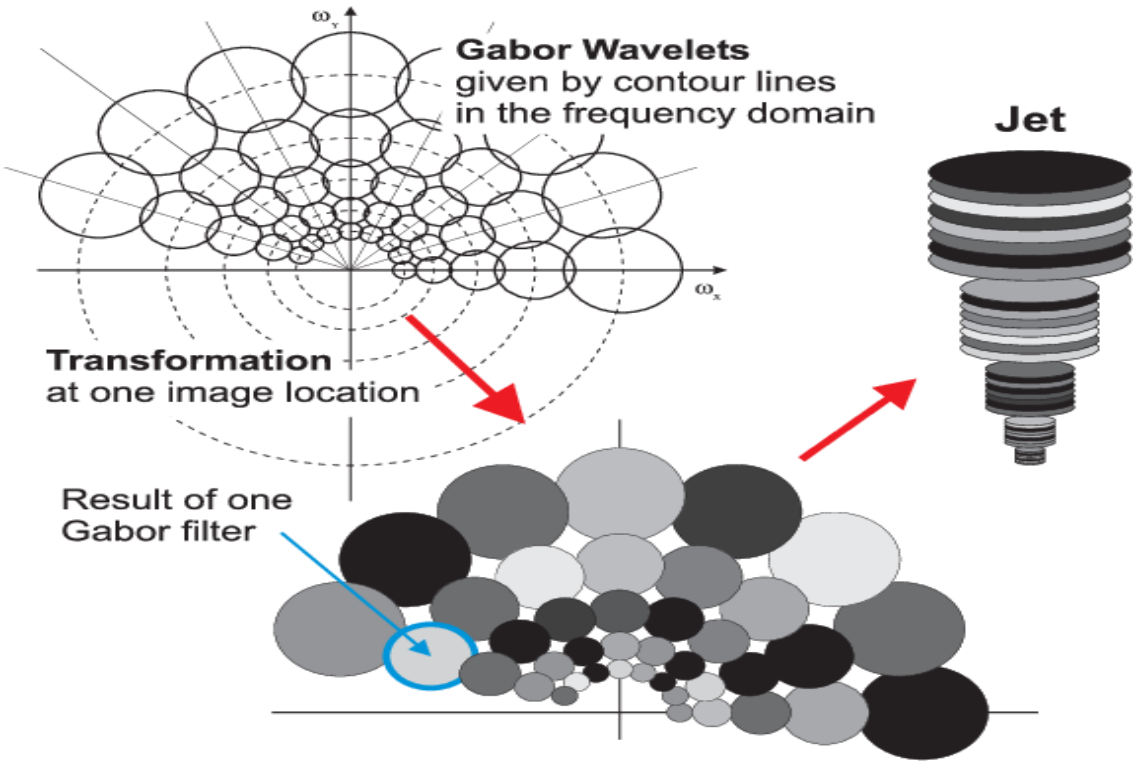
Một đáp ứng wavelet Gabor duy nhất tại một vị trí ảnh $\vec{x_0}$ có thể được tính bằng cách tập trung các wavelet $\Psi_{\vec{x}}(\vec{x})$ tại $\vec{x}_0$, nhân hình ảnh các giá trị mức xám $I(\vec{x})$ với các giá trị của các wavelet ở vị trí tương ứng, và tích hợp trên tất cả vị trí hình ảnh. Điều này sẽ được thực hiện với tất cả $L{\cdot}M$ wavelets để mang lại một mô tả kết cấu xung quanh $\vec{x}_0$ so sánh với một biến đổi Fourier cục bộ. Tuy nhiên, trong quá trình so khớp, những đáp

ứng wavelet Gabor cần thiết được thực hiện ở tất cả các vị trí của hình ảnh. Vì vậy, ta có thể áp dụng một tích chập không gian của hình ảnh với wavelets $L \cdot M$ tạo ra đáp ứng ở tất cả các địa điểm:

$$WI\left(\vec{x}_0, \vec{k}\right) = \left(\psi_{\vec{k}} * I\right)\left(\vec{x}_0\right) = \int \psi_{\vec{k}}\left(\vec{x}_0 - \vec{x}\right) I\left(\vec{x}\right) d^2x$$

Biến đổi Gabor Jet

Biến đổi Gabor wavelet tạo ra một giá trị cho mỗi wavelet ở tất cả các vị trí của hình ảnh. Như vậy, với các thông số tiêu chuẩn và hình ảnh số hóa sẽ nó mang lại 80 (40 thực + 40 ảo) giá trị tại vị trí pixel bất kỳ. Tập các giá trị cho một vị trí duy nhất điểm ảnh được gọi là một Gabor Jet (Hình 1.12). Do các Gabor Jet chứa các giá trị từ các wavelets ở tần số và hướng khác nhau, ta có thể xem nó như một biến đổi Fourier địa phương, và như vậy nó là như là một đại diện như vậy của các đặc điểm riêng biệt. Trong thực tế có thể tái tạo lại các giá trị ảnh xám từ một Gabor Jet trong một môi trường nhỏ xung quanh vị trí của nó, ngoại trừ giá trị trung bình.



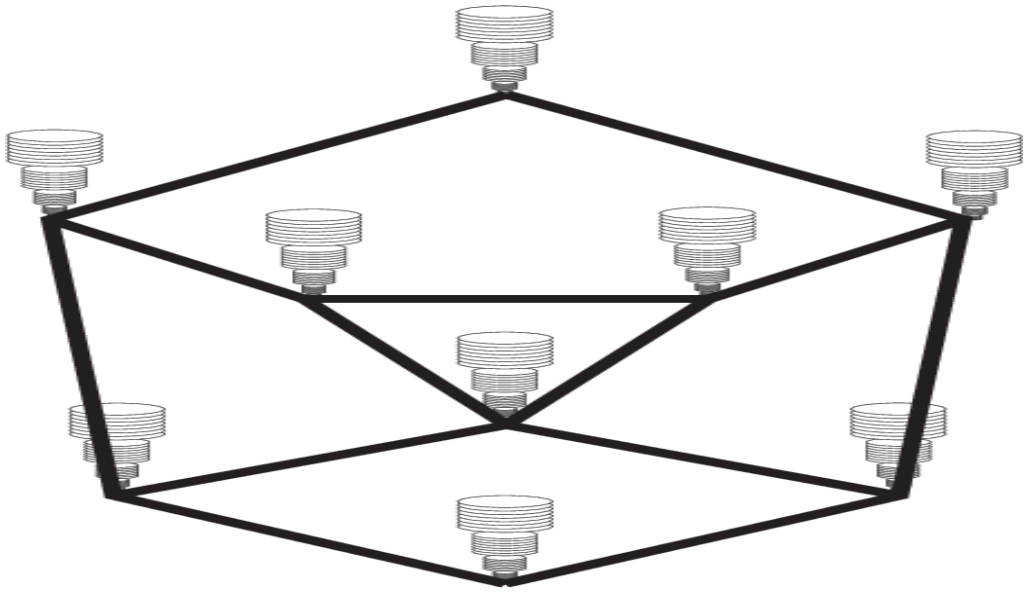
Hình 2.9 Tạo Gabor Jet từ biến đổi Gabor wavelet [5]

Các Gabor wavelet là một cặp bộ lọc cosin (phần thực) và sin (phần ảo). Bản thân mỗi bộ lọc tương đối nhạy cảm với sự thay đổi nhỏ tại một trong các c vị trí điểm ảnh trong một ảnh tĩnh. Tuy nhiên, việc bình phương và cộng thêm đáp ứng của các cặp như thế làm giảm số lượng các giá trị xuống còn 40.

Đồ thị

Một Gabor Jet là một đại diện của một đặc điểm địa phương riêng biệt. Để đại diện cho hình ảnh của toàn bộ đối tượng, người ta cần phải kết hợp nhiều Gabor Jet trong một tổ chức không gian xác định. Người ta có thể sử dụng Gabor Jet ở tất cả các vị trí điểm ảnh trong các khu vực của đối tượng, nhưng đó sẽ là thừa. Vì vậy một trong những Gabor Jet sẽ đại diện mẫu của hình ảnh. Trong dạng đơn giản nhất này có thể là một mảng hình chữ nhật với khoảng cách cố định mà từ đó Gabor Jet được ghi lại và lưu trữ để đại diện cho một đối tượng. Trong trường hợp tổng quát hơn ta có thể xác định một đồ thị (với đỉnh V và cạnh E) đại diện cho các đối tượng và cho phép xác định vị trí các nút ở các

điểm đặc biệt quan trọng về đối tượng, được sử dụng như là điểm mốc.



Hình 2.10 Các nút được gắn nhãn với các Gabor Jet tương ứng

Các đại diện đầy đủ của một đối tượng duy nhất tạo thành một đồ thị có nhãn với đỉnh i∈V là Gabor Jet Ji, cạnh (i, j) ∈E với khoảng cách $\overrightarrow{\Delta_{ij}}$ giữa các vị trí điểm ảnh mà từ đó các Gabor Jet đã được thực hiện trong ảnh gốc. Có thể N là tương đối dễ dàng tái tạo lại hình ảnh của một đối tượng từ như một đại diện đồ thị nếu pha của các Gabor Jet được bảo tồn, tức là 80 giá trị mỗi Gabor Jet được lưu trữ theo các thông số tiêu chuẩn.

Hai đồ thị G^I và G^M với cấu trúc tương tự có thể được so sánh bằng một hàm phân loại sao cho đạt giá trị cao khi các các đồ thị giống hệt nhau và thấp cho các đồ thị từ những hình ảnh rất khác nhau. Nó bao gồm một sự kết hợp tuyến tính tương tự với các đỉnh và các cạnh [4].

$$S_v\left(J^I, J^M\right) = \frac{\left|J^I\right| . \left|J^M\right|}{\left\|J^I\right\| \left\|J^M\right\|}$$

$$S_e\left(\vec{\Delta}_{ij}{}^I, \vec{\Delta}_{ij}{}^M\right) = -\frac{\left\|\vec{\Delta}_{ij}{}^I - \vec{\Delta}_{ij}{}^M\right\|}{\left\|\vec{\Delta}_{ij}{}^M\right\|^2}$$

$$S_G\left(G^I, G^M\right) = \sum_{i \in v} S_v\left(J_i{}^I, J_i{}^M\right) + \lambda \sum_{i,j \in E} S_e\left(\vec{\Delta}_{ij}{}^I, \vec{\Delta}_{ij}{}^M\right)$$

So khớp đồ thị đàn hồi

Giả sử chúng ta đưa ra một mô hình đồ thị G_M và một hình ảnh mới I. So khớp đồ thị đàn hồi là quá trình mà một đồ thị được tìm thấy trong các hình ảnh đó cũng phù hợp với mô hình đồ thị theo hàm đánh giá độ tương tự đồ thị

$$S_G\left(G^I, G^M\right) = \sum_{i \in v} S_v\left(J_i{}^I, J_i{}^M\right) + \lambda \sum_{i,j \in E} S_e\left(\vec{\Delta}_{ij}{}^I, \vec{\Delta}_{ij}{}^M\right)$$

Đây là một bài toán tối ưu hóa trong một không gian có kích thước gấp đôi số đỉnh của đồ thị do mỗi node được phối hợp với nhau theo chiều dọc và chiều ngang. Đơn giản nhất có thể chọn vị trí nút $\vec{x_i}$, i∈V, trong hình ảnh một cách ngẫu nhiên và tính toán sự tương tự của các kết quả đồ thị hình ảnh G_I với đồ thị mô hình G_M. Lặp lại quá trình này với các đồ thị mẫu M đã có và chỉ giữ lại những kết quả phù hợp nhất sẽ mang lại một đồ thị mô hình phù hợp với đồ thị hình ảnh.

Bó đồ thị

Việc so khớp đồ thị giải quyết các vấn đề của việc tìm kiếm một đối tượng được biết đến trong một hình ảnh bằng cách kết hợp mô hình đồ thị hình ảnh. Tuy nhiên, sẽ thuận lợi hơn nếu ta cũng có thể tìm thấy một đối tượng không rõ trong một hình ảnh và tái tạo ra một đồ thị hình ảnh cho nó. Điều này mang lại ít nhất là hai lợi thế lớn: Thứ nhất, đồ thị mô hình cho các đối tượng mới có thể được tạo ra mà không cần sự trợ giúp của nhãn. Thứ hai, có thể sẽ không cần phải so khớp với mọi mô hình trong tập mẫu các hình ảnh. Một bó đồ thị hình ảnh có thể được tạo ra một cách độc lập từ một mô hình cụ thể và sau đó chỉ đồ thị so sánh mới được thực hiện cho từng mô hình, trong đó sẽ tiết kiệm rất nhiều tính toán.

Đánh giá thuật toán

Ưu điểm

- Có độ chính xác cao hơn so với PCA và LDA. Có thể áp dụng để nhận dạng hình ảnh khuôn mặt với nhiều đặc tính phi tuyến.

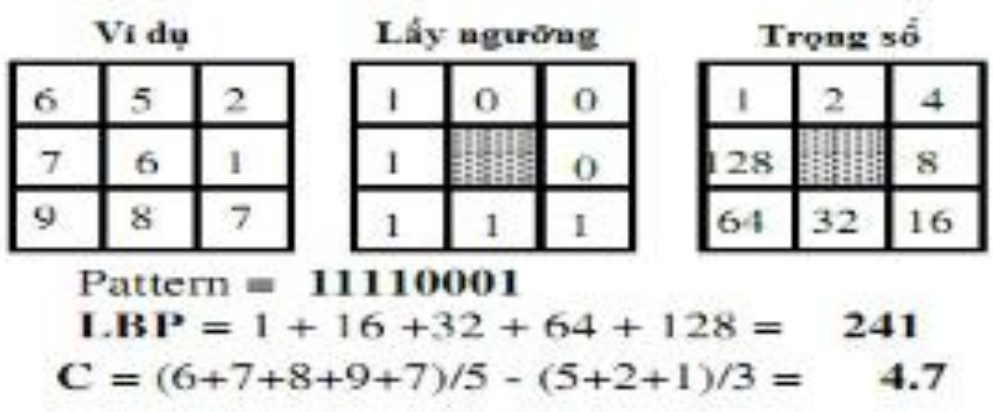
- Có thể áp dụng khi đặc tính của khuôn mặt thay đổi do ảnh hưởng của các yếu tố như: điều kiện chiếu sáng (ánh sáng ngoài trời so với đèn huỳnh quang trong nhà), tư thế (đứng thẳng so với nghiêng) và cảm xúc khuôn mặt (cười so với giận giữ).

Nhược điểm

- Khó thực hiện bởi nó đòi hỏi vị trí chính xác của các nút trên mô hình lưới

2.3 Phương pháp LBP (*Local Binary Pattern*)

LBP là một toán tử kernel 3×3, nó tổng quát hóa cấu trúc không gian cục bộ của một ảnh. Ojala và các đồng nghiệp đã giới thiệu phương pháp LBP và chỉ ra khả năng phân tách cao của chúng cho sự phân lớp vân. Tại một vị trí pixel (x_c, y_c) cho trước, LBP được định nghĩa như một chuỗi nhị phân có trật tự dựa trên sự so sánh giá trị độ xám của pixel trung tâm (x_c, y_c) và 8 pixel lân cận của nó. Như vậy mỗi pixel sẽ được biểu diễn bởi một chuỗi nhị phân, giá trị thập phân của chuỗi nhị phân này chính là giá trị của pixel trung tâm trong sự biểu diễn bởi toán tử LBP.

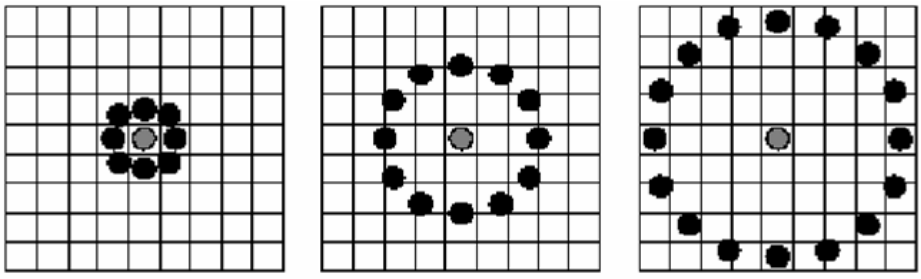


Hình 2.11 Ví dụ tính toán LBP

Nguồn gốc: Dãy LBP được Ojala trình bày vào năm 2002. Định nghĩa một cấu trúc điểm ảnh T là một phân phối đại số của cấp độ xám của P +1 (P > 0) điểm ảnh.

$$T = t(g_c, g_0, \ldots, g_{p-1})$$

Với g_c ứng với cấp độ xám của điểm ảnh trung tâm P_{tt} , g_p (p = 0,…,1) tương ứng với P điểm ảnh xung quanh, P điểm ảnh này nằm trên đường tròn bán kính R và tâm là P_{tt} .



P=8, R=1.0 P=12, R=2.5 P=16, R=4.0

Hình 2.12 Tập hợp các điểm xung quanh P_{tt}

Không mất thông tin, có thể trừ g_p đi một lượng là g_c

$$T = t(g_c, g_0 - g_c, \ldots, g_p - g_c)$$

Giả sử sự sai số giữa g_p và g_c là độc lập với g_c, ta có thể nhân tử hóa g_c như sau:

$$T = t(g_c)t(g_0 - g_c, \ldots, g_{p-1} - g_c)$$

$t(g_c)$ biểu thị xu hướng độ sáng tối của cả bức ảnh nên không liên quan đến kết cấu của ảnh cục bộ do đó có thể bỏ qua

$$T \sim t((g_0 - g_c), \ldots, (g_{p-1} - g_c))$$

Mặc dù tính bất biến ngược với độ thay đổi tỷ lệ xám của điểm ảnh, sự khác biệt ảnh hưởng bởi tỷ lệ. Để thu được đặc điểm bất biến với bất kỳ một sự thay đổi nào của ảnh đen trắng (gray scale) chỉ quan tâm đến dấu của độ lệch:

$$T \sim t(s(g_0 - g_c), \ldots, s(g_{p-1} - g_c))$$

Với s là hàm dấu $s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$

Trọng số 2^p được dùng cho các hàm dấu $s(g_p - g_c)$ để chuyển sự khác biệt giữa các điểm ảnh bên cạnh về một giá trị duy nhất.

$$LBP_{P,R} = \sum_{p=0}^{p-1} s(g_p - g_c) * 2^p$$

Theo Eq.2 cứ P pixel thì có 2^p giá trị $LBP_{P,R}$ trong khoảng $[0,2^p - 1]$ nhưng để đơn giản ta có thể chọn một số giá trị trong 2^p giá trị ký hiệu là $LBP_{P,R}$

Nguyên lý phân lớp không tham biến: Trong phân lớp, sự khác biệt giữa mẫu và mô hình phân phối LBP được đánh giả bởi kiểm tra thống kê không tham biến. Phương pháp tiếp cận này có ưu điểm là không cần phải có những giả thiết về phân phối của các đặc trưng. Thông thường, những kiểm tra thống kê được chọn cho mục đích là nguyên lý cross- entropy được giới thiệu bởi Kullback (1968). Sau đó, Sokal và Rohlf (1969) gọi cách đo này là thống kê G.

$$G(S,M) = 2^* \sum_{b=1}^{B} Sb \log \frac{Sb}{Mb} = 2 \sum_{b=1}^{B} [Sb * \log Sb - Sb * \log Mb]$$

Với S,M kí hiệu phân phối mẫu và mô hình mong muốn. S_b và M_b là xác suất để b thuộc vào phân phối mẫu hoặc mô hình . B là số phần tử trong phân phối. Thống kê G sử dụng trong phân lớp có thể viết lại như sau:

$$L(S,M) = - \sum_{b=1}^{B} Sb \log Mb$$

Kiến trúc mô hình có thể xem như xử lý ngẫu nhiên có đặc tính có thể xác định bởi phân phối LBP. Trong một phân lớp đơn giản , mỗi lớp được biểu diễn bởi một mô hình phân phối đơn giản M^i. Tương tự , một kiến trúc mẫu không xác định có thể miêu tả bởi phân phối S. L là một giả ma trận đo khả năng mẫu S có thể thuộc lớp i.

Lớp C của một mẫu không xác định có thể được xác định bởi luật "hàng xóm gần nhất":

$$C = \text{argmin}_i L(S,M^i)$$

Bên cạnh đó, một thống kê log- likelihood có thể xem như đơn vị đo sự khác biệt và có thể sử dụng để liên kết nhiều bộ phân lớp giống như bộ phân lớp k-NN hoặc self-

oganizing map (SOM). Log-likelihood đúng trong một số trường hợp nhưng không ổn định khi mà cỡ mẫu nhỏ.Trong trường hợp này Chi-square- distance thường cho kết quả tốt hơn :

$$X^2(S,M) = \sum_{b=1}^{B} \frac{(Sb - Mb)^2}{(Sb + Mb)}$$

Để đạt được độ chính xác cao sử dụng giao histogram

$$H(S,M) = \sum_{b=1}^{B} \min(Sb, Mb)$$

*Phép quay bất biến :*Để không bị ảnh hưởng bởi sự quay, mỗi giá trị LBP cần quay ngược lại về vị trí ban đầu, cách tốt nhất là tạo ra tất cả các trường hợp quay của một mẫu, sự quay có thể định nghĩa như sau:

$$LBP_{R,I}^{ri} = \min \{ ROR(LBP_{P,R}, i) \; i=0,1,\ldots,P-1\}$$

Trong đó ri là viết tắt của rotation invariant (quay bất biến), ROR(x,i) dịch vòng tròn số nhị phân P - bit (x) i lần theo chiều kim đồng hồ.

Độ tương phản và kết cấu mẫu: Kết cấu có thể được coi là một hiện tượng hai chiều được đặc trưng bởi hai đặc tính trực giao: cấu trúc không gian (mô hình) và độ tương phản (độ mạnh của mô hình). Quay bất biến tương phản địa phương có thể được đo trong một hình tròn đối xứng xung quanh giống như LBP:

$$VAR_{P,R} = \frac{1}{P} \sum_{p=0}^{P-1} (g_p - \mu)^2$$

Trong đó

$$\mu = \frac{1}{P}\sum_{p=0}^{P-1} g_p$$

Tổng hợp lại ta có : $LBP^{ri}_{P1,R1}$ / $VAR_{P2,R2}$

2.4 Phương pháp Fisherfaces

Phương pháp nhận dạng khuôn mặt fisherface được mô tả bởi Belhumeur và cộng sự sử dụng cả phân tích thành phần cơ bản - Principal Components Analysis (PCA) và phân tích phân biệt tuyến tính - Linear Discriminant Analysis (LDA) để tạo ra một ma trận chiếu không gian con, tương tự như được sử dụng trong phương pháp eigenface. Tuy nhiên, phương pháp fisherface có thể tận dụng lợi thế của thông tin trong các lớp, giảm thiểu sự thay đổi trong mỗi lớp, nhưng vẫn tối đa hóa tách lớp. Giống như quá trình xây dựng eigenface, bước đầu tiên là lấy từng mảng hình ảnh (n * m) và định hình lại thành một ((N * M) x1) vector.

Thuật toán

- Cho X là một véc tơ ngẫu nhiên với các mẫu được rút ra từ các lớp c:

X = { X_1, X_2, …, X_c } trong đó X_i = { x_1 , x_2, …, x_2 }

- Các ma trận phân tán S_B và S_w được tính như sau:

$$S_B = \sum_{i=1}^{n} N_i(\mu_i - \mu\ )(\mu_i - \mu\ )^T$$

$$S_w = \sum_{i=1}^{c} (x_j - \mu_i)(x_j - \mu_i)^T$$

trong đó μ là tổng bình quân:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Và μ_i là bình quân của lớp i ∈ {1, …, c}

$$\mu_i = \frac{1}{|\ X_i\ |} \sum_{x_j \in X_i} x_j$$

Thuật toán cổ điển Fisher hiện nay sẽ cho một phép chiếu W, nhằm tối đa hóa các tiêu chí phân chia lớp:

$$W_{opt} = \arg\max_W \frac{|W^T S_B W|}{|W^T S_W W|}$$

Một giải pháp cho vấn đề tối ưu hóa này được đưa ra bằng cách giải quyết các vấn đề giá trị riêng:

$$S_B v_i = \lambda_i S_w v_i$$

$$S_W^{-1} S_B v_i = \lambda_i v_I$$

Có một vấn đề còn cần giải quyết: Việc xếp hạng của SW là nhiều nhất (N - c), với N mẫu và c lớp. Trong vấn đề nhận dạng mẫu số mẫu N là hầu như luôn luôn nhỏ hơn kích thước của dữ liệu đầu vào (số lượng điểm ảnh), vì vậy ma trận phân tán SW trở thành số ít. Điều này đã được giải quyết bằng cách thực hiện một phần phân tích chính trên các dữ liệu và dự các mẫu vào không gian (N-c) chiều. Một phân tích phân biệt tuyến tính sau đó được thực hiện trên các dữ liệu giảm, bởi vì SW không ít nữa. [6]

Các vấn đề tối ưu hóa sau đó có thể được viết lại như sau:

$$W_{pca} = \arg\max_W |W^T S_T W|$$

$$W_{fld} = \arg\max_W \frac{W^T W_{pca}^T S_B W_{pca} W}{W^T W_{pca}^T S_W W_{pca} W}$$

Việc chuyển đổi ma trận W, mà đưa một mẫu vào không gian (c -1) chiều được cho bởi:

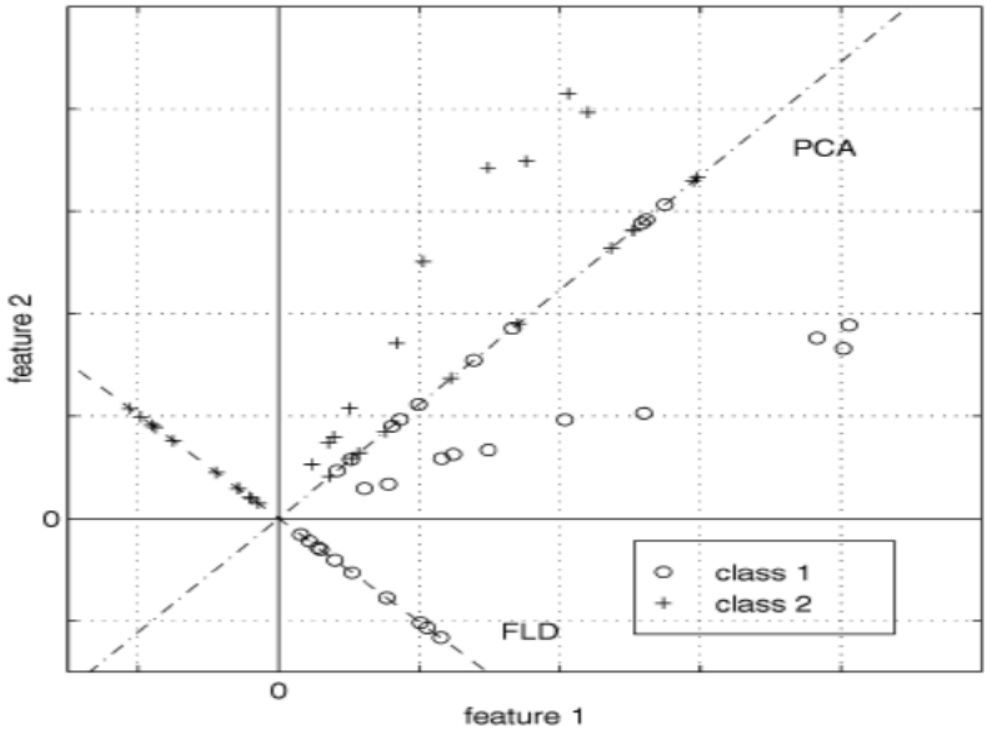
$$W = W_{fld}^T W_{pca}^T$$

Fisherface tương tự như Eigenface nhưng với những cải tiến trong phân loại tốt hơn về hình ảnh các lớp khác nhau. Có Fisher’s Linear Discriminant (FLD), chúng ta có thể phân loại tập huấn luyện để xử lý với những người khác nhau và biểu hiện trên khuôn mặt khác nhau. Chúng ta có thể có độ chính xác tốt hơn trong biểu hiện trên khuôn mặt hơn phương pháp tiếp cận Eigenface. Bên cạnh đó, Fisherface loại bỏ ba thành phần chính

đầu tiên chịu trách nhiệm những thay đổi cường độ ánh sáng, nó là bất biến hơn với cường độ ánh sáng.

Fisherface phức tạp hơn Eigenface trong việc tìm kiếm các dự báo về không gian mặt. Tính tỷ lệ giữa các lớp phân tán đến phân tán trong lớp đòi hỏi rất nhiều thời gian xử lý. Bên cạnh đó, do nhu cầu của phân loại tốt hơn, kích thước của chiếu trong không gian khuôn mặt là không nhỏ gọn như Eigenface, kết quả trong lưu trữ khuôn mặt lớn hơn và nhiều thời gian xử lý hơn trong nhận dạng .

Phân biệt tuyến tính Fisher (FLD, Fisherface) tiếp cận bản đồ các đặc trưng không gian con hai lớp riêng biệt nhất.



Hình 2.13 So sánh PCA và FLD trong bài toán hai lớp, nơi dữ liệu cho mỗi lớp nằm gần một không gian con tuyến tính

Ưu điểm

- FLD đạt được sự phân bố giữa các lớp tốt hơn PCA

- Sử dụng ít bộ nhớ.
- Chính xác hơn phương pháp tiếp cận eigenface

Chương 3: CHƯƠNG TRÌNH THỬ NGHIỆM

3.1 Yêu cầu bài toán.

Mục tiêu chính của luận văn là xây dựng phần mềm nhận dạng công dân. Đây là công cụ hỗ trợ cán bộ tiếp dân nhận diện công dân đến, tránh phiền phức trong việc chứng minh nhân thân, hồ sơ liên quan tới công dân, nhắc nhở công việc, diễn tiến công việc.

Đối với các phòng hành chính tiếp dân thì việc ghi chép và xác định nhân thân của công dân mỗi lần tới là công việc phiền phức lặp đi lặp lại nhiều lần, sau đó phải tìm kiếm lại tài liệu hay kết quả liên quan tới phần tiếp công dân các lần trước đó. Ý tưởng của chương trình này có thể giúp tăng tốc độ xác định thông tin về công dân và tra cứu kết quả các lần tiếp trước, tăng độ chính xác khi xử lý thông tin khi công dân tới quên không mang theo giấy tờ tùy thân, hai công dân trùng họ tên...

3.2 Mô tả thu thập dữ liệu thử nghiệm.

Bước 1: Thu thập dữ liệu ảnh công dân

Đặt camera chương trình tại vị trí có nguồn ánh sáng ổn định, ngang tầm khuôn mặt người để có chất lượng ảnh tốt nhất. Khi có công dân tới phòng tiếp dân , cán bộ tiếp sẽ yêu cầu công dân đứng trước camera để làm thủ tục trước khi tiếp dân.

Bước 2: Nhập dữ liệu cho công dân

Khi công dân đứng trước camera, thì sẽ có hai tình huống xảy ra:

- Công dân đã từng đến và được nhận diện trước đó, máy tính sẽ hiện lên thông tin của công dân cùng các kết quả vụ việc trước đó.
- Công dân chưa đến lần nào, máy tính sẽ nhắc nhở cán bộ tiếp dân ghi lại hình ảnh khuôn mặt của công dân và các tông tin nhân thân của công dân, phục vụ cho lần nhận diện sau.

Bước 3 : Nhận diện công dân các lần đến sau đó

Như đã trình bày ở trên, công dân đã từng đến và được nhận diện trước đó, máy tính sẽ hiện lên thông tin của công dân cùng các kết quả vụ việc trước đó

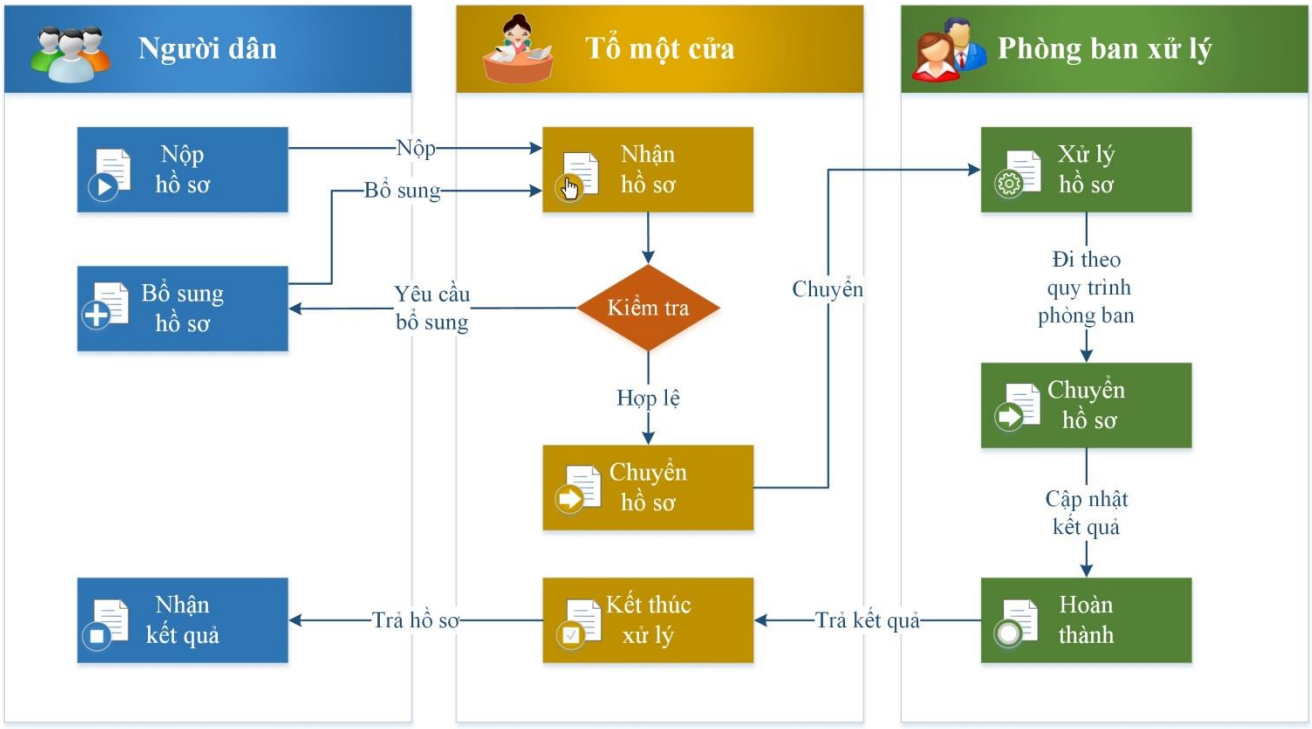
3.3 Phân tích thiết kế chương trình thử nghiệm.

Môi trường cài đặt: Ngôn ngữ C#, trên công cụ Visual Studio 2015 của hãng Microsoft

Thư viện: EmguCV

Cơ sở dữ liệu: CSDL SQL Server 2008, lưu trữ thông tin công dân đến và hình ảnh khuôn mặt công dân. Các hình ảnh khuôn mặt được biểu diễn dưới dạng mảng byte.

Thiết kế của chương trình:



Hình 3.1 Thiết kế hệ thống

Hình 3.1 là toàn bộ hệ thống chương trình quản lý tiếp dân, việc ứng dụng nhận dạng khuôn mặt được tích hợp vào hệ thống ở các khâu người dân tới tiếp xúc với tổ một cửa. Khi người dân tới lần đầu nộp hồ sơ thì tiến hành chụp lại ảnh và đăng ký thông tin cá nhân đưa vào cơ sở dữ liệu để lưu trữ. Ở các khâu tiếp theo khi công dân tới bổ sung hồ sơ hay nhận kết quả, chương trình sẽ đưa ra thông tin công dân và kết quả hồ sơ kèm theo.

Ý định xây dựng lên phần mềm có các chức năng cơ bản như sau:

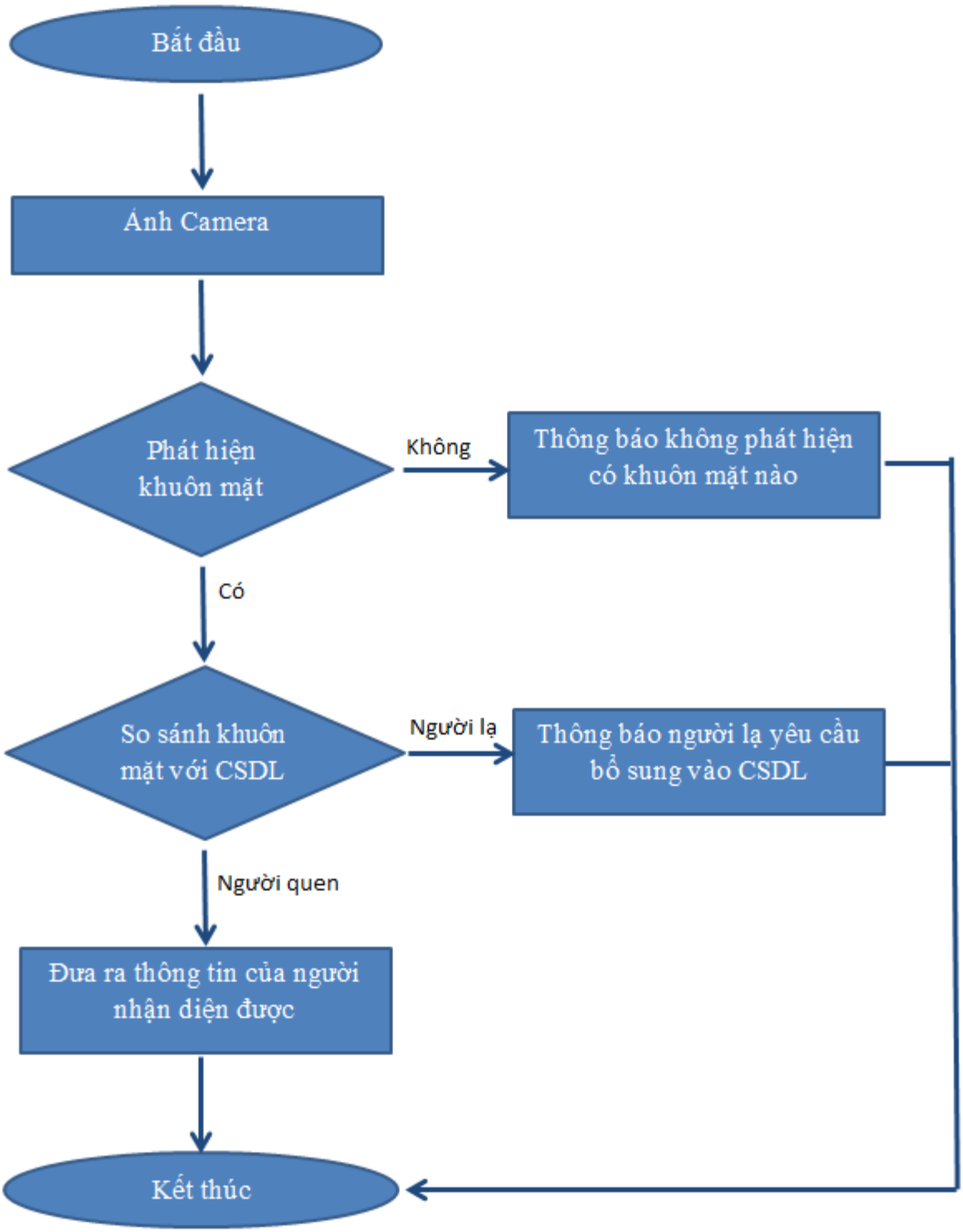
Quản lý tiếp công dân: Các chức năng cho phép người dùng thực hiện việc theo dõi tình hình tiếp công dân cho việc thụ lý đơn thư, giải quyết công việc. Bên cạnh đó, cho phép người dùng thông báo về lịch tiếp công dân, về các văn bản trả lời giải quyết đơn thư của các cơ quan chức năng.

Quản lý đơn thư: Các chức năng cho phép người dùng tại phòng tiếp nhận đơn thư thực hiện các hoạt động tiếp nhận đơn thư và phân phối đơn thư một cách hợp lý đến các đơn vị chuyên môn liên quan. Module cũng cho phép các đơn vị chuyên môn tiếp nhận thụ lý và xử lý các đơn thư.

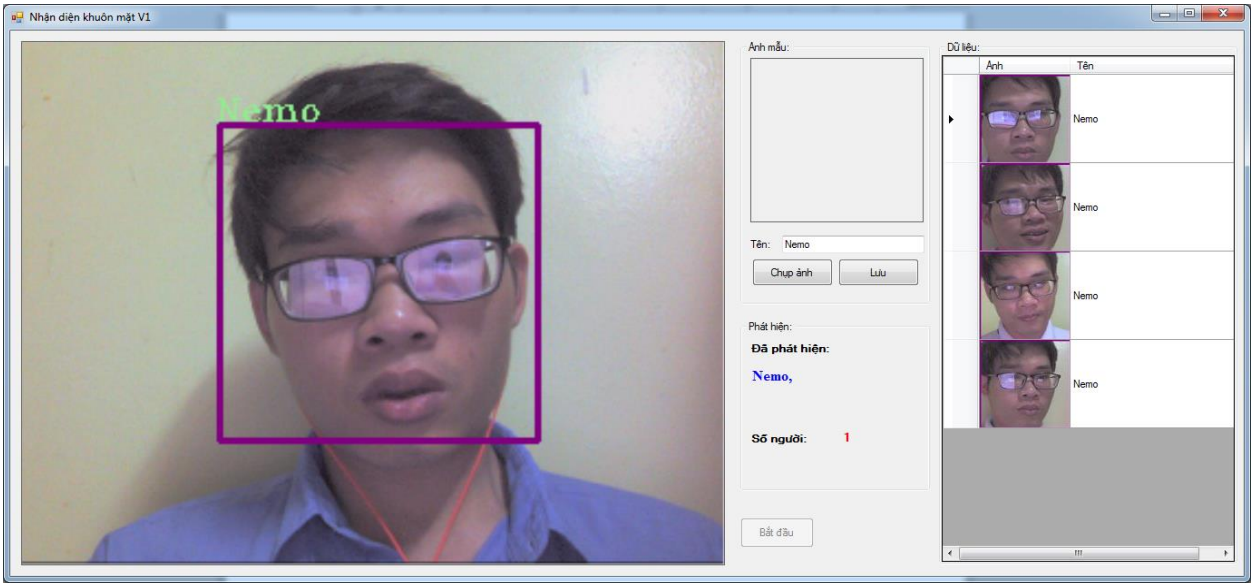
Tra cứu đơn thư: Các chức năng cho phép người dùng tra cứu thông tin đơn thư.

Thống kê, báo cáo đơn thư: Các chức năng tổng hợp, thống kê, báo cáo các số liệu phục vụ cho các nhu cầu quản lý của từng cấp liên quan.

Ứng dụng nhận dạng khuôn mặt được tích hợp vào chương trình nhằm giảm bớt thủ thục chứng minh nhân thân của công dân nếu đã đăng ký nhận dạng trước đó. Trong khuôn khổ của luận văn chương trình thử nghiệm chỉ xây dựng được phần lấy mẫu khuôn mặt và phát hiện khuôn mặt đã qua lấy mẫu theo mô hình trên đây. Đầu vào là ảnh khuôn mặt người, chương trình sẽ cho ra thông tin của người có khuôn mặt đó nếu có. Chương trình sử dụng thuật toán PCA để nhận dạng dựa trên thư viện mở EmguCV.



Hình 3.2 Sơ đồ xử lý nhận dạng



Hình 3.3 Giao diện chương trình đăng ký thông tin và nhận dạng

3.4 Đánh giá độ chính xác

Tính khoảng cách khi số lượng vector đặc trưng thay đổi

Tính toán khoảng cách đến ảnh và đến không gian ảnh, khi số lượng đặc trưng mặt K (eigenface) thay đổi. Điều này sẽ dẫn tới sự thay đổi về mặt thời gian cũng như độ chính xác của chương trình. Để đánh giá được điều này, chia ảnh thử nghiệm làm ba loại:

+ Ảnh mặt của người mà mặt người đó có trong tập ảnh luyện

+ Ảnh mặt của người mới mà người đó không xuất hiện trong tập ảnh huấn luyện

+ Ảnh bất kì (không phải là mặt)

Với mỗi loại ảnh thử nghiệm, đều tính toán hai khoảng cách

+ Khoảng cách đến ảnh mặt gần nhất trong tập luyện

+ Khoảng cách đến không gian mặt

Các khoảng cách là khoảng cách trung bình của tập thử nghiệm tương ứng với mỗi loại. Kết quả thực nghiệm trong Bảng 3.1

Số lượng Eigenfaces	Khoảng cách tới ảnh luyện gần nhất	Khoảng cách đến không gian mặt
Ảnh có khuôn mặt nằm trong tập ảnh luyện		
250	778.6	11519.0
500	1566.5	11351.1
750	2380.8	11061.7
1000	3438.6	4314.9
Ảnh có khuôn mặt không nằm trong tập luyện (gương mặt mới)		
250	1194.4	10951.4
500	2129.3	10839.0
750	3892.7	10470.0
1000	10485.0	7327.3
Ảnh bất kì (không có khuôn mặt)		
250	3257.7	26386.2
500	5874.3	24569.9
750	9996.3	25938.5
1000	18847.0	17617.6

Bảng 3.1 Khoảng cách thay đổi theo số lượng eigenface

Ảnh thử nghiệm được chia làm bốn loại: ảnh trong chính tập luyện, ảnh mặt chưa luyện nhưng có trong tập luyện, gương mặt mới và cuối cùng là ảnh bất kỳ không phải là mặt. Kết quả trả ra là số lượng các ảnh được nhận dạng, không nhận dạng ra mặt và không phải là mặt.

Kết quả thực nghiệm cụ thể trong Bảng 3.2:

Số lượng Eigenfaces	Nhận ra	Không nhận ra	Không phải là mặt	Tổng số
Ảnh có khuôn mặt nằm trong tập ảnh luyện				
250	20	5	0	25
500	22	3	0	25
750	24	1	0	25
1000	26	0	0	25
Ảnh có khuôn mặt không nằm trong tập luyện (gương mặt mới)				
250	7	18	0	25
500	5	20	0	25
750	3	22	0	25
1000	0	25	0	25
Ảnh bất kì (không có khuôn mặt)				
250	0	0	15	15
500	0	0	15	15
750	0	0	15	15
1000	0	0	15	15

Bảng 3.2 Độ chính xác theo số lượng eigenface

Từ bảng thực nghiệm cho thấy :

+ Việc phân biệt ảnh có mặt với ảnh không có mặt (ảnh bất kỳ) đạt được độ chính xác khá cao ngay cả khi thay đổi số lượng eigenface. Đạt được điều này là do sự chênh lệch lớn về khoảng cách đến không gian ảnh giữa hai loại ảnh này như được chỉ ra trong Bảng 3.1.

+ Độ chính xác của phương pháp thay đổi theo số lượng eigenface. Khi số lượng eigenface càng lớn thì khả năng nhận dạng của chương trình càng cao. Qua đây ta có thể rút ra kết luận rằng để tăng độ chính xác của phương pháp nhận dạng, ta cần phải tiến hành tiền xử lý ảnh: chuẩn hóa ảnh mặt theo góc nghiêng, hướng nhìn, điều kiện ánh sáng cũng như ảnh nền.

PHẦN KẾT LUẬN

Qua tìm hiểu cho thấy việc nhận dạng khuôn mặt người hiện nay đã có rất nhiều tiến bộ, được rất nhiều các công ty hàng đầu về công nghệ quan tâm và đưa vào nghiên ứng dụng trong các sản phẩm của họ. Đặc biệt một số còn đưa công nghệ nhận dạng khuôn mặt vào hệ thống bảo mật, xác thực thông tin...cho thấy tiềm năng to lớn của công nghệ này trong cuộc sống.

Tuy nhiên qua quá trình tìm hiểu một số thuật toán cơ bản về nhận dạng khuôn mặt, cho thấy vẫn còn tồn tại nhiều khuyết điểm trong các thuật toán này dẫn đến việc nhận dạng thiếu chính xác. Vì vậy theo quan điểm của học viên đề tài còn có một số hướng phát triển sau:

- Tiếp tục tìm hiểu các thuật toán khác dễ thực hiện và có độ chính xác cao hơn.
- Kết hợp nhiều thuật toán nhận dạng để khắc phục khuyết điểm tồn tại của một thuật toán làm tăng độ chính xác.

TÀI LIỆU THAM KHẢO

[1] Lê Hoàng Thanh , *Dò tìm và nhận dạng khuôn mặt người bằng Eigenface.*

[2] Phạm Thế Bảo, Nguyễn Thành Nhựt, Cao Minh Thịnh, Trần Anh Tuấn, Phan Phú Doãn (2007), *Tổng quan các phương pháp xác định khuôn mặt người*.

[3] Mạch Thị Kim Hạnh (2013), *Xác thực sinh trắc học khuôn mặt trên thiết bị di động*, Luận văn thạc sĩ, Trường Đại học Lạc Hồng, Đồng Nai.

[4] Laurenz Wiskott (2014), *Elastic Bunch Graph Matching***,** Scholarpedia.

[5] Farooq Bhat & M. Arif Wani (2015), *Elastic Bunch Graph Matching Based Face Recognition Under Varying Lighting, Pose, and Expression Conditions*, International Journal of Advance Foundation And Research In Science & Engineering - Volume 1, Issue 8, January 2015

[6] Opencv dev team, *Face Recognition with OpenCV*, http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html

PHỤ LỤC

Mã nguồn:

Form Main

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.CV.CvEnum;
using System.IO;
using System.Diagnostics;
using System.Data;
using System.Drawing.Imaging;
using System.Linq;
using System.ComponentModel;

namespace NhanDienV1
{
   public partial class Main : Form
   {
      //Declararation of all variables, vectors and haarcascades
      Image<Bgr, Byte> currentFrame;
      Capture capture;
      HaarCascade face;
      //HaarCascade eye;
      MCvFont font = new MCvFont(FONT.CV_FONT_HERSHEY_TRIPLEX, 0.5d,
0.5d);
      //Image<Gray, byte> result, TrainedFace = null;
      Image<Bgr, Byte> frame;
      Image<Gray, Byte> tg;
      //Image<Bgr, Byte> bgr = null;
      Image<Gray, byte> gray = null;
      //List<Image<Gray, byte>> trainingImages = new List<Image<Gray, byte>>();
      List<Image<Gray, byte>> listImages = new List<Image<Gray, byte>>();
      List<string> labels = new List<string>();
      List<string> NamePersons = new List<string>();
```

```
int ContTrain, NumLabels, t;
string name, names = null;
private bool captureInProgress;

private void btn_Add_Click(object sender, EventArgs e)
{
   try
   {
      //Trained face counter
      ContTrain = ContTrain + 1;
      //Face Detector
      MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(face, 1.2, 10,
Emgu.CV.CvEnum.HAAR_DETECTION_TYPE.DO_CANNY_PRUNING, new
Size(20, 20));

      //Action for each element detected
      foreach (MCvAvgComp f in facesDetected[0])
      {
         //TrainedFace = currentFrame.Copy(f.rect).Convert<Gray, byte>();
         frame = currentFrame.Copy(f.rect);
         break;
      }

      //resize face detected image for force to compare the same size with the
      //test image with cubic interpolation type method
      frame = frame.Resize(100, 100,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
      ////Chụp toàn bộ khuôn hình
      //imagedata.Image = currentFrame.ToBitmap();
      ////Chỉ chụp mặt
      imagedata.Image = frame.ToBitmap();
   }catch
   {
      MessageBox.Show("Không phát hiện được đối tượng chụp");
   }
}

public Main()
{
   InitializeComponent();
```

```
}

private void Main_Load(object sender, EventArgs e)
{
   //Load haarcascades for face detection
   face = new HaarCascade("haarcascade_frontalface_default.xml");
   //eye = new HaarCascade("haarcascade_eye.xml");

   MyDataDataContext myDB = new MyDataDataContext();

   NapDL();
}

void NapDL()
{
   MyDataDataContext myDB = new MyDataDataContext();

   listImages = (from p in myDB.Faces
                 select convertByteToImage(p.FaceImage.ToArray())).ToList();
   var ListData = (from t in myDB.Faces
                   select new
                   {
                      FaceImage = convertByteToImageBgr(t.FaceImage.ToArray()),
                      Face_Name = t.Face_Name,
                   }).ToList();
   labels = (from p in myDB.Faces
             select p.Face_Name).ToList();
   NumLabels = labels.Count();
   ContTrain = NumLabels;
   dataGr.DataSource = ListData;
   dataGr.Columns[0].Width = 100;
   dataGr.Columns[1].Width = 200;
}

private Image<Gray,byte> convertByteToImage(byte[] imag)
{
   MemoryStream memoryStream = new MemoryStream(imag);
   Image returnImage = Image.FromStream(memoryStream);
   Image<Bgr, byte> tg = new Image<Bgr, byte>(((Bitmap)returnImage));
   Image<Gray, byte> kq = tg.Convert<Gray, byte>();
```

```
    return kq;
}

private void bt_Save_Click(object sender, EventArgs e)
{
    try
    {
        //Lưu ảnh vào CSDL
        MemoryStream stream = new MemoryStream();
        imagedata.Image.Save(stream, ImageFormat.Jpeg);
        MyDataDataContext myDB = new MyDataDataContext();
        Face fa = new Face();
        fa.Face_Name = txt_Name.Text;
        fa.FaceImage = stream.ToArray();
        myDB.Faces.InsertOnSubmit(fa);
        myDB.SubmitChanges();
        NapDL();
    }
    catch
    {
        MessageBox.Show("Lưu hình ảnh thất bại");
    }
}

private Image convertByteToImageBgr(byte[] imag)
{
    MemoryStream memoryStream = new MemoryStream(imag);
    Image returnImage = Image.FromStream(memoryStream);
    return returnImage;
}

private void btn_start_Click(object sender, EventArgs e)
{
    //Initialize the capture device
    capture = new Capture();
    capture.QueryFrame();
    //Initialize the FrameGraber event
    Application.Idle += new EventHandler(FrameGrabber);
    btn_start.Enabled = false;
}
```

```
        void FrameGrabber(object sender, EventArgs e)
        {
            Lb_NumberFaces.Text = "0";
            //label4.Text = "";
            NamePersons.Add("");

            //Get the current frame form capture device
            currentFrame = capture.QueryFrame().Resize(320, 240,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);

            //Convert it to Grayscale
            gray = currentFrame.Convert<Gray, Byte>();

            //Face Detector
            MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(face, 1.2, 10,
Emgu.CV.CvEnum.HAAR_DETECTION_TYPE.DO_CANNY_PRUNING, new
Size(20, 20));

                //Action for each element detected
                foreach (MCvAvgComp f in facesDetected[0])
                {
                    t = t + 1;
                    tg = currentFrame.Copy(f.rect).Convert<Gray, byte>().Resize(100, 100,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
                    //draw the face detected in the 0th (gray) channel with blue color
                    currentFrame.Draw(f.rect, new Bgr(Color.Purple), 2);


                    if (listImages.ToArray().Length != 0)
                    {
                        //TermCriteria for face recognition with numbers of trained images
like maxIteration
                        MCvTermCriteria termCrit = new MCvTermCriteria(ContTrain,
0.001);

                        //Eigen face recognizer
                        EigenObjectRecognizer recognizer = new
EigenObjectRecognizer(listImages.ToArray(), labels.ToArray(), 3000, ref termCrit);
```

```
                    name = recognizer.Recognize(tg);

                    //Draw the label for each face detected and recognized
                    currentFrame.Draw(name, ref font, new Point(f.rect.X - 2, f.rect.Y -
2), new Bgr(Color.LightGreen));
                }

                NamePersons[t-1] = name;
                NamePersons.Add("");

                //Set the number of faces detected on the scene
                Lb_NumberFaces.Text = facesDetected[0].Length.ToString();
            }
                t = 0;

                //Names concatenation of persons recognized
            for (int nnn = 0; nnn < facesDetected[0].Length; nnn++)
            {
                names = names + NamePersons[nnn] + ", ";
            }
            //Show the faces procesed and recognized
            CamImageBox.Image = currentFrame;
            lb_NameFaces.Text = names;
            names = "";
            //Clear the list(vector) of names
            NamePersons.Clear();
        }
    }
}
```

Class EigenObjectRecognizer:

```
using System;
using System.Diagnostics;
```

```
using Emgu.CV.Structure;

namespace Emgu.CV
{
   /// <summary>
   /// An object recognizer using PCA (Principle Components Analysis)
   /// </summary>
   [Serializable]
   public class EigenObjectRecognizer
   {
      private Image<Gray, Single>[] _eigenImages;
      private Image<Gray, Single> _avgImage;
      private Matrix<float>[] _eigenValues;
      private string[] _labels;
      private double _eigenDistanceThreshold;

      /// <summary>
      /// Get the eigen vectors that form the eigen space
      /// </summary>
      /// <remarks>The set method is primary used for deserialization, do not attemps to
set it unless you know what you are doing</remarks>
      public Image<Gray, Single>[] EigenImages
      {
         get { return _eigenImages; }
         set { _eigenImages = value; }
      }

      /// <summary>
      /// Get or set the labels for the corresponding training image
      /// </summary>
      public String[] Labels
      {
         get { return _labels; }
         set { _labels = value; }
      }

      /// <summary>
      /// Get or set the eigen distance threshold.
      /// The smaller the number, the more likely an examined image will be treated as
unrecognized object.
```

```
    /// Set it to a huge number (e.g. 5000) and the recognizer will always treated the examined image as one of the known object.
    /// </summary>
    public double EigenDistanceThreshold
    {
      get { return _eigenDistanceThreshold; }
      set { _eigenDistanceThreshold = value; }
    }

    /// <summary>
    /// Get the average Image.
    /// </summary>
    /// <remarks>The set method is primary used for deserialization, do not attemps to set it unless you know what you are doing</remarks>
    public Image<Gray, Single> AverageImage
    {
      get { return _avgImage; }
      set { _avgImage = value; }
    }

    /// <summary>
    /// Get the eigen values of each of the training image
    /// </summary>
    /// <remarks>The set method is primary used for deserialization, do not attemps to set it unless you know what you are doing</remarks>
    public Matrix<float>[] EigenValues
    {
      get { return _eigenValues; }
      set { _eigenValues = value; }
    }

    private EigenObjectRecognizer()
    {
    }


    /// <summary>
    /// Create an object recognizer using the specific tranning data and parameters, it will always return the most similar object
    /// </summary>
```

```
    /// <param name="images">The images used for training, each of them should be
the same size. It's recommended the images are histogram normalized</param>
    /// <param name="termCrit">The criteria for recognizer training</param>
    public EigenObjectRecognizer(Image<Gray, Byte>[] images, ref MCvTermCriteria
termCrit)
      : this(images, GenerateLabels(images.Length), ref termCrit)
    {
    }

    private static String[] GenerateLabels(int size)
    {
      String[] labels = new string[size];
      for (int i = 0; i < size; i++)
        labels[i] = i.ToString();
      return labels;
    }

    /// <summary>
    /// Create an object recognizer using the specific tranning data and parameters, it
will always return the most similar object
    /// </summary>
    /// <param name="images">The images used for training, each of them should be
the same size. It's recommended the images are histogram normalized</param>
    /// <param name="labels">The labels corresponding to the images</param>
    /// <param name="termCrit">The criteria for recognizer training</param>
    public EigenObjectRecognizer(Image<Gray, Byte>[] images, String[] labels, ref
MCvTermCriteria termCrit)
      : this(images, labels, 0, ref termCrit)
    {
    }

    /// <summary>
    /// Create an object recognizer using the specific tranning data and parameters
    /// </summary>
    /// <param name="images">The images used for training, each of them should be
the same size. It's recommended the images are histogram normalized</param>
    /// <param name="labels">The labels corresponding to the images</param>
    /// <param name="eigenDistanceThreshold">
    /// The eigen distance threshold, (0, ~1000].
```

```
    /// The smaller the number, the more likely an examined image will be treated as unrecognized object.
    /// If the threshold is &lt; 0, the recognizer will always treated the examined image as one of the known object.
    /// </param>
    /// <param name="termCrit">The criteria for recognizer training</param>
    public EigenObjectRecognizer(Image<Gray, Byte>[] images, String[] labels, double eigenDistanceThreshold, ref MCvTermCriteria termCrit)
    {
      Debug.Assert(images.Length == labels.Length, "The number of images should equals the number of labels");
      Debug.Assert(eigenDistanceThreshold >= 0.0, "Eigen-distance threshold should always >= 0.0");

      CalcEigenObjects(images, ref termCrit, out _eigenImages, out _avgImage);

      /*
      _avgImage.SerializationCompressionRatio = 9;

      foreach (Image<Gray, Single> img in _eigenImages)
        //Set the compression ration to best compression. The serialized object can therefore save spaces
        img.SerializationCompressionRatio = 9;
      */

      _eigenValues = Array.ConvertAll<Image<Gray, Byte>, Matrix<float>>(images,
        delegate(Image<Gray, Byte> img)
        {
          return new Matrix<float>(EigenDecomposite(img, _eigenImages, _avgImage));
        });

      _labels = labels;

      _eigenDistanceThreshold = eigenDistanceThreshold;
    }

    #region static methods
    /// <summary>
    /// Caculate the eigen images for the specific traning image
```

```
    /// </summary>
    /// <param name="trainingImages">The images used for training </param>
    /// <param name="termCrit">The criteria for tranning</param>
    /// <param name="eigenImages">The resulting eigen images</param>
    /// <param name="avg">The resulting average image</param>
    public static void CalcEigenObjects(Image<Gray, Byte>[] trainingImages, ref
MCvTermCriteria termCrit, out Image<Gray, Single>[] eigenImages, out Image<Gray,
Single> avg)
    {
      int width = trainingImages[0].Width;
      int height = trainingImages[0].Height;

      IntPtr[] inObjs = Array.ConvertAll<Image<Gray, Byte>, IntPtr>(trainingImages,
delegate(Image<Gray, Byte> img) { return img.Ptr; });

      if (termCrit.max_iter <= 0 || termCrit.max_iter > trainingImages.Length)
        termCrit.max_iter = trainingImages.Length;

      int maxEigenObjs = termCrit.max_iter;

      #region initialize eigen images
      eigenImages = new Image<Gray, float>[maxEigenObjs];
      for (int i = 0; i < eigenImages.Length; i++)
        eigenImages[i] = new Image<Gray, float>(width, height);
      IntPtr[] eigObjs = Array.ConvertAll<Image<Gray, Single>, IntPtr>(eigenImages,
delegate(Image<Gray, Single> img) { return img.Ptr; });
      #endregion

      avg = new Image<Gray, Single>(width, height);

      CvInvoke.cvCalcEigenObjects(
        inObjs,
        ref termCrit,
        eigObjs,
        null,
        avg.Ptr);
    }

    /// <summary>
    /// Decompose the image as eigen values, using the specific eigen vectors
```

```
        /// </summary>
        /// <param name="src">The image to be decomposed</param>
        /// <param name="eigenImages">The eigen images</param>
        /// <param name="avg">The average images</param>
        /// <returns>Eigen values of the decomposed image</returns>
        public static float[] EigenDecomposite(Image<Gray, Byte> src, Image<Gray,
Single>[] eigenImages, Image<Gray, Single> avg)
        {
          return CvInvoke.cvEigenDecomposite(
              src.Ptr,
              Array.ConvertAll<Image<Gray, Single>, IntPtr>(eigenImages,
delegate(Image<Gray, Single> img) { return img.Ptr; }),
              avg.Ptr);
        }
        #endregion

        /// <summary>
        /// Given the eigen value, reconstruct the projected image
        /// </summary>
        /// <param name="eigenValue">The eigen values</param>
        /// <returns>The projected image</returns>
        public Image<Gray, Byte> EigenProjection(float[] eigenValue)
        {
          Image<Gray, Byte> res = new Image<Gray, byte>(_avgImage.Width,
_avgImage.Height);
          CvInvoke.cvEigenProjection(
              Array.ConvertAll<Image<Gray, Single>, IntPtr>(_eigenImages,
delegate(Image<Gray, Single> img) { return img.Ptr; }),
              eigenValue,
              _avgImage.Ptr,
              res.Ptr);
          return res;
        }

        /// <summary>
        /// Get the Euclidean eigen-distance between <paramref name="image"/> and every
other image in the database
        /// </summary>
        /// <param name="image">The image to be compared from the training
images</param>
```

```
    /// <returns>An array of eigen distance from every image in the training images</returns>
    public float[] GetEigenDistances(Image<Gray, Byte> image)
    {
      using (Matrix<float> eigenValue = new Matrix<float>(EigenDecomposite(image, _eigenImages, _avgImage)))
        return Array.ConvertAll<Matrix<float>, float>(_eigenValues,
          delegate(Matrix<float> eigenValueI)
          {
            return (float)CvInvoke.cvNorm(eigenValue.Ptr, eigenValueI.Ptr, Emgu.CV.CvEnum.NORM_TYPE.CV_L2, IntPtr.Zero);
          });
    }

    /// <summary>
    /// Given the <paramref name="image"/> to be examined, find in the database the most similar object, return the index and the eigen distance
    /// </summary>
    /// <param name="image">The image to be searched from the database</param>
    /// <param name="index">The index of the most similar object</param>
    /// <param name="eigenDistance">The eigen distance of the most similar object</param>
    /// <param name="label">The label of the specific image</param>
    public void FindMostSimilarObject(Image<Gray, Byte> image, out int index, out float eigenDistance, out String label)
    {
      float[] dist = GetEigenDistances(image);

      index = 0;
      eigenDistance = dist[0];
      for (int i = 1; i < dist.Length; i++)
      {
        if (dist[i] < eigenDistance)
        {
          index = i;
          eigenDistance = dist[i];
        }
      }
      label = Labels[index];
    }
```

```
    /// <summary>
    /// Try to recognize the image and return its label
    /// </summary>
    /// <param name="image">The image to be recognized</param>
    /// <returns>
    /// String.Empty, if not recognized;
    /// Label of the corresponding image, otherwise
    /// </returns>
    public String Recognize(Image<Gray, Byte> image)
    {
      int index;
      float eigenDistance;
      String label;
      FindMostSimilarObject(image, out index, out eigenDistance, out label);

      return (_eigenDistanceThreshold <= 0 || eigenDistance <
_eigenDistanceThreshold )  ? _labels[index] : String.Empty;
    }
  }
}
```