|  |  |
| --- | --- |
| **bỘ GIÁO DỤC VÀ ĐÀO TẠO** | **BỘ NÔNG NGHIỆP VÀ PTNT** |

TRƯỜNG ĐẠI HỌC THỦY LỢI



NGUYỄN ĐÌNH DU

**XÂY DỰNG MODEL CẢNH BÁO NGỦ GẬT**

**SỬ DỤNG DLIB, RESNET**

ĐỒ ÁN TỐT NGHIỆP

HÀ NỘI, NĂM 2022

|  |  |
| --- | --- |
| **bỘ GIÁO DỤC VÀ ĐÀO TẠO** | **BỘ NÔNG NGHIỆP VÀ PTNT** |

TRƯỜNG ĐẠI HỌC THỦY LỢI

NGUYỄN ĐÌNH DU

**XÂY DỰNG MODEL CẢNH BÁO NGỦ GẬT**

**SỬ DỤNG DLIB, RESNET**

Ngành: Công Nghệ Thông Tin

Mã số:

NGUỜI HƯỚNG DẪN: Trương Xuân Nam

HÀ NỘI, NĂM 2022

**GÁY BÌA ĐỒ ÁN TỐT NGHIỆP, KHÓA LUẬN TỐT NGHIỆP**

**NGUYỄN ĐÌNH DU ĐỒ ÁN TỐT NGHIỆP HÀ NỘI, NĂM 2022**

LỜI CAM ĐOAN

Tác giả xin cam đoan đây là Đồ án tốt nghiệp của bản thân tác giả. Các kết quả trong Đồ án tốt nghiệp này là trung thực, và không sao chép từ bất kỳ một nguồn nào và dưới bất kỳ hình thức nào.Việc tham khảo các nguồn tài liệu đã được thực hiện trích dẫn và ghi nguồn tài liệu tham khảo đúng quy định.

|  |
| --- |
| **Tác giả ĐATN** |
|  |
| **Nguyễn Đình Du** |

LỜI CÁM ƠN

Trong thời gian thực hiện đồ án em đã được học hỏi nhiều kiến thức bổ ích trong giáo trình và thực tế từ thầy cô. Qua đây em xin gửi lời cảm ơn chân thành đến:

Quý thầy cô trong khoa Công nghệ thông tin của trường Đại Học Thủy Lợi đã truyền đạt kiến thức, những kinh nghiệm quý báu trong chuyên môn cũng như trong các lĩnh vực khác. Sự tận tụy say mê, lòng nhân ái, nhiệt tình của thầy cô là động lực lớn giúp chúng em trau dồi thêm kiến thức giúp chúng em vượt qua khó khăn trong học tập cũng như trong cuộc sống.

Chúng em xin gửi lời cảm ơn chân thành nhất đến thầy Trương Xuân Nam đã giúp đỡ và hướng dẫn em hoàn thành đồ án tốt nghiệp này.

Đồng thời cũng cảm ơn đến tất cả các bạn bè đã gắn bó và giúp đỡ nhau trong học tập cũng như trong suốt quá trình thực hiện đồ án tốt nghiệp.

Xin chân thành cảm ơn!

**MỤC LỤC**

[**DANH MỤC HÌNH ẢNH** v](#_Toc119940276)

[**DANH MỤC BẢNG BIỂU** vi](#_Toc119940277)

[**DANH MỤC CÁC TỪ VIẾT TẮT VÀ GIẢI THÍCH CÁC THUẬT NGỮ** vii](#_Toc119940278)

[**1.** **CHƯƠNG 1 GIỚI THIỆU** 1](#_Toc119940279)

[**2.** **CHƯƠNG 2 CƠ SỞ LÝ THUYẾT** 1](#_Toc119940280)

[**2.1.** **Tổng quan về xử lý ảnh** 1](#_Toc119940281)

[**2.1.1.** **Giới thiệu về xử lý ảnh** 1](#_Toc119940282)

[**2.1.2.** **Các thành phần cơ bản của hệ thống ảnh** 3](#_Toc119940283)

[**2.1.3.** **Các vấn đề cơ bản trong xử lý ảnh** 4](#_Toc119940284)

[**2.2.** **GIẢI THUẬT ADABOOST** 6](#_Toc119940285)

[**2.3.** **ĐẶC TRƯNG HAAR-LIKE** 8](#_Toc119940286)

[**2.4.** **GIẢI THUẬT FACIAL LANDMARKS** 12](#_Toc119940287)

[**2.4.1.** **Giới thiệu về Facial Landmarks** 13](#_Toc119940288)

[**2.4.2.** **Tìm hiểu về bộ phát hiện dấu mốc trên khuôn mặt của dlib** 14](#_Toc119940289)

[**2.5.** **KHOẢNG CÁCH EUCLIDE** 15](#_Toc119940290)

[**2.6.** **PYTHON** 16](#_Toc119940291)

[**2.6.1.** **Giới thiệu sơ lược về Python** 16](#_Toc119940292)

[**2.6.2.** **Các đặc điểm của ngôn ngữ python** 16](#_Toc119940293)

[**2.6.3.** **Ứng dụng của Python** 19](#_Toc119940294)

[**2.7.** **THƯ VIỆN DLIB** 20](#_Toc119940295)

[**3.** **CHƯƠNG 3 THỰC NGHIỆM** 22](#_Toc119940296)

[**3.1.** **SƠ ĐỒ KHỐI HỆ THỐNG** 22](#_Toc119940297)

[**3.1.1.** **Ảnh từ camera** 23](#_Toc119940298)

[**3.1.2.** **Tiền xử lý** 23](#_Toc119940299)

[**3.1.3.** **Phát hiện khuôn mặt dùng HOG** 23](#_Toc119940300)

[**3.1.4.** **Đánh dấu cấu trúc khuôn mặt dùng Facial Landmarks** 25](#_Toc119940301)

[**3.1.5.** **Trích xuất vùng mắt** 25](#_Toc119940302)

[**3.1.6.** **Tính toán tỷ lệ mắt** 25](#_Toc119940303)

[**3.1.7.** **Phát hiện ngủ gật** 27](#_Toc119940304)

[**3.1.8.** **Cảnh báo** 28](#_Toc119940305)

[**3.2.** **KẾT QUẢ** 28](#_Toc119940306)

[**4.** **CHƯƠNG 4 KẾT LUẬN** 28](#_Toc119940307)

[**TÀI LIỆU THAM KHẢO** 28](#_Toc119940308)

# **DANH MỤC HÌNH ẢNH**

[Hình 2.1. Các bước cơ bản trong xử lý ảnh. 2](#_Toc119940309)

[Hình 2.2. Các thành phần cơ bản của hệ thống xử lý ảnh 3](#_Toc119940310)

[Hình 2.3. Độ phân giải của ảnh. 4](file:///E:\drowsiness_detection\RP.docx#_Toc119940311)

[Hình 2.4. Sự khác nhau giữa ảnh màu và ảnh xám. 5](#_Toc119940312)

[Hình 2.5. Mô hình phân tầng kết hợp các bộ phân loại yếu 7](#_Toc119940313)

[Hình 2.6. Kết hợp các bộ phân loại yếu thành bộ phân loại mạnh 8](#_Toc119940314)

[Hình 2.7. Đặc trưng theo cạnh 8](#_Toc119940315)

[Hình 2.8. Đặc trưng theo đường 9](#_Toc119940316)

[Hình 2.9. Đặc trưng theo xung quanh tâm 9](#_Toc119940317)

[Hình 2.10. Đặc trưng theo đường chéo 9](#_Toc119940318)

[Hình 2.11. Cách tính Integral Image của ảnh 10](#_Toc119940319)

[Hình 2.12. Cách tính tổng giá trị pixel vùng cần tính 11](#_Toc119940320)

[Hình 2.13. Hệ thống phát hiện khuôn mặt 12](#_Toc119940321)

[Hình 2.14. 68 tọa độ đánh dấu cấu trúc khuôn mặt từ tập dữ liệu iBUG 300-W 14](#_Toc119940322)

[Hình 2.15. Thư viện DLIB 20](#_Toc119940323)

[Hình 3.1. Sơ đồ khối hệ thống 22](#_Toc119940324)

[Hình 3.2. Đánh dấu mắt bằng 6 điểm 25](#_Toc119940325)

[Hình 3.3. Hình ảnh của mốc đánh dấu mắt khi mắt mở, mắt nhắm và tỉ lệ 26](#_Toc119940326)

# **DANH MỤC BẢNG BIỂU**

# **DANH MỤC CÁC TỪ VIẾT TẮT VÀ GIẢI THÍCH CÁC THUẬT NGỮ**

# **CHƯƠNG 1 GIỚI THIỆU**

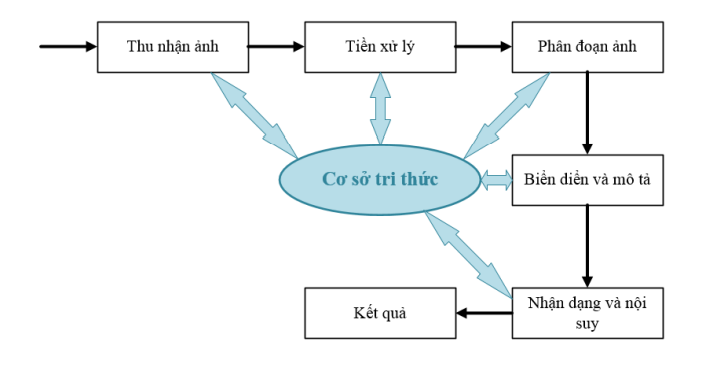
# **CHƯƠNG 2 CƠ SỞ LÝ THUYẾT**

* 1. **Tổng quan về xử lý ảnh**
     1. **Giới thiệu về xử lý ảnh**

Trong những năm gần đây, xử lý ảnh là một nghành khoa học mới mẻ nhưng tốc độ phát triển nhanh chóng và được nghiên cứu, phát triển bởi các trung tâm nghiên cứu, các trường đại học.v.v. Và với rất nhiều ứng dụng khác nhau.

Các phương pháp xử lý ảnh bắt đầu từ những ứng dụng chính như: nâng cao chất lượng độ sáng và độ phân giải của ảnh, phân tích ảnh. Ứng dụng đầu tiên được biết đến đó chính là nâng cao chất lượng hình ảnh báo được truyền qua cáp từ Luân Đôn đến New York từ những năm 1920. Càng về sau, nhờ sự xuất hiện và phát hiện mạnh mẽ của máy tính đã tạo điều kiện cho các quá trình thực hiện các thuật toán xử lý ảnh được nâng cao và phát triển hơn. Các ứng dụng của xử lý ảnh càng được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau như: khôi phục hình ảnh, chỉnh sửa, điều chỉnh độ phân giải; trong lĩnh vực y tế; trong do thám, thám hiểm; truyền và mã hóa; thị giác máy tính, robot; xử lý màu; lĩnh vực nhận dạng.v.v.

Các bước cần thiết trong xử lý ảnh. Trước đây, hình ảnh được thu từ camera là các ảnh tương tự. Gần đây, với sự phát triển không ngừng của công nghệ, ảnh màu hoặc ảnh đen trắng được lấy từ camera, sau đó được chuyển trực tiếp qua ảnh số để dễ dàng cho các bước xử lý tiếp theo. Dưới đây sẽ mô tả về các bước trong xử lý ảnh.



Hình 2.1. Các bước cơ bản trong xử lý ảnh.

**Thu nhận ảnh**: Ảnh được nhận qua camera màu hoặc trắng đen. Thông thường ảnh được nhận qua camera, video, máy scan.v.v.

**Tiền xử lý**: Sau bộ thu nhận ảnh, hình ảnh có thể có độ tương phản thấp nên cần đưa vào bộ tiền xử lý để nâng cao chất lượng hình ảnh. Bộ tiền xử lý có chức năng lọc nhiễu, nâng độ tương phản để làm cho ảnh rõ hơn và sắc nét hơn.

**Phân đoạn ảnh**: Còn gọi là phân vùng ảnh. Là tách một ảnh đầu vào thành các vùng thành phần nhỏ hơn để biểu diễn phân tích và nhận dạng ảnh.

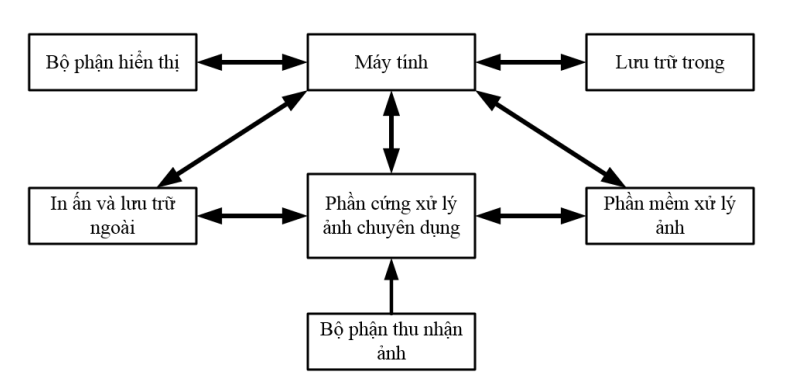
**Biểu diễn ảnh và mô tả ảnh**: Ảnh đã được phân loại chứa nhiều điểm ảnh của vùng ảnh. Việc biến đổi các số liệu này thành dạng thích hợp cho việc xử lý tiếp theo của máy tính. Chúng ta phải tìm các vùng đặc trưng của ảnh, tách các đặc tính của ảnh dưới dạng các thông tin định lượng hoặc để làm cơ sở cho sự phân biệt giữa lớp đối tượng này với lớp đối tượng khác trong phạm vi của ảnh mà chúng ta nhận được.

**Nhận dạng và nội suy ảnh**: Nhận dạng ảnh là quá trình xác định ảnh. Bằng cách so sánh mẫu với mẫu chuẩn đã được lưu trữ từ trước. Nội suy là phán đoán theo ý nghĩa trên cơ sở nhận dạng. Các mô hình toán học về ảnh được phân loại với hai dạng cơ bản:

* Nhận dạng theo tham số.
* Nhận dạng theo cấu trúc.

**Cơ sở tri thức:** Như đã biết, ảnh là một đối tượng phức tạp về đường nét, độ sáng tối, dung lượng điểm ảnh. Trong nhiều quá trình xử lý và phân tích ảnh. Ngoài việc đơn giản hóa các phương pháp toán học để đảm bảo tiện lợi cho xử lý, người ta mong muốn bắt chước quy trình tiếp nhận và xử lý ảnh theo phương pháp trí tuệ con người. Cho nên, cơ sở tri thức được phát huy và được xử lý theo pháp trí tuệ con người ở nhiều khâu khác nhau.

* + 1. **Các thành phần cơ bản của hệ thống ảnh**



Hình 2.2. Các thành phần cơ bản của hệ thống xử lý ảnh

**Bộ phận thu nhận ảnh:** Máy quay (Camera), máy quét (scaners) chuyên dụng, các bộ cảm biến ảnh.

**Phần cứng xử lý ảnh chuyên dụng:** Bộ số hóa (chuyển đổi ảnh truyền thống từ bên ngoài thành dạng dữ liệu số mà máy tính có thể hiểu được). Phần cứng thực hiện các thao tác cơ bản để nâng cao tốc độ xử lý ảnh.

**Máy tính:** Thiết bị thông thường hoặc chuyên dụng.

**Bộ phận lưu trữ:** Bắt buộc phải có. Lưu trữ tạm thời để phục vụ và sử dụng cho quá trình xử lý hiện tại. Lưu trữ vĩnh viễn là lưu trữ dữ diệu, truy cập không thường xuyên.

**Bộ phận hiện thị:** Màn hình máy tính...

**In ấn:** Ghi lại ảnh: máy in, máy chiếu...

* + 1. **Các vấn đề cơ bản trong xử lý ảnh**
       1. *Ảnh và điểm ảnh*

Ảnh số là một tập hợp nhiều điểm ảnh, mỗi điểm ảnh được gọi là một pixel. Điểm ảnh là một phần tử của ảnh số tại tọa độ (x, y) biểu diễn một màu sắc nhất định (có thể là độ xám với ảnh đen trắng). Mỗi điểm ảnh được xem như là một chấm nhỏ li ti trong một tấm ảnh. Bằng phương pháp đo lường và thống kê một lượng lớn các điểm ảnh, chúng ta có thể tái cấu trúc các điểm ảnh này thành một ảnh mới gần giống với ảnh ban đầu.

* + - 1. *Độ phân giải của ảnh*

Độ phân giải của ảnh là mật độ điểm ảnh được ấn định trên một ảnh số được hiện thị. Khoảng cách giữa các điểm ảnh sao cho mắt người vẫn thấy được sự liên tục của ảnh. Độ phân giải được phân bố theo trục x và y trong không gian hai chiều. Với cùng một ảnh, độ phân giải càng cao thì ảnh càng chứa nhiều thông tin và sắc nét hơn. Ví dụ như hình bên dưới.

Hình 2.3. Độ phân giải của ảnh.

Ở hình a có độ phân giải là 960x640 pixels, hình b có độ phân giải là 220x147 pixels. Cho thấy rằng, với độ phân giải càng cao thì độ sắc nét của ảnh càng cao và hình ảnh càng rõ nét.

* + - 1. *Mức xám của ảnh*

Một điểm ảnh có hai đặc trưng cơ bản đó chính là vị trí (x, y) của điểm ảnh và độ xám của ảnh. Mức xám của điểm ảnh là cường độ sáng của nó được gán bằng giá trị số tại thời điểm đó.



Hình 2.4. Sự khác nhau giữa ảnh màu và ảnh xám.

Với hình a là ảnh màu và hình b là ảnh xám.

* + - 1. *Biến đổi ảnh*

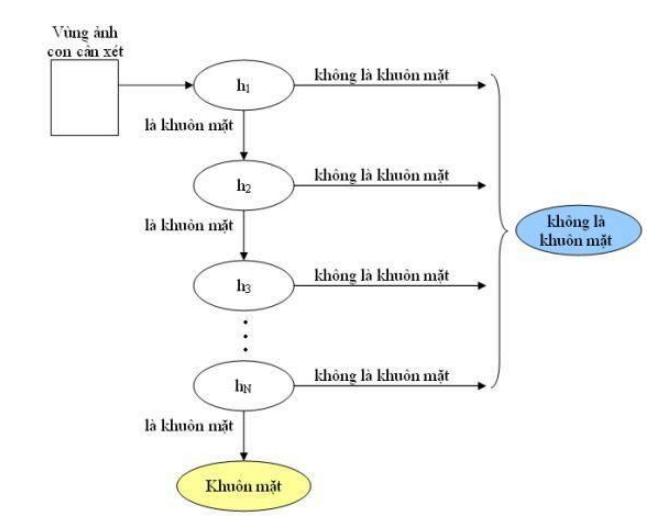
Trong xử lý ảnh do số điểm ảnh lớn hơn các tính toán nhiều (độ phức tạp tính toán cao) đòi hỏi dung lượng bộ nhớ lớn, thời gian tính toán lâu. Các phương pháp khoa học kinh điển áp dụng cho xử lý ảnh hầu như khó khả thi. Người ta sử dụng các phép toán tương đương hoặc biến đổi sang miền xử lý khác để dể tính toán, sau khi đã xử lý dể dàng, dùng biến đổi ngược để đưa về miền xác định ban đầu, các biến đổi thường gặp trong xử lý ảnh bao gồm:

* Biến đổi Fourier, Cosin, Sin.
* Biến đổi (mô tả) ảnh bằng tích chập, tích Kronecker.
* Các biển đổi khác như KL (Karhumen Loeve), Hadamard.
  1. **GIẢI THUẬT ADABOOST**

AdaBoost là một bộ phân loại mạnh phi tuyến phức dựa trên hướng tiếp cận boosting được Freund và Schapire đưa ra vào năm 1995. Adaboost cũng hoạt động trên nguyên tắc kết hợp tuyến tính các “weak classifiers” để hình thành một “strong classifier”.

Là một cải tiến của tiếp cận boosting, AdaBoost sử dụng thêm khái niệm trọng số (weight) để đánh dấu các mẫu khó nhận dạng. Trong quá trình huấn luyện, cứ mỗi “weak classifier” được xây dựng, thuật toán sẽ tiến hành cập nhật lại trọng số để chuẩn bị cho việc xây dựng “weak classifier” kế tiếp: tăng trọng số của các mẫu bị nhận dạng sai và giảm trọng số của các mẫu được nhận dạng đúng bởi “weak classifier” vừa xây dựng. Bằng cách này “weak classifer” sau có thể tập trung vào các mẫu mà các “weak classifiers” trước nó làm chưa tốt. Sau cùng, các “weak classifers” sẽ được kết hợp tùy theo mức độ tốt của chúng để tạo nên “strong classifier”.

Viola và Jones dùng AdaBoost kết hợp các bộ phân loại yếu sử dụng các đặc trưng Haar-like theo mô hình phân tầng (cascade) như sau:



Hình 2.5. Mô hình phân tầng kết hợp các bộ phân loại yếu

Trong đó, hk là các bộ phân loại yếu, được biểu diễn như sau:

x: cửa sổ con cần xét

Ok: ngưỡng (O = teta)

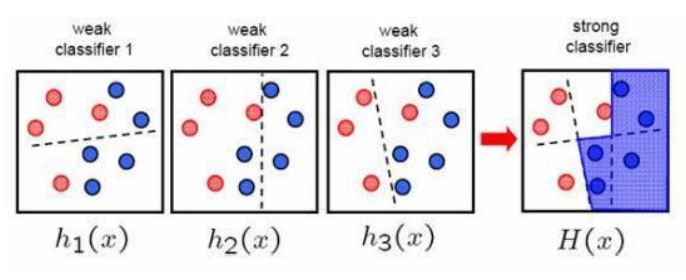
fk: giá trị của đặc trưng Haar-like

pk: hệ số quyết định chiều của phương trình

AdaBoost sẽ kết hợp các bộ phân loại yếu thành bộ phân loại mạnh như sau:

*H(x) = sign(a1h1(x) +a2h2(x) + ... + anhn(x)) (a = alpha)*

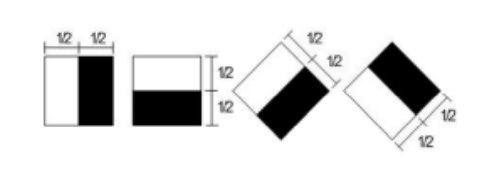
Với: at >= 0 là hệ số chuẩn hoá cho các bộ phân loại yếu



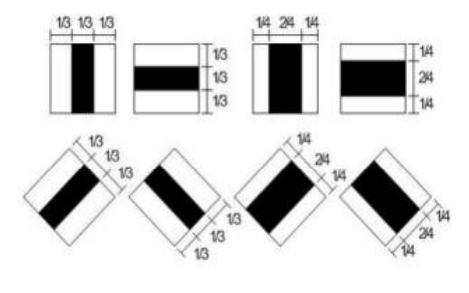
Hình 2.6. Kết hợp các bộ phân loại yếu thành bộ phân loại mạnh

* 1. **ĐẶC TRƯNG HAAR-LIKE**

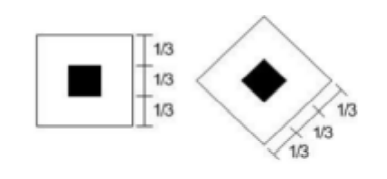
Đặc trưng Haar-like được sử dụng trong việc nhận dạng đối tượng trong ảnh số được phát biểu bởi Viola và Jones gồm 4 đặc trưng cơ bản để xác định một đối tượng trong ảnh. Mỗi đặc trưng Haar-like là sự kết hợp gồm 2 hoặc 3 khối chữ nhật mang giá trị “đen” hoặc “trắng”. Những khối chữ nhật này thể hiện sự liên hệ tương quan giữa các bộ phận trong ảnh mà bản thân từng giá trị pixel không thể diễn đạt được.



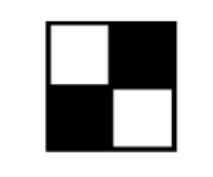
Hình 2.7. Đặc trưng theo cạnh



Hình 2.8. Đặc trưng theo đường



Hình 2.9. Đặc trưng theo xung quanh tâm



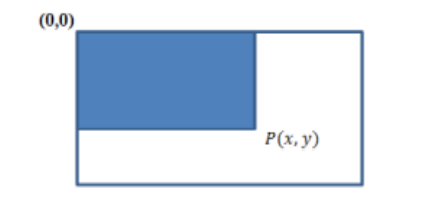
Hình 2.10. Đặc trưng theo đường chéo

Dựa theo các đặc trưng trên, giá trị của đặc trưng Haar-like được xây dựng bởi độ chênh lệch giữa tổng các pixel của các vùng đen so với tổng các pixel của các vùng trắng.

Dưới đây là công thức tính giá trị đặc trưng Haar-like:

*F(x) = ∑vùng đen(pixel) - ∑vùng trắng(pixel)*

Như vậy để tính giá trị đặc trưng Haar-like cần phải thực hiện tính toán tổng các vùng pixel trên ảnh. Điều này làm cho chi phí bài toán lớn không thể đáp ứng các tính năng yêu cầu thời gian thực. Do vậy Viola và Jones đã đề xuất ra khái niệm “Integral Image” để giảm thiểu chi phí cho bài toán tính giá trị của đặc trưng Haar-like để bài toán có thể xử lý với thời gian thực. Tính “Integral Image” bằng cách sử dụng mảng 2 chiều với kích thước bằng kích thước của ảnh cần tính giá trị đặc trưng Haar-like. Ảnh chia nhỏ ở vị trí (x, y) được tính bằng tổng các giá trị pixel của vùng từ vị trí (0,0) đến vị trí (x−1, y−1). Việc tính toán đơn giản là thực hiện phép cộng số nguyên nên tốc độ thực hiện được tối ưu hóa.



Hình 2.11. Cách tính Integral Image của ảnh

*𝑃(𝑥, 𝑦) = ∑ x’ ≤ x, y’ ≤ y (x’, y’)*

Kết quả có được sau khi tính Integral Image, việc tính tổng giá trị pixel trong vùng cần tính thực hiện như sau:

Gọi vùng cần tính tổng các giá trị pixel là vùng “A4”.

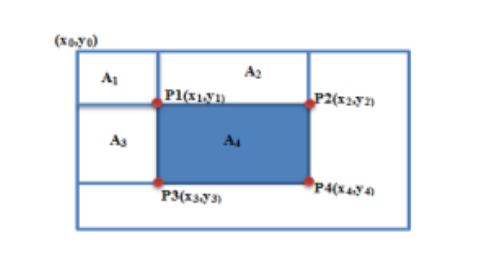
𝑃1 (𝑥1 , 𝑦1 ) = 𝐴1

𝑃2 (𝑥2 , 𝑦2 ) = 𝐴1 + 𝐴2

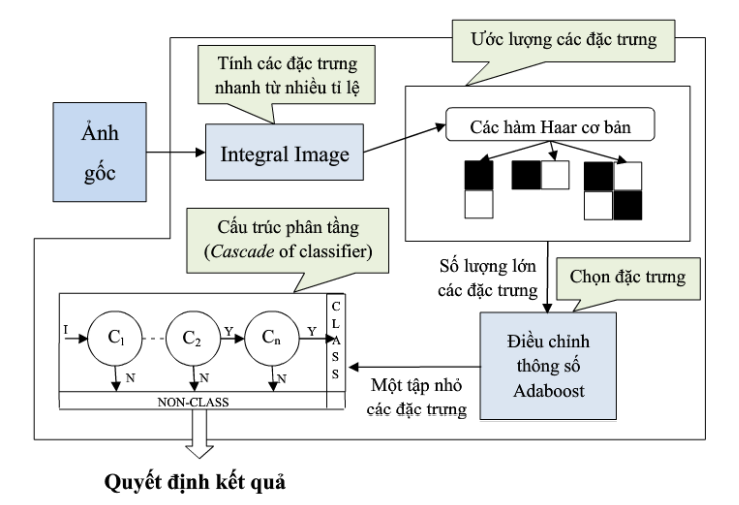
𝑃3 (𝑥3 , 𝑦3 ) = 𝐴1 + 𝐴3

𝑃4 (𝑥4 , 𝑦4 ) = 𝐴1 + 𝐴2 + 𝐴3 + 𝐴4

Ta được: 𝐴4 = 𝑃4 + 𝑃1 − 𝑃2 − 𝑃3



Hình 2.12. Cách tính tổng giá trị pixel vùng cần tính



Hình 2.13. Hệ thống phát hiện khuôn mặt

Từ ảnh gốc ban đầu, hệ thống sẽ chia ảnh thành nhiều tỉ lệ rất nhỏ để tính các đặc trưng qua các hàm đặc trưng cơ bản Haar-like. Sau khi tính toán được các hàm đặc trưng, ta nhận được một số lượng rất lớn các đặc trưng. Thì để lựa chọn và xác định có thể là khuôn mặt, ta dùng giải thuật Adaboost để lựa chọn và kết hợp các vùng trọng số yếu thành trọng số mạnh. Cuối cùng, các vùng đã được lựa chọn sẽ được đưa vào bộ phân loại “Cascade of classifier” để loại bỏ các vùng không phải khuôn mặt và xác định vùng nào là khuôn mặt. Và đưa ra kết quả cuối cùng.

* 1. **GIẢI THUẬT FACIAL LANDMARKS**

Chúng ta sẽ sử dụng dlib và OpenCV để phát hiện các điểm đánh dấu các bộ phận trên khuôn mặt trong một hình ảnh. Điểm đánh dấu các bộ phận trên khuôn mặt được sử dụng để định vị và trình bày cho các vùng nổi bật trên mặt, chẳng hạn như: Mắt, lông mày, mũi, miệng, đường viền khuôn mặt.

* + 1. **Giới thiệu về Facial Landmarks**

Facial Landmarks là một tập con của vấn đề dự đoán hình dạng. Với một hình ảnh ngõ vào (và thông thường là một ROI chỉ định đối tượng quan tâm trong ảnh), một dự đoán hình dạng cố gắng định vị điểm quan trọng theo hình dạng. Mục tiêu là phát hiện các cấu trúc khuôn mặt quan trọng trên mặt bằng các phương pháp dự đoán hình dạng. Việc phát hiện các điểm đánh dấu trên mặt là một quá trình gồm hai bước:

* Bước 1: Định vị khuôn mặt trong hình ảnh.
* Bước 2: Phát hiện các cấu trúc mặt chính trên mặt ROI

Phát hiện khuôn mặt (Bước 1) có thể đạt được theo nhiều cách:

* Chúng ta có thể sử dụng thuật toán Haar Cascade được xây dựng trong OpenCV.
* Chúng ta có thể áp dụng một máy phát hiện dùng thuật toán HOG kết hợp Linear

SVM đã được huấn luyện đặc biệt cho nhiệm vụ phát hiện khuôn mặt.

Trong cả hai trường hợp, thuật toán thực tế được sử dụng để phát hiện khuôn mặt trong hình ảnh mục đích là để thông qua các phương pháp đó chúng tôi có được hộp bao quanh khuôn mặt (tức là, các tọa độ của (x, y) của khuôn mặt trong hình ảnh).

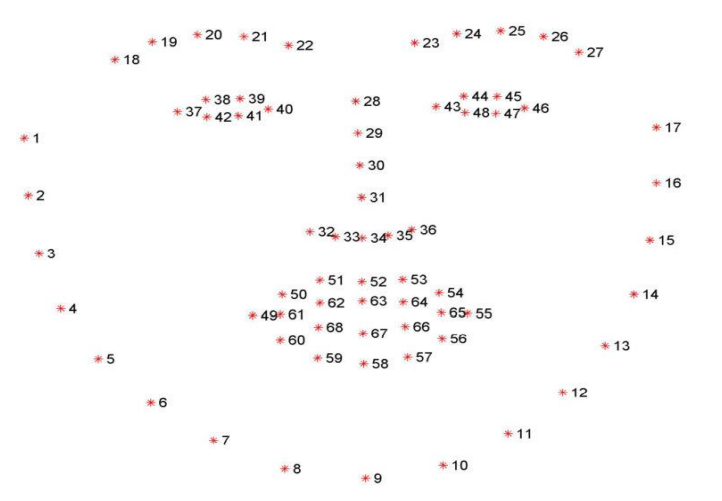
Khi đã có vùng mặt, chúng ta có thể áp dụng Bước 2 để phát hiện các cấu trúc mặt chính ở vùng mặt. Có rất nhiều bộ phát hiện các điểm đánh dấu bộ phận trên khuôn mặt nhưng tất cả các phương pháp chủ yếu là cố gắng để định vị và gắn nhãn các khu vực mặt sau đây: Miệng, lông mày phải, lông mày trái, mắt phải, mắt trái, mũi, hàm. Máy phát hiện các dấu mốc trên khuôn mặt trong thư viện dlib là việc thực hiện tài liệu One Millisecond Face Alignment with an Ensemble of Regression Trees của Kazemi và Sullivan (2014).

Phương pháp này bắt đầu bằng cách sử dụng tập huấn luyện các điểm đánh dấu trên khuôn mặt được gắn nhãn trên một hình ảnh. Những hình ảnh này được dán nhãn theo cách thủ công, chỉ định các tọa độ cụ thể (x, y) của các vùng xung quanh mỗi cấu trúc mặt. Cụ thể hơn hết là xác suất về khoảng cách giữa các cặp điểm ảnh đầu vào. Với dữ liệu huấn luyện này, một nhóm các cây hồi qui được đào tạo để ước lượng các vị trí dấu mốc trên khuôn mặt trực tiếp từ các cường độ điểm ảnh (nghĩa là không có "trích xuất các đặc trưng" nào đang diễn ra). Kết quả cuối cùng là máy phát hiện điểm đánh dấu trên mặt có thể được sử dụng để phát hiện trong thời gian thực với các dự đoán chất lượng cao.

* + 1. **Tìm hiểu về bộ phát hiện dấu mốc trên khuôn mặt của dlib**

Bộ phát hiện các dấu mốc trên khuôn mặt đã được huấn luyện trước trong thư viện dlib được sử dụng để ước tính vị trí của 68 tọa độ (x, y) các cấu trúc mặt trên mặt.

Các chỉ số của 68 tọa độ có thể được hình dung trên hình dưới đây:



Hình 2.14. 68 tọa độ đánh dấu cấu trúc khuôn mặt từ tập dữ liệu iBUG 300-W

Các chú thích này là một phần của bộ dữ liệu iBUG 300-W 68 điểm mà bộ dự đoán mốc mặt trên dlib đã được huấn luyện. Điều quan trọng cần lưu ý để thêm sinh động cho máy phát hiện mốc mặt là mô hình điểm 194 có thể được huấn luyện về bộ dữ liệu HELEN.

Bất kể bộ dữ liệu nào được sử dụng, cùng một khuôn khổ dlib có thể được tận dụng để đào tạo một dự đoán hình dạng trên dữ liệu huấn luyện đầu vào. Dấu mốc trên khuôn mặt đã được áp dụng thành công để căn chỉnh mặt, ước tính đầu người, trao đổi khuôn mặt, phát hiện chớp mắt và nhiều hơn nữa.

* 1. **KHOẢNG CÁCH EUCLIDE**

Trong toán học, khoảng cách Euclide là khoảng cách giữa hai điểm mà người ta có thể đo bằng thước, và được tính bằng công thức Pytago.

Khoảng cách Euclide giữa hai điểm A và B là chiều dài đoạn thẳng 𝐴𝐵. Trong hệ tọa độ Descartes, nếu A = (A1, A2,…, An) và B = (B1, B2,…, Bn) là hai điểm trong không gian Euclide n chiều thì khoảng cách từ A đến B bằng:

*d(A, B) = =*

Dạng chuẩn Euclide là khoảng cách của một điểm đến điểm gốc trong không gian Euclide:

*||A|| =*

Trong đó phương trình sau cùng là tích vô hướng. Đây là chiều dài của p, khi ta xem nó là một Véc-tơ Euclide có gốc nằm ở gốc tọa độ. Khoảng cách khi đó bằng:

*||A – B || =*

* 1. **PYTHON**
     1. **Giới thiệu sơ lược về Python**

Python là một ngôn ngữ lập trình hướng đối tượng và rất thông dụng. Bộ thông dịch có thể được dùng một cách tương tác, làm cho việc thử nghiệm các tính năng của ngôn ngữ trở nên dễ dàng. Được tạo ra bởi Guido van Rossum tại Amsterdam vào năm 1990. Python hoàn toàn tạo kiểu động và dùng cơ chế cấp phát bộ nhớ tự động. Python được phát triển trong một dự án mã mở, do tổ chức phi lợi nhuận Python Software Foundation quản lý. Python là một ngôn ngữ lập trình mạnh và dễ học. Python có tính hiệu quả rất cao với các cấu trúc dữ liệu và đơn giản nhưng lại rất hữu dụng với ngôn ngữ lập trình hướng đối tượng. Cú pháp Python gọn gàng và có tính năng gõ động, cùng với tính diễn dịch tự nhiên làm Python là một ngôn ngữ lý tưởng dành cho viết kịch bản và phát triển ứng dụng nhanh chóng trong nhiều lĩnh vực trên nhiều nền tảng khác nhau.

* + 1. **Các đặc điểm của ngôn ngữ python**

Dễ học, dễ đọc: Python được thiết kế để trở thành một ngôn ngữ dễ học, mã nguồn dễ đọc, bố cục rõ ràng, dễ hiểu.

Từ khóa: Python tăng cường sử dụng từ khóa tiếng Anh, hạn chế các kí hiệu và cấu trúc, cú pháp so với các ngôn ngữ khác. Python là một ngôn ngữ phân biệt kiểu chữ HOA, chữ thường.

Khối lệnh: Trong các ngôn ngữ khác, khối lệnh thường được đánh dấu bằng cặp kí hiệu hoặc từ khóa. Ví dụ, trong C/C++, cặp ngoặc nhọn { } được dùng để bao bọc một khối lệnh. Ngược lại, Python có một cách rất đặc biệt để tạo khối lệnh, đó là thụt các câu lệnh trong khối vào sâu hơn (về bên phải) so với các câu lệnh của khối lệnh cha chứa nó. Ta có thể dùng dấu Tab để thụt các khối lệnh vào để dễ dàng viết lệnh hơn.

Khả năng mở rộng: Python có thể được mở rộng. Nếu ta biết sử dụng C, ta có thể dễ dàng viết và tích hợp vào Python nhiều hàm tùy theo nhu cầu. Các hàm này sẽ trở thành hàm xây dựng sẵn (built-in) của Python. Ta cũng có thể mở rộng chức năng của trình thông dịch, hoặc liên kết các chương trình Python với các thư viện chỉ ở dạng nhị phân (như các thư viện đồ họa do nhà sản xuất thiết bị cung cấp). Hơn thế nữa, ta cũng có thể liên kết trình thông dịch của Python với các ứng dụng viết từ C và sử dụng nó như là một mở rộng hoặc một ngôn ngữ dòng lệnh hỗ trợ cho ứng dụng đó.

Trình thông dịch: Python là một ngôn ngữ lập trình dạng thông dịch, vì vậy nên Python tiết kiệm được thời gian phát triển ứng dụng vì không cần phải thực hiện biên dịch và liên kết. Trình thông dịch có thể được sử dụng để chạy file script, hoặc cũng có thể được sử dụng theo cách tương tác. Ở chế độ tương tác, trình thông dịch Python tương tự shell của các hệ điều hành họ Unix. Tại đó, ta có thể nhập vào từng biểu thức rồi gõ Enter, và kết quả thực thi sẽ được hiển thị ngay lập tức. Đặc điểm này rất hữu ích cho người mới học, giúp họ nghiên cứu tính năng của ngôn ngữ; hoặc để các lập trình viên chạy thử mã lệnh trong suốt quá trình phát triển phần mềm. Ngoài ra, cũng có thể tận dụng đặc điểm này để thực hiện các phép tính như với máy tính bỏ túi.

Lệnh và cấu trúc điều khiển: Mỗi câu lệnh trong Python nằm trên một dòng mã nguồn. Ta không cần phải kết thúc câu lệnh bằng bất kì kí tự gì. Như các ngôn ngữ khác, Python cũng có các cấu trúc điều khiển.

Chúng bao gồm:

* Cấu trúc rẽ nhánh: Cấu trúc if (có thể sử dụng thêm elif hoặc else), dùng để thực thi có điều kiện một khối mã cụ thể.
* Lệnh while: Thực thi lặp đi lặp lại các lệnh hoặc phần thân của vòng lặp miễn là điều kiện đã cho là true. Khi điều kiện là false, thì điều khiển sẽ thoát ra khỏi vòng lặp.
* Vòng lặp for: Lặp qua từng phần tử của một dãy, mỗi phần tử sẽ được đưa vào biến cục bộ để sử dụng với khối mã trong vòng lặp.

Python cũng có từ khóa class dùng để khai báo lớp (sử dụng trong lập trình hướng đối tượng) và lệnh def dùng để định nghĩa hàm

Hệ thống kiểu dữ liệu: Python sử dụng hệ thống kiểu duck typing, còn gọi là latent typing (tự động xác định kiểu). Có nghĩa là, Python không kiểm tra các ràng buộc về kiểu dữ liệu tại thời điểm dịch, mà là tại thời điểm thực thi. Khi thực thi, nếu một thao tác trên một đối tượng bị thất bại, thì có nghĩa là đối tượng đó không sử dụng một kiểu thích hợp. Python cũng là một ngôn ngữ định kiểu mạnh. Nó cấm mọi thao tác không hợp lệ. Ở Python, ta không cần phải khai báo biến. Biến được xem là đã khai báo nếu nó được gán một giá trị lần đầu tiên. Căn cứ vào mỗi lần gán, Python sẽ tự động xác định kiểu dữ liệu của biến.

Module: Python cho phép chia chương trình thành các module để có thể sử dụng lại trong các chương trình khác. Python cũng cung cấp sẵn một tập hợp các modules chuẩn mà lập trình viên có thể sử dụng lại trong chương trình của họ. Các module này cung cấp nhiều chức năng hữu ích, như các hàm truy xuất tập tin, các lời gọi hệ thống, trợ giúp lập trình mạng.

Đa năng: Python là một ngôn ngữ lập trình đơn giản nhưng rất hiệu quả.

* So với Unix shell, Python hỗ trợ các chương trình lớn hơn và cung cấp nhiều cấu trúc hơn.
* So với C, Python cung cấp nhiều cơ chế kiểm tra lỗi hơn. Nó cũng có sẵn nhiều kiểu dữ liệu cấp cao, ví dụ như các mảng (array) linh hoạt và từ điển (dictionary) mà ta sẽ phải mất nhiều thời gian nếu viết bằng C.

Python là một ngôn ngữ lập trình cấp cao có thể đáp ứng phần lớn yêu cầu của lập trình viên:

* Python thích hợp với các chương trình lớn hơn cả AWK và Perl.
* Python được sử dụng để lập trình Web. Nó có thể được sử dụng như một ngôn ngữ kịch bản.
* Python được tích hợp sẵn nhiều công cụ và có một thư viện chuẩn phong phú, Python cho phép người dùng dễ dàng tạo ra các dịch vụ Web, sử dụng các thành phần COM hay CORBA, hỗ trợ các loại định dạng dữ liệu Internet như email, HTML, XML và các ngôn ngữ đánh dấu khác. Python cũng được cung cấp các thư viện xử lý các giao thức Internet thông dụng như HTTP, FTP,…
* Python có khả năng giao tiếp đến hầu hết các loại cơ sở dữ liệu, có khả năng xử lí văn bản, tài liệu hiệu quả, và có thể làm việc tốt với các công nghệ Web khác.
* Python đặc biệt hiệu quả trong lập trình tính toán khoa học nhờ các công cụ Python Imaging Library, pyVTK, MayaVi 3D Visualization Toolkits, Numeric Python, ScientificPython,…
* Python có thể được sử dụng để phát triển các ứng dụng desktop. Lập trình viên có thể dùng wxPython, PyQt, PyGtk để phát triển các ứng dụng giao diện đồ họa (GUI) chất lượng cao. Python còn hỗ trợ các nền tảng phát triển phần mềm khác như MFC, Carbon, Delphi, X11, Motif, Tk, Fox, FLTK, …
* Python cũng có sẵn một unit testing framework để tạo ra các các bộ test (test suites).

Multiple paradigms (đa biến hóa): Python là một ngôn ngữ đa biến hóa (multiple paradigms). Có nghĩa là, thay vì ép buộc mọi người phải sử dụng duy nhất một phương pháp lập trình, Python lại cho phép sử dụng nhiều phương pháp lập trình khác nhau: hướng đối tượng, có cấu trúc, chức năng, hoặc chỉ hướng đến một khía cạnh. Python kiểu kiểu động và sử dụng bộ thu gom rác để quản lí bộ nhớ. Một đặc điểm quan trọng nữa của Python là giải pháp tên động, kết nối tên biến và tên phương thức lại với nhau trong suốt thực thi của chương trình.

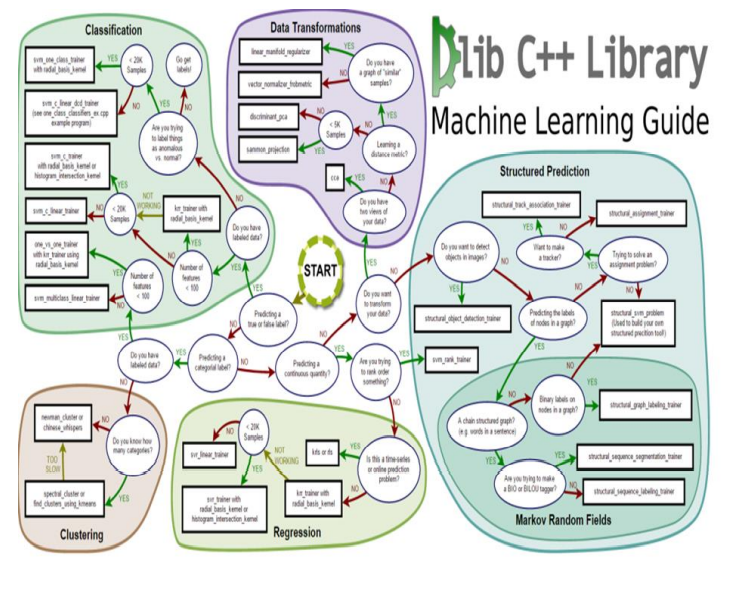
* + 1. **Ứng dụng của Python**

Python được ứng dụng trong nhiều lĩnh vực khác nhau:

* Xây dựng các tiện ích nhỏ để tự động hóa các công việc nào đó như: tự động tìm kiếm, phân loại tập tin theo tiêu chí riêng, tự động cập nhật các tập tin văn bản theo yêu cầu nào đó…
* Xây dựng ứng dụng web: Python cung cấp nhiều framework để ta có thể lựa chọn để phát triển ứng dụng web tùy theo mô hình của ứng dụng như: Django, Pyramid, Flask,…
* Lập trình các tính toán khoa học, số liệu nhờ các công cụ và lớp thư viện được xây dựng sẵn như: SciPy, IPython,…
* Lập trình ứng dụng desktop (wxWidgets), lập trình màn hình tương tác (Kivy),…

Bên cạnh đó, Python còn là ngôn ngữ lập trình được lựa chọn để giảng dạy về lập trình các khóa học nhập môn lập trình ở các trường Đại học lớn trên thế giới.

* 1. **THƯ VIỆN DLIB**



Hình 2.15. Thư viện DLIB

Ngoài thư viện OpenCV, chúng tôi còn sử dụng dlib, một thư viện mã nguồn mở khác cho việc cài đặt hệ thống. Dlib được tạo ra từ 2002 bởi tác giả Davis King, được viết trên ngôn ngữ lập trình C++.

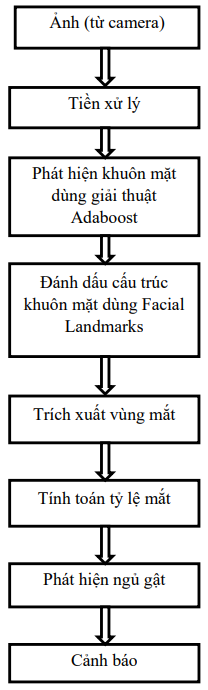
Khác với mục đích của OpenCV là cung cấp hạ tầng thuật toán cho các ứng dụng xử lý ảnh và thị giác máy tính, dlib được thiết kế cho các ứng dụng máy học và trí tuệ nhân tạo với các thư viện con chính như sau:

* Classification: các kỹ thuật phân lớp chủ yếu dựa trên hai phương pháp cơ sở là kNN và SVM
* Data transformation: các thuật toán biến đổi dữ liệu nhằm giảm số chiều, loại bỏ các dữ liệu dư thừa và tăng cường tính khác biệt (discriminant) của các đặc điểm được giữ lại
* Clustering: các kỹ thuật phân cụm
* Regression: các kỹ thuật hồi qui
* Structure prediction: các thuật toán dự đoán có cấu trúc
* Markov Random Fields: các thuật toán dựa trên các trường Markov ngẫu nhiên

Cụ thể trong bài này, phần phát hiện khuôn mặt người một cách tự động sẽ được lập trình bằng cách sử dụng thư viện dlib với dữ liệu huấn luyện và kỹ thuật hồi qui đã được cung cấp trước.

# **CHƯƠNG 3 THỰC NGHIỆM**

* 1. **SƠ ĐỒ KHỐI HỆ THỐNG**



Hình 3.1. Sơ đồ khối hệ thống

Trước tiên, ta sẽ thiết lập một camera theo dõi khuôn mặt. Để phát hiện một người có buồn ngủ hay không, ta chỉ cần vùng mắt. Một khi đã có vùng mắt, ta có thể áp dụng tỷ lệ cạnh mắt để xác định xem mắt có đóng lại hay không. Nếu đôi mắt đã được đóng lại trong khoảng thời gian đủ dài, ta có thể giả định rằng tài xế có nguy cơ buồn ngủ và báo động kịp thời để thu hút sự chú ý của tài xế.

* + 1. **Ảnh từ camera**

Để truy cập vào camera, ta cần dùng thư viện imutils – một bộ các chức năng xử lý hình ảnh giúp làm việc trên OpenCV dễ dàng hơn. Ban đầu chương trình sẽ thiết lập kết nối camera hoặc webcam máy tính và lấy từng frame ảnh để xử lý.

* + 1. **Tiền xử lý**

Chúng tôi bắt đầu vòng lặp vô hạn các khung hình trong video. Sau đó tiến hành bước tiền xử lý bằng cách thay đổi kích thước nó để có chiều rộng là 450 pixel và chuyển sang màu xám với câu lệnh trong OpenCV:

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

* + 1. **Phát hiện khuôn mặt dùng HOG**

Để tìm khuôn mặt trong hình ảnh ta cần chuyển ảnh màu thành ảnh xám. Sau đó nhìn vào mỗi pixel đơn lẻ và các điểm xung quanh nó.

Mục tiêu là tìm ra được độ tối của điểm ảnh hiện tại so với các điểm ảnh xung quanh. Sau đó vẽ một mũi tên chỉ ra hướng mà hình ảnh trở nên tối hơn.

Lặp lại quá trình đó cho mỗi pixel đơn trong ảnh, quá trình sẽ kết thúc với mỗi pixel được thay thế bằng một mũi tên. Những mũi tên này được gọi là gradients và chúng thể hiện hướng từ sáng đến tối trên toàn bộ hình ảnh.

Lý do để thay thế các điểm ảnh bằng các gradient là nếu phân tích pixel trực tiếp ảnh thật sự tối và thật sự sáng của cùng một người sẽ có các giá trị pixel khác nhau. Nhưng khi xem xét về hướng thay đổi độ sáng thì khi kết thúc quá trình ảnh thật sự sáng và thật sự tối sẽ cùng được diễn tả chính xác.

Việc lưu trữ gradient cho mỗi pixel sẽ cho quá nhiều chi tiết nên tốt hơn là chỉ cần nhìn thấy hướng cơ bản của ánh sáng hay bóng tối ở mức cao hơn để nhìn thấy các mẫu cơ bản của hình ảnh. Để thực hiện việc này, thực hiện chia nhỏ hình ảnh thành những ô vuông nhỏ kích thước 16x16 pixel. Trong mỗi hình vuông đó, thực hiện đếm gradient trong mỗi hướng chính (bao nhiêu điểm lên, điểm lên phải, điểm lên trái, điểm xuống…). Sau đó thay thế hình vuông đó trong hình ảnh bằng hướng mũi tên mạnh nhất. Kết quả cuối cùng là biến hình ảnh ban đầu thành một biểu diễn đơn giản để nắm bắt cấu trúc cơ bản của khuôn mặt một cách đơn giản.

Để phát hiện mặt người với các đặc trưng HOG, thực hiện tìm một phần của hình ảnh trong giống nhất với một mẫu HOG đã biết được trích xuất từ một loạt khuôn mặt đã được huấn luyện, chúng tôi tiến hành các bước sau:

* Bước 1: Chuẩn bị P mẫu là ảnh mặt người và trích xuất các vector đặc trưng HOG từ các bức ảnh này.
* Bước 2: Chuẩn bị N mẫu không phải ảnh mặt người (N rất lớn so với P) và trích xuất các vector HOG từ các ảnh này.
* Bước 3: Sử dụng một bộ phân loại SVM tuyến tính để học các vector của các mẫu tích cực (là ảnh mặt người) và tiêu cực (các ảnh không phải mặt người) đã chuẩn bị.
* Bước 4: Đối với mỗi bức ảnh trong bộ ảnh tiêu cực, sử dụng một cửa sổ trượt di chuyển đi qua tất cả các vị trí có thể của ảnh vào. Tại mỗi vị trí của cửa sổ trượt tính vector HOG của cửa sổ và đưa vào bộ phân lớp. Nếu bộ phân lớp sai một cửa sổ là ảnh mặt thì ghi lại vector tương ứng cùng với xác xuất phân lớp.
* Bước 5: Lấy các mẫu nhận dạng sai ở bước 4 và sắp xếp chúng theo mức xác xuất nhận dạng sai và cho bộ phân lớp học lại sử dụng các mẫu sai này.
* Bước 6: Áp dụng bộ phân lớp đã được học lại với các ảnh cần phát hiện mặt người.

Trong đề tài này, để tìm và phát hiện vị trí khuôn mặt trong ảnh, chúng tôi sử dụng hệ thống phát hiện khuôn mặt của thư viện dlib:

detector = dlib.get\_frontal\_face\_detector().

* + 1. **Đánh dấu cấu trúc khuôn mặt dùng Facial Landmarks**

Bước tiếp theo là áp dụng thuật toán đánh dấu cấu trúc với 68 điểm trên vùng mặt của thư viện dlib để định vị từng khu vực quan trọng trên khuôn mặt. Các khu vực đó bao gồm: mày, mắt, mũi, miệng và đường viền khuôn mặt.

Sau đó, chuyển đổi kết quả sang mảng NumPy.

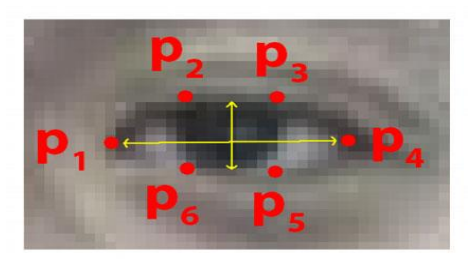
* + 1. **Trích xuất vùng mắt**

Vì đề tài tập trung vào trạng thái của mắt nên ta chỉ cần quan tâm đến vùng mắt. Sử dụng phương pháp cắt mảng NumPy, chúng ta có thể trích xuất các tọa độ (x, y) của mắt trái và mắt phải.

* + 1. **Tính toán tỷ lệ mắt**

Với các tọa độ của (x, y) cho cả hai mắt, ta sẽ tính tỷ lệ mắt, trong bài báo của Soukupová và Čech khuyến nghị sử dụng cả hai tỷ lệ cạnh mắt để có được ước tính tốt.

Mỗi mắt được đại diện bởi 6 tọa độ (x, y), bắt đầu từ góc trái của mắt (như thể bạn đang nhìn người đó), và sau đó làm việc theo chiều kim đồng hồ xung quanh phần còn lại của khu vực:



Hình 3.2. Đánh dấu mắt bằng 6 điểm

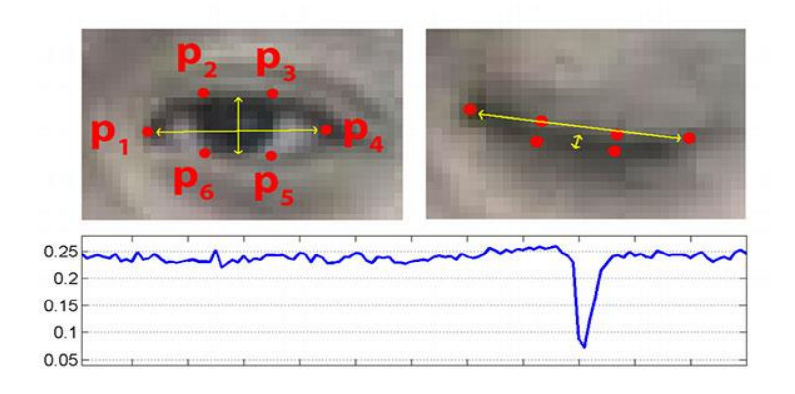
Có một mối quan hệ giữa chiều rộng và chiều cao của các tọa độ này. Dựa trên bài báo của Soukupová và Čech trong bài báo năm 2016 [4], chúng ta có thể lấy được một phương trình phản ánh mối quan hệ này gọi là tỉ số mắt (EAR):

Trong đó p1đến p6 là các vị trí đánh dấu mốc trên mắt. Tử số của phương trình này tính khoảng cách giữa các điểm mốc dọc trong khi mẫu số tính khoảng cách giữa các điểm mốc ngang, trọng số mẫu số bằng 2 là thích hợp vì chỉ có một tập các điểm ngang mà có hai bộ điểm thẳng đứng.

Tỷ số mắt là khoảng không đổi trong khi mắt mở, nhưng sẽ nhanh chóng giảm xuống không khi một nháy mắt diễn ra.

Sử dụng phương trình đơn giản này, chúng ta có thể tránh các kỹ thuật xử lý hình ảnh và chỉ đơn giản dựa vào tỷ lệ khoảng cách mốc thời gian để xác định xem một mắt đang nhắm hay không.

Để làm cho điều này rõ ràng hơn, hãy xem xét các con số sau đây từ Soukupová và Čech:



Hình 3.3. Hình ảnh của mốc đánh dấu mắt khi mắt mở, mắt nhắm và tỉ lệ

Ở phía trên bên trái của hình, chúng ta có một mắt hoàn toàn mở - tỉ lệ mắt ở đây sẽ lớn và tương đối ổn định theo thời gian.

Tuy nhiên, một khi mắt nhấp nháy (phía trên bên phải hình) tỷ lệ mắt giảm đáng kể gần bằng không.

Phát hiện ngủ gật: Đầu tiên chúng ta sẽ thiết lập một ngưỡng mắt để nhận dạng trạng thái mắt nhắm hay mắt mở. Và chúng ta sẽ chọn ngưỡng mắt là 0,2. Theo báo cáo năm 2016 trong bài Real-Time Eye Blink Detection using Facial Landmarks của Tereza Soukupová and Jan Čech [4]. Dó đó, chúng ta chọn ngưỡng mắt là 0,2. Sau đó chúng ta sẽ kiểm tra tỉ số mắt EAR đã tính toán ở trước đó. Nếu tỉ số mắt nhỏ hơn ngưỡng mắt trong một khoảng thời gian nào đó thì cho rằng người đó đang ngủ gật và gửi thông báo tới màn hình. Công việc này sẽ được thực hiện trong suốt quá trình để phát hiện trạng thái ngủ gật.

* + 1. **Phát hiện ngủ gật**

Chúng tôi bắt đầu phát hiện ngủ gật bằng việc thiết lập trước các giá trị:

* Ngưỡng của mắt NGUONG\_MAT để nhận dạng trạng thái mắt nhắm hay mở.
* Biến đếm DEM là tổng số khung liên tiếp mà người đó đã nhắm mắt.
* Số khung hình SO\_KHUNG\_HINH để nhận biết tài xế đang tỉnh táo hay đang bắt đầu ngủ gật.
* Ban đầu cảnh báo đang tắt CANH\_BAO = False.

Tiếp đến chúng tôi kiểm tra tỷ số mắt EAR đã được tính có dưới ngưỡng NGUONG\_MAT hay không để xác định mắt đóng hoặc mở. Nếu tỷ số mắt EAR được xác định nhỏ hơn ngưỡng NGUONG\_MAT thì tăng biến DEM. Nếu DEM vượt quá SO\_KHUNG\_HINH đã đặt trước thì chúng tôi giả định rằng người đó đang ngủ gật và bắt đầu bật cảnh báo. Ngược lại nếu tỷ số mắt lớn hơn ngưỡng mắt hoặc tổng số khung hình mắt nhắm liên tiếp không lớn hơn SO\_KHUNG\_HINH thì thiết lập lại DEM ban đầu =0 và tắt cảnh báo. Thực hiện lại công việc đó trong suốt quá trình thu hình để phát hiện tình trạng ngủ gật.

* + 1. **Cảnh báo**
  1. **KẾT QUẢ**

# **CHƯƠNG 4 KẾT LUẬN**

# **TÀI LIỆU THAM KHẢO**