

TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP

KHOA ĐIỆN TỬ

BỘ MÔN TIN HỌC CÔNG NGHIỆP

BÀI GIẢNG THỊ GIÁC MÁY TÍNH

(LƯU HÀNH NỘI BỘ)

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU TỔNG QUAN.....	5
1.1. Khái niệm về thị giác máy tính	5
1.2. Tại sao thị giác máy tính lại cần thiết	6
1.3. Thị giác máy tính hoạt động như thế nào	7
1.4. Sự phát triển của thị giác máy tính	7
Chương 2: Lọc ảnh	12
2.1. Lọc ảnh trong miền không gian	12
2.1.1. Hàm tích chập hai ảnh.....	12
2.1.2. Lọc trung bình (Median Filter)	14
2.1.3. Lọc trung vị	17
2.1.4. Bộ lọc Sobel	21
2.1.5. Bộ lọc Laplace.....	23
2.1.6. Cài đặt bộ lọc trong OpenCV.....	24
2.2. Lọc ảnh trong miền tần số.....	26
2.2.1. Khái niệm	26
2.2.2. Lọc thông thấp - Low Pass Filter	29
2.2.3. Lọc thông cao - High Pass Filter.....	29
2.2.4. Bộ lọc Blur	29
Chương 3: Phát hiện đường biên	32
3.1. Các phương pháp phát hiện đường biên	32
3.1.1. Phương pháp Gradient	32
3.1.2. Phương pháp Laplacian.....	34
3.2. Một số chương trình phát hiện biên ảnh	34
3.3. Phát hiện đường thẳng với Hough transform.....	40
3.3. 1. Phương trình đường thẳng trong không gian ảnh	40

3.3. 2. Hàm houghlines trong opencv	41
3.3. 3. Chương trình tìm đường tròn, đường thẳng trong ảnh bằng biến đổi Hough	42
3.4. Tìm contours trong ảnh	43
3.4.1. Contour là gì	43
3.4.2. Tìm contours trong ảnh với Opencv	44
3.4.3. Vẽ contours trong ảnh với Opencv	46
CHƯƠNG 4: PHÂN ĐOẠN ẢNH	48
4.1. Giới thiệu các phương pháp phân đoạn ảnh	48
4.2. Chọn ngưỡng cố định	49
4.3. Chọn ngưỡng dựa trên lược đồ	49
4.3. 1. Thuật toán đẳng hiệu	49
4.3. 2. Thuật toán đối xứng nền	50
4.3. 3. Thuật toán tam giác	51
4.3. 4. Chọn ngưỡng đối với Bimodal Histogram	51
4.4. Phương pháp phân đoạn dựa trên ngưỡng cục bộ thích nghi	52
4.4. 1. Phân đoạn bằng Watershed	53
4.4. 2. Tìm ngưỡng cục bộ thích nghi	55
4.5. Phân đoạn ảnh sử dụng openCV	58
4.5.1. Adaptive Thresholding (phân ngưỡng thích nghi)	59
4.5.2. Nhị phân hóa bằng thuật toán Otsu	60
CHƯƠNG 5: TRÍCH ĐẶC TRƯNG VÀ ĐỐI SÁNH ẢNH	62
5. 1. Các đặc trưng cơ bản	62
5. 2. Đặc trưng màu sắc	62
5.2. 1. Mô tả các đặc trưng về màu sắc	62
5.2. 2. Độ đo tương đồng về màu sắc	63
5. 3. Đặc trưng kết cấu	63
5.3.1. Mô tả các đặc trưng kết cấu	63
5.3.2. Độ đo tương đồng cho kết cấu ảnh	64
5. 4. Đặc trưng hình dạng	64
5.4.1. Mô tả các đặc trưng hình dạng	64
5.4.2. Độ đo tương đồng cho hình dạng	64

5.5. Đặc trưng Haar-like.....	65
5.5.1. Giới thiệu.....	65
5.5.2. Các đặc trưng Haar-Like	65
5.6. Đặc trưng cục bộ SIFT.....	67
5.6.1. Phát hiện đặc trưng cục bộ bất biến SIFT	67
5.6.2. Độ đo tương đồng cho đặc trưng cục bộ bất biến	71
CHƯƠNG 6: NHẬN DẠNG	72
6.1. Nhận dạng đối tượng.....	72
6.1.1. Rút trích đặc trưng.....	72
6.1.2. Sử dụng tập dữ liệu để phát hiện đối tượng.....	76
6.1.3. Ví dụ nhận diện biển số xe máy sử dụng OpenCV	77
6.2. Nhận dạng mặt người.....	79
6.2.1. Nhận diện khuôn mặt: Bước 1: face detection.....	79
6.2.2. Nhận diện khuôn mặt – Bước 2: face recognition	80
6.2.3. Nhận diện khuôn mặt – Bước 3: tối ưu hoá	81
6.3. Eigenfaces	84
6.3.1. Thuật toán PCA và ứng dụng trong nhận dạng mặt người.	85
6.3.2. Tìm EigenFaces.....	87
6.3.3. So sánh khoảng cách-Compare Distance.	93

CHƯƠNG 1: GIỚI THIỆU TỔNG QUAN

1.1. Khái niệm về thị giác máy tính

Thị giác máy tính là một lĩnh vực trong Artificial Intelligence (Trí tuệ nhân tạo) và Computer Science (Khoa học máy tính) nhằm giúp máy tính có được khả năng nhìn và hiểu giống như con người.

Thị giác máy tính (computer vision) được định nghĩa là một lĩnh vực bao gồm các phương pháp thu nhận, xử lý ảnh kỹ thuật số, phân tích và nhận dạng các hình ảnh từ thế giới thực để đưa ra các thông tin số. Thị giác máy tính cũng được mô tả là sự tổng thể của một dải rộng các quá trình tự động và tích hợp các thể hiện cho các nhận thức thị giác

Quá trình mô phỏng thị giác con người được chia thành 3 giai đoạn nối tiếp (tương tự cách con người nhìn): mô phỏng mắt (thu nhận), mô phỏng vỏ não thị giác (xử lý) và mô phỏng phần còn lại của bộ não (phân tích).

- **Thu nhận**

Mô phỏng mắt là lĩnh vực đạt được nhiều thành công nhất. Chúng ta đã tạo ra các cảm biến, vi xử lý hình ảnh giống khả năng nhìn của mắt người và thậm chí còn tốt hơn.

Camera có thể chụp hàng ngàn ảnh mỗi giây và nhận diện từ xa với độ chính xác cao. Tuy nhiên cảm biến camera tốt nhất cũng không thể nhận diện được một quả bóng chày dừng nói là bắt được chúng. Nói cách khác, phần cứng bị giới hạn khi không có phần mềm - đến giờ vẫn là khó khăn lớn nhất. Tuy vậy, camera ngày nay cũng khá linh hoạt và làm nền tảng tốt để nghiên cứu.

- **Xử lý**

Bộ não được xây dựng từ con số 0 với các hình ảnh dần dần lấp đầy tâm trí, nó làm nhiệm vụ liên quan tới thị giác nhiều hơn bất kì công việc nào khác và việc này đều xuống tới cấp độ tế bào. Hàng tỉ tế bào phối hợp để lấy ra các hình mẫu, bắt được tín hiệu.

Một nhóm nơ-ron sẽ báo cho nhóm khác khi có sự khác biệt dọc theo một đường thẳng (theo một góc nào đó, như chuyển động nhanh hơn hay theo một hướng khác). Nghiên cứu đầu tiên về thị giác máy tính cho rằng mạng lưới nơ-ron phức tạp tới nỗi không thể hiểu nổi khi tiếp cận theo hướng lý giải từ trên xuống dưới. Với một số đối tượng thì cách này cũng hiệu quả nhưng khi mô tả từng đối tượng, từ nhiều góc nhìn, nhiều biến thể về màu sắc, chuyển động và nhiều thứ khác thì hãy hình dung sẽ khó thế nào. Ngay cả mức

nhận thức của một em bé cũng sẽ cần lượng dữ liệu lớn vô cùng. Cách tiếp cận từ dưới lên bất chước cách não bộ hoạt động có vẻ hứa hẹn hơn.

Những năm qua chứng kiến sự bùng nổ của các nghiên cứu và sử dụng hệ thống này trong việc bắt chước não người. Quá trình nhận diện hình mẫu vẫn đang tăng tốc và chúng ta vẫn liên tục đạt được tiến bộ.

- **Phân tích**

Ta có thể xây dựng một hệ thống nhận diện được một quả táo, từ bất cứ góc nào, trong bất kì tình huống nào, dù đứng im hay chuyển động nhưng chúng không thể nhận diện được một quả cam, không thể nói cho ta quả táo là gì, có ăn được không, lớn nhỏ ra sao hay dùng để làm gì. Như vậy phần cứng và phần mềm tốt cũng không làm được gì nếu không có hệ điều hành.

Đó chính là phần còn lại của bộ não: bộ nhớ ngắn/dài hạn, dữ liệu từ các giác quan, sự chú ý, nhận thức, bài học khi tương tác với thế giới... được viết lên mạng lưới nơ-ron kết nối phức tạp hơn bất cứ thứ gì chúng ta từng thấy, theo cách mà chúng ta không thể hiểu. Đó là nơi mà khoa học máy tính và trí tuệ nhân tạo gặp mặt.

Dù mới trong thời kì sơ khai, thị giác máy tính vẫn vô cùng hữu ích. Nó có mặt trong camera nhận diện khuôn mặt (Face ID) và nụ cười. Nó giúp xe tự lái nhận diện biển báo, người đi đường. Nó nằm trong các robot trong nhà máy, nhận diện sản phẩm, truyền cho con người.

1.2. Tại sao thị giác máy tính lại cần thiết

Thị giác máy tính cho phép các máy tính cũng như robot, các phương tiện điều khiển từ máy tính và mọi thứ từ nhà máy, thiết bị nông trại đến xe hơi và máy bay có thể thực hiện một số hoạt động tự động, nó hoạt động một cách hiệu quả, thậm chí an toàn hơn.

Tầm quan trọng của nó đã trở nên rõ ràng hơn trong một thời đại kỹ thuật số. Chúng ta đã nhìn thấy được ứng dụng công nghệ này qua việc hỗ trợ người dùng tổ chức và truy cập vào bộ sưu tập hình ảnh của họ mà không cần gắn thẻ tag hoặc đánh dấu trong Google Photos. Nhưng điều đáng nói làm thế nào nó vẫn duy trì liên tục khi mà số lượng hình ảnh được chia sẻ mỗi ngày lên đến hàng tỷ. Với con người thao tác thủ công là điều không thể làm được.

Một nghiên cứu của dịch vụ in ảnh Photoworld đã cho ra một số liệu như sau: Một người sẽ mất đến 10 năm để có thể xem qua tất cả hình ảnh được chia sẻ trên snapchat (chỉ trong 1 giờ) chưa đề cập đến việc phân loại. Và dĩ nhiên trong 10 năm đó thì số lượng ảnh tương ứng cũng tăng theo cấp số nhân. Điều này cho thấy thế giới ngày nay tràn ngập

những hình ảnh kỹ thuật số và chúng ta cần những công nghệ máy tính này mới có thể xử lý được tất cả – nó đã vượt qua khả năng mà con người không thể xử lý được.

1.3. Thị giác máy tính hoạt động như thế nào

Trên một cấp độ nhất định thì đây chính là tất cả về công nghệ nhận dạng mẫu. Cách để huấn luyện cho máy tính hiểu được dữ liệu hình ảnh thực tế chính là cung cấp cho nó hình ảnh, rất nhiều hình có thể là hàng ngàn, hàng triệu được tổ chức và gắn nhãn trước.

Bước tiếp theo đó, các nhà phát triển phần mềm sẽ vẽ nên một thuật toán tuân theo các kỹ thuật phần mềm khác nhau cho phép máy tính dò tìm tất cả các mẫu theo nhiều yếu tố liên quan đến các nhãn đó.

Ví dụ, nếu bạn cung cấp cho máy tính một triệu hình ảnh về loài chim cánh cụt, **thị giác máy tính** sẽ tuân theo tất cả các thuật toán cho phép chúng phân tích màu sắc trong ảnh, các hình dạng và khoảng cách giữa các bộ phận. Khi kết thúc thuật toán, máy tính sẽ có thể ứng dụng trải nghiệm của nó nếu được cung cấp các hình ảnh không nhãn khác để định dạng những hình ảnh của chim cánh cụt.

1.4. Sự phát triển của thị giác máy tính

Các hệ thống thị giác máy tính truyền thống là sự kết hợp của các thuật toán phối hợp với nhau trong nỗ lực giải quyết các nhiệm vụ nói trên. Mục tiêu chính là trích xuất các đặc điểm (feature) từ hình ảnh như phát hiện cạnh, phát hiện góc và phân đoạn dựa trên màu. Độ chính xác của các thuật toán được sử dụng để trích xuất các đặc điểm phụ thuộc vào thiết kế và tính linh hoạt của từng thuật toán.

Vấn đề chính với cách tiếp cận này là hệ thống cần được cho biết những đặc điểm cần tìm trong hình ảnh. Về cơ bản, do thuật toán hoạt động như đã được xác định bởi nhà thiết kế thuật toán, các features được trích xuất được thiết kế bởi con người. Trong các triển khai như vậy, hiệu suất của thuật toán có thể được xử lý thông qua tinh chỉnh, chẳng hạn như bằng cách điều chỉnh các tham số hoặc sửa đổi cấp mã để điều chỉnh hành vi. Tuy nhiên, những thay đổi như thế này cần phải được thực hiện thủ công và được mã hóa cứng hoặc cố định cho một ứng dụng cụ thể.

- **Xu hướng hiện nay từ Deep Learning**

Mặc dù vẫn còn những trở ngại đáng kể trong con đường phát triển của thị giác máy tính đến “cấp độ con người”, các hệ thống Deep Learning đã đạt được tiến bộ đáng kể trong việc xử lý một số nhiệm vụ phụ có liên quan. Lý do cho sự thành công này một phần dựa trên trách nhiệm bổ sung được giao cho các hệ thống deep learning.

Điều hợp lý để nói rằng sự khác biệt lớn nhất với các hệ thống deep learning là chúng không còn cần phải được lập trình để tìm kiếm các đặc điểm cụ thể. Thay vì tìm kiếm các đặc điểm cụ thể bằng thuật toán được lập trình cẩn thận, các mạng lưới thần kinh bên trong các hệ thống deep learning được đào tạo. Ví dụ: nếu ô tô trong hình ảnh bị phân loại sai thành xe máy thì bạn không tinh chỉnh các tham số hoặc viết lại thuật toán. Thay vào đó, bạn tiếp tục đào tạo cho đến khi hệ thống làm cho đúng.

Với sức mạnh tính toán tăng lên được cung cấp bởi các hệ thống deep learning hiện đại, có sự tiến bộ ổn định và đáng chú ý hướng tới điểm mà một máy tính sẽ có thể nhận ra và phản ứng với mọi thứ mà nó nhìn thấy.

1.1. Ứng dụng của thị giác máy tính trong thực tiễn

- **Phát hiện các khiếm khuyết**

Đây có lẽ là ứng dụng phổ biến nhất của thị giác máy tính. Cho đến bây giờ thì việc phát hiện ra các yếu tố bị lỗi thường được tiến hành bởi những người giám sát chỉ định và mở rộng hơn họ không thể nào kiểm soát được toàn bộ cả một quy trình hệ thống được.

Với thị giác máy tính, chúng ta có thể kiểm tra tất cả các lỗi nhỏ nhất từ vết nứt kim loại, lỗi sơn, bản in xấu, có kích thước nhỏ hơn 0,05mm. Việc xử lý này còn nhanh và tốt hơn mắt thường của con người gấp nhiều lần. Thuật toán này được thiết kế và đào tạo đặc biệt cho từng ứng dụng cụ thể thông qua hình ảnh có khiếm khuyết và không có khuyết tật.

- **Trình đọc tự động**

Nếu bạn đã từng sử dụng ứng dụng Google translate, bạn có thể đã phát hiện ra khả năng trở camera điện thoại thông minh của bạn vào văn bản từ bất kỳ số ngôn ngữ nào và dịch nó sang ngôn ngữ khác trên màn hình gần như ngay lập tức. Sử dụng thuật toán nhận dạng ký tự (OCR) để trích xuất thông tin, cụ thể là nhận dạng ký tự quang học – cho phép một bản dịch chính xác sau đó chuyển thành lớp phủ lên văn bản thực.

- **Vận hành tự động**

Có lẽ bạn đã thấy trên tivi những chiếc xe không người lái, lĩnh vực này phụ thuộc rất nhiều vào Computer vision và Deep learning. Mặc dù chưa đến thời điểm thay thế hoàn toàn người lái, công nghệ xe tự hành đã tiến bộ đáng kể trong vài năm qua.

Công nghệ AI phân tích dữ liệu thu thập được từ hàng triệu người lái xe, học hỏi từ hành vi lái xe để tự động tìm làn đường, ước tính độ cong đường, phát hiện các mối nguy hiểm và giải thích các tín hiệu và tín hiệu giao thông.

- **Xử lý dữ liệu**

Để hỗ trợ con người thực hiện các nhiệm vụ nhận dạng và tổ chức thông tin, các công cụ Computer Vision và mô hình Deep Learning đã được đưa vào nghiên cứu, đòi hỏi khối lượng dữ liệu lớn được dán nhãn. Khi các thuật toán Deep Learning phát triển, chúng chủ yếu thay thế quy trình gán thẻ thủ công thông qua một phương pháp tiếp cận được gọi là nghiên cứu dữ liệu đám đông – thu thập theo thời gian thực tự động và gán thẻ dữ liệu do các chuyên gia tạo ra và từ đó máy học sẽ bắt đầu quy trình nhận dạng các đối tượng.

- **Lĩnh vực y tế**

Những tiến bộ lớn liên tục xuất hiện trong các lĩnh vực nhận dạng mẫu và xử lý hình ảnh. Đồng thời, không có gì đáng ngạc nhiên khi cộng đồng y tế và các chuyên gia trong lĩnh vực chăm sóc sức khỏe cho rằng hình ảnh y khoa (kỹ thuật tạo hình ảnh trực quan về bên trong của cơ thể để phân tích lâm sàng và can thiệp y tế, cũng như biểu thị trực quan chức năng của một số cơ quan hoặc mô sinh lý học) đã trở thành một phần thiết yếu trong cách thức làm việc của họ, hướng đến các công cụ chẩn đoán tốt hơn và tăng đáng kể khả năng đưa ra các hành động hiệu quả hơn.

Phân tích hình ảnh y khoa là một trợ giúp lớn cho phân tích dự đoán và trị liệu. Ví dụ, thị giác máy tính được áp dụng cho hình ảnh nội soi có thể làm tăng mức độ hợp lệ và đáng tin cậy của dữ liệu để giảm tỷ lệ tử vong liên quan đến ung thư đại trực tràng.

Trong một ví dụ khác, công nghệ thị giác máy tính cũng cung cấp hỗ trợ kỹ thuật cho phẫu thuật. Mô hình hình ảnh 3D của hộp sọ, như là một phần của điều trị khối u não, cung cấp tiềm năng to lớn trong việc chuẩn bị phẫu thuật thần kinh tiên tiến. Ngoài ra, khi mà học sâu ngày càng được sử dụng trong các công nghệ AI, việc tận dụng nó để phân loại các nốt phổi đã đạt được tiến bộ to lớn để chẩn đoán sớm ung thư phổi.

- **Bán lẻ**

Thị giác máy tính đang được sử dụng trong các cửa hàng ngày càng nhiều, đặc biệt là giúp cải thiện trải nghiệm của khách hàng. Pinterest Lens là một công cụ tìm kiếm sử dụng thị giác máy tính để phát hiện các đối tượng. Bằng cách sử dụng ứng dụng điện thoại thông minh trong các cửa hàng, bạn có thể hình dung một sản phẩm trông như thế nào và nhận được các sản phẩm khác liên quan đến nó.

Nhận dạng khuôn mặt là một ứng dụng nổi tiếng về thị giác máy tính có thể được sử dụng trong trung tâm mua sắm hoặc trong cửa hàng. Lolli & Pops, một cửa hàng kẹo có trụ sở tại Mỹ, đang sử dụng nhận dạng khuôn mặt để tích điểm cho khách hàng trung thành. “Hãy tưởng tượng: Bạn bước vào cửa hàng yêu thích của mình và nhân viên bán hàng chào đón bạn bằng tên và bất cứ lúc nào bạn cần, họ chia sẻ với bạn những sản phẩm mới

nhất của mình mà bạn có thể sẽ quan tâm nhất.” Sự đổi mới công nghệ có thể đưa ra các giới thiệu được cá nhân hóa cụ thể cho từng khách hàng.

Dường như không có giới hạn khi nói về các trường hợp sử dụng của thị giác máy tính trong lĩnh vực bán lẻ, chúng cũng có thể bao gồm phân tích các kệ hoặc tầng trong cửa hàng, thậm chí cả phân tích tâm trạng của khách hàng, cụ thể phát hiện cảm xúc dựa trên các thuật toán thông qua hình ảnh trong video và phân tích các biểu cảm nhỏ nhất trên gương mặt, xử lý chúng và cuối cùng, diễn giải cảm xúc chung.

Chấm dứt việc phải xếp hàng để thanh toán có thể là mục tiêu cuối cùng của cải tiến công nghệ trong các cửa hàng. Amazon đã phát triển một mô hình mới, Amazon Go, thúc đẩy các công nghệ bao gồm thị giác máy tính, IoT và AI để phát hiện, theo dõi và phân tích hành vi cũng như hành động của khách hàng trong cửa hàng để xử lý tự động quá trình thanh toán và gửi cho họ hóa đơn điện tử.

- **Ngân hàng**

Khi nói đến việc liên kết các công nghệ AI với ngân hàng, chúng ta chủ yếu nghĩ đến việc phát hiện gian lận. Mặc dù đó là một lĩnh vực tập trung đặc biệt cho công nghệ tiên tiến trong lĩnh vực này, thị giác máy tính có thể cải tiến nhiều thứ hơn nữa. Các ứng dụng nhận dạng hình ảnh sử dụng học máy để phân loại và trích xuất dữ liệu phục vụ cho việc giám sát quá trình xác thực các tài liệu như thẻ căn cước hoặc giấy phép lái xe có thể được sử dụng để cải thiện trải nghiệm của khách hàng từ xa và tăng cường bảo mật.

- **Phát hiện cháy dựa trên máy bay không người lái**

Việc sử dụng rộng rãi và đa dạng thị giác máy tính cũng áp dụng cho các lĩnh vực an ninh. Máy bay không người lái, hoặc UAV, có thể tận dụng các hệ thống thị giác máy tính để tăng cường khả năng phát hiện của con người trong việc phát hiện cháy rừng, sử dụng hình ảnh hồng ngoại (IR) như một phần của các giao thức giám sát cháy rừng. Các thuật toán nâng cao phân tích các đặc điểm hình ảnh video như chuyển động hoặc độ sáng để phát hiện lửa. Hệ thống đang thực hiện các trích xuất được nhắm mục tiêu để phát hiện dễ dàng các mẫu và tính toán cách để thấy sự khác biệt giữa các đám cháy và chuyển động thực tế có thể bị hiểu nhầm là hỏa hoạn.

Máy bay không người lái cũng có thể cải thiện an ninh và hiệu quả của hoạt động cứu hỏa bằng cách giám sát hoặc nghiên cứu các khu vực nguy hiểm. Nhân viên cứu hỏa có thể chạy các phân tích dựa trên thuật toán tiên tiến để kiểm tra khói và lửa, từ đó đánh giá rủi ro và đưa ra dự đoán về sự lan truyền lửa.

- **Nhận diện khuôn mặt**

Nhận diện khuôn mặt lập bản đồ và lưu trữ danh tính kỹ thuật số nhờ vào các thuật toán học sâu. Loại nhận dạng sinh trắc học này có thể được so sánh với các công nghệ nhận dạng giọng nói, móng mắt hoặc dấu vân tay hiện đang rất phổ biến.

Khái niệm này xuất hiện từ năm 2011 khi Google chứng minh rằng có thể tạo ra một máy dò tìm khuôn mặt chỉ bằng những hình ảnh không được gắn nhãn. Họ đã thiết kế một hệ thống có thể tự học để phát hiện hình ảnh con mèo mà không cần giải thích với hệ thống là con mèo trông như thế nào.

Vào thời điểm đó, mạng lưới thần kinh là 1.000 máy tính được tạo thành từ 16.000 lõi. Nó được nuôi dưỡng với 10 triệu video YouTube ngẫu nhiên, Tiến sĩ J. Dean, người làm việc trong dự án này, đã giải thích trong một cuộc phỏng vấn với New York Times rằng họ không bao giờ nói với hệ thống trong quá trình đào tạo rằng “đây là một con mèo”, vì vậy nó, về cơ bản, tự phát minh ra khái niệm về một con mèo.

Ngày nay, điện thoại thông minh có thể sử dụng máy ảnh chất lượng cao để nhận dạng. Ví dụ: iPhone X của Apple chạy công nghệ Face ID để người dùng có thể mở khóa điện thoại của họ. Dữ liệu khuôn mặt này được mã hóa và lưu trữ trên đám mây và nó cũng có thể được sử dụng cho mục đích khác như xác thực khi thanh toán.

Chương 2: Lọc ảnh

2.1. Lọc ảnh trong miền không gian

2.1.1. Hàm tích tích chập hai ảnh

Convolution là kỹ thuật quan trọng trong Xử Lý Ảnh, được sử dụng chủ yếu trong các phép toán trên ảnh như: đạo hàm ảnh, làm trơn ảnh, trích xuất biên cạnh trong ảnh.

Theo toán học, tích chập là phép toán tuyến tính, cho ra kết quả là một hàm bằng việc tính toán dựa trên hai hàm đã có (f và g).

Ví dụ: Đối với phép lọc ảnh, phép tích chập giữa ma trận lọc và ảnh, cho ra kết quả ảnh đã được xóa nhiễu (làm mờ).

Công thức tích chập giữa hàm ảnh $f(x, y)$ và bộ lọc $k(x, y)$ (kích thước $m \times n$):



$$k(x, y) \star f(x, y) = \sum_{u=-m/2}^{m/2} \sum_{v=-n/2}^{n/2} k(u, v) f(x - u, y - v)$$

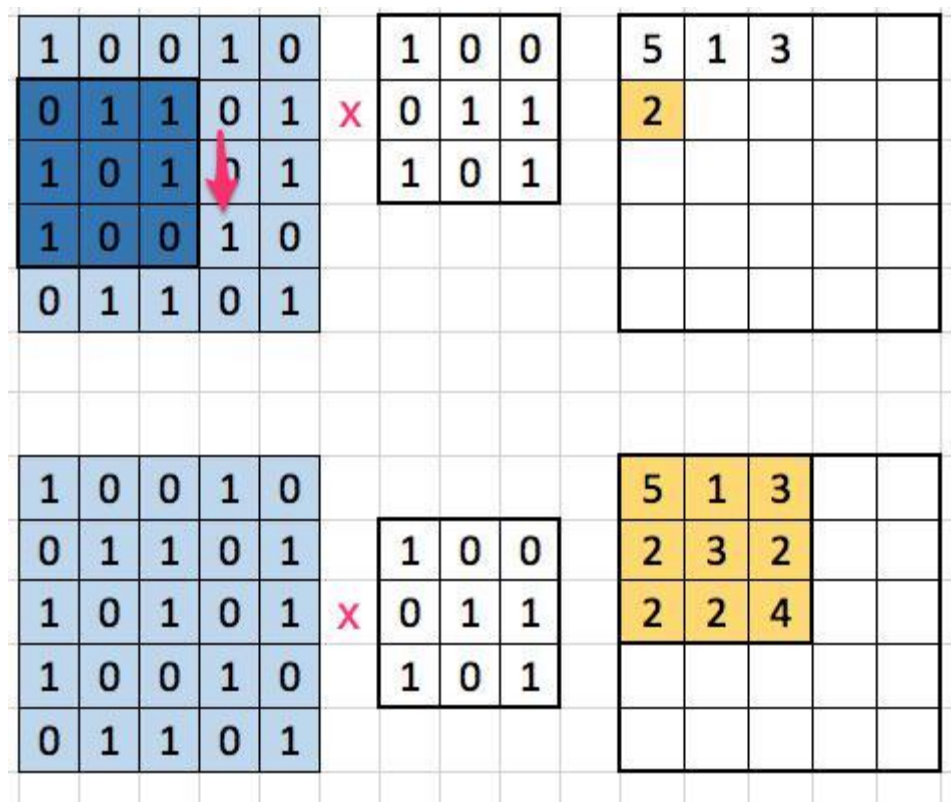
Thành phần không thể thiếu của phép tích chập là ma trận kernel (bộ lọc). Điểm neo (anchor point) của kernel sẽ quyết định vùng ma trận tương ứng trên ảnh để tích chập, thông thường anchor point được chọn là **tâm** của kernel. Giá trị mỗi phần tử trên kernel được xem như là hệ số tổ hợp với lần lượt từng giá trị độ xám của điểm ảnh trong vùng tương ứng với kernel.

Phép tích chập được hình dung thực hiện bằng việc dịch chuyển ma trận kernel lần lượt qua tất cả các điểm ảnh trong ảnh, bắt đầu từ góc bên trái trên của ảnh. Và đặt anchor point tương ứng tại điểm ảnh đang xét. Ở mỗi lần dịch chuyển, thực hiện tính toán kết quả mới cho điểm ảnh đang xét bằng công thức tích chập.

Ví dụ minh họa thử nghiệm với ma trận ảnh đầu vào đủ nhỏ để tính toán thủ công: Bên trái là ma trận đầu vào, có thể là ảnh. Bên phải là kernel là một ma trận dùng để biến đổi ma trận đầu vào trong 2 vòng lặp lồng nhau convolution. Từ trái qua phải, từ trên xuống dưới dịch chuyển cửa sổ có kích thước bằng với kernel trên ma trận đầu vào, thực hiện phép nhân từng phần tử cùng vị trí ở ma trận cửa sổ với kernel rồi tính tổng ra giá trị scalar điền vào ma trận kết quả. Đây gọi là dot product

$$a \cdot b = \sum_{i=1}^n a_i b_i = a_1 b_1 + b_2 b_2 + \dots + b_n b_n$$

image		kernel		result																																																											
<table><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	1	0	0	1	0	0	1	1	0	1	1	0	1	0	1	1	0	0	1	0	0	1	1	0	1	x	<table><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr></table>	1	0	0	0	1	1	1	0	1		<table><tr><td>5</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>	5																								
1	0	0	1	0																																																											
0	1	1	0	1																																																											
1	0	1	0	1																																																											
1	0	0	1	0																																																											
0	1	1	0	1																																																											
1	0	0																																																													
0	1	1																																																													
1	0	1																																																													
5																																																															
																																																															
<table><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	1	0	0	1	0	0	1	1	0	1	1	0	1	0	1	1	0	0	1	0	0	1	1	0	1	x	<table><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr></table>	1	0	0	0	1	1	1	0	1		<table><tr><td>5</td><td>1</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>	5	1																							
1	0	0	1	0																																																											
0	1	1	0	1																																																											
1	0	1	0	1																																																											
1	0	0	1	0																																																											
0	1	1	0	1																																																											
1	0	0																																																													
0	1	1																																																													
1	0	1																																																													
5	1																																																														
																																																															
<table><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	1	0	0	1	0	0	1	1	0	1	1	0	1	0	1	1	0	0	1	0	0	1	1	0	1	x	<table><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr></table>	1	0	0	0	1	1	1	0	1		<table><tr><td>5</td><td>1</td><td>3</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>	5	1	3																						
1	0	0	1	0																																																											
0	1	1	0	1																																																											
1	0	1	0	1																																																											
1	0	0	1	0																																																											
0	1	1	0	1																																																											
1	0	0																																																													
0	1	1																																																													
1	0	1																																																													
5	1	3																																																													



- **Lưu ý:**
 - Tích chập biến đổi các điểm ảnh nằm kề nhau trong ma trận cửa sổ
 - Ma trận kernel có hàng và cột là số lẻ để giá trị dot product gán được vào ô chính giữa, ma trận đầu đôi một hàng, một cột
 - Không nhất thiết phải bằng nhau, như thông thường kernel là ma trận vuông
 - Kết quả sau tích chập convolution sẽ bị hụt đi vài hàng và cột. Ví dụ kernel matrix là (3x3), thì hụt 2 hàng, 2 cột, kernel matrix (4, 4) sẽ hụt 3 hàng, 3 cột.
- **Một số cách xử lý vùng kernel vượt ra ngoài khỏi ảnh:**
 - Bỏ qua, không thực hiện tính phần tử đó vào kết quả.
 - Sử dụng một hằng số để tính toán.
 - Duplicate pixel nằm ở biên của ảnh.

2.1.2. Lọc trung bình (Median Filter)

- **Ý tưởng**

Trong lọc trung bình, mỗi điểm ảnh (Pixel) được thay thế bằng trung bình trọng số của các điểm trong vùng lân cận.

Giả sử rằng có 1 ma trận lọc (Kernel) (3x3) quét qua từng điểm ảnh của ảnh đầu vào I_{src} . Tại vị trí mỗi điểm ảnh lấy giá trị của các điểm ảnh tương ứng trong vùng (3x3) của ảnh gốc đặt vào ma trận lọc (Kernel). Giá trị điểm ảnh của ảnh đầu ra I_{dst} là giá trị trung bình của tất cả các điểm trong ảnh trong ma trận lọc (Kernel).

- **Thuật toán**

1 ảnh đầu vào với $I(x,y)$ là giá trị điểm ảnh tại 1 điểm (x,y) và 1 ngưỡng θ .

- Bước 1: Tính tổng các thành phần trong ma trận lọc (Kernel).
- Bước 2: Chia lấy trung bình của tổng các thành phần trong ma trận được tính ở trên với số lượng các phần tử của cửa sổ lọc ra 1 giá trị $I_{tb}(x, y)$.
- Bước 3: Hiệu chỉnh:
 - Nếu $I(x,y) - I_{tb}(x,y) > \theta$ thì $I(x,y) = I_{tb}(x,y)$.
 - Nếu $I(x,y) - I_{tb}(x,y) \leq \theta$ thì $I(x,y) = I(x,y)$.

Chú ý: θ là 1 giá trị cho trước và có thể có hoặc không tùy thuộc vào mục đích.

- **Tác dụng**

Trong lọc trung bình, thường ưu tiên cho các hướng để bảo vệ biên của ảnh khỏi bị mờ khi làm trơn ảnh. Các kiểu ma trận lọc (Kernel) được sử dụng tùy theo các trường hợp khác nhau. Các bộ lọc trên là bộ lọc tuyến tính theo nghĩa là điểm ảnh ở tâm cửa sổ sẽ được thay bởi tổ hợp các điểm lân cận chập với ma trận lọc (Kernel).

Ví dụ

Ma trận lọc (Kernel) M ở nội dung phần Ví dụ: Lọc số ảnh là gì? như ở trên và phía dưới lần lượt là ma trận điểm ảnh đầu vào (I_{src}) và ma trận điểm ảnh đầu ra (I_{dst}).

$$I_{src} = \begin{bmatrix} 2 & 4 & 3 & 7 & 2 \\ 5 & 7 & 2 & 1 & 4 \\ 7 & 6 & 2 & 8 & 2 \\ 5 & 6 & 7 & 7 & 2 \\ 8 & 2 & 1 & 6 & 2 \end{bmatrix} \quad I_{dst} = \frac{1}{9} \begin{bmatrix} 18 & 23 & 24 & 19 & 14 \\ 31 & 38 & 40 & 29 & 24 \\ 36 & 47 & 46 & 35 & 24 \\ 34 & 44 & 45 & 37 & 27 \\ 21 & 29 & 19 & 25 & 17 \end{bmatrix}$$

$$I_{dst}(3,3) = (7+2+1+6+2+8+6+7+7) / 9 = 46 / 9 = 5.$$

Chú ý

Các hệ số trong ma trận lọc (Kernel) này có tổng bằng 1, nên độ sáng ảnh giữ nguyên, và các hệ số đều dương nên nó có khuynh hướng làm nhoè ảnh.

- **Chương trình cài đặt**

```
#include<stdio.h>
#include"opencv2\core\core.hpp"
#include"opencv2\highgui\highgui.hpp"
#include"opencv2\imgproc\imgproc.hpp"
#include<cmath>
using namespace cv;
using namespace std;
int h2(Mat a, int x, int y)
{
    return a.at<uchar>(x - 1, y - 1) + a.at<uchar>(x - 1, y) + a.at<uchar>(x - 1, y + 1)
        + a.at<uchar>(x, y - 1) + a.at<uchar>(x, y) + a.at<uchar>(x, y + 1)
        + a.at<uchar>(x + 1, y - 1) + a.at<uchar>(x + 1, y) + a.at<uchar>(x + 1, y +
1);
}
int main()
{
    Mat A = imread("D://anhnhieu5.jpg", CV_LOAD_IMAGE_GRAYSCALE);
    Mat B = A.clone();
    if (!A.data)
    {
        printf("Khong tim thay anh!");
        return -1;
    }
    for (int i = 0; i < A.rows; i++)
    {
        for (int j = 0; j < A.cols; j++)
        {
            B.at<uchar>(i, j) = 0;
        }
    }
    int GtTb = 0;
    int TheTa = 3;
    for (int i = 1; i < A.rows - 1; i++)
    {
        for (int j = 1; j < A.cols - 1; j++)
```



```

    {
        GtTb = h2(A, i, j);
        GtTb = round(GtTb / 9.0);
        if ((A.at<uchar>(i, j) - GtTb) < TheTa) {
            B.at<uchar>(i, j) = A.at<uchar>(i, j);
        }
        else
        {
            B.at<uchar>(i, j) = GtTb;
        }
    }
}
imshow("Anh goc: ", A);
imshow("Anh ket qua: ", B);
waitKey(0);
return 0;
}

```

- **Kết quả của chương trình**



2.1.3. Lọc trung vị

- **Ý tưởng**

Lọc trung vị là lọc phi tuyến. 1 phép lọc phi tuyến là 1 kết quả không thể thu được từ 1 tổng trọng số của các điểm ảnh (Pixel) lân cận. Sau khi đã định nghĩa kích thước vùng lân cận, giá trị điểm ảnh (Pixel) trung tâm được thay bằng trung vị tức là giá trị chính giữa của tất cả các giá trị của các điểm trong vùng lân cận.

Cho 1 dãy số $X_1, X_2, X_3, \dots, X_n$ được sắp xếp theo thứ tự tăng dần hoặc giảm dần. Khi đó X_{tv} được tính bởi công thức:

Nếu n lẻ:

$$X_{tv} = X\left(\frac{n}{2} + 1\right)$$

Nếu n chẵn:

$$X_{tv} = \frac{X\left(\frac{n}{2}\right) + X\left(\frac{n}{2} + 1\right)}{2}$$

Kích thước của ma trận lọc (Kernel) thường dùng có kích thước 3x3, 5x5, 7x7.

- **Thuật toán**

- Bước 1: Xác định điểm ảnh (Pixel) $I(x, y)$.
- Bước 2: Sắp xếp các điểm ảnh (Pixel) lân cận của theo 1 thứ tự nhất định (Tăng dần hoặc giảm dần).
- Bước 3: Xác định I_{tv} (Chính là X_{tv} trong nội dung phía trên).
- Bước 4: Hiệu chỉnh:
 - Nếu $I(x,y) - I_{tv}(x,y) > \theta$ thì $I(x,y) = I_{tv}(x,y)$.
 - Nếu $I(x,y) - I_{tv}(x,y) \leq \theta$ thì $I(x,y) = I(x,y)$.

Chú ý

θ là 1 giá trị cho trước và có thể có hoặc không tùy thuộc vào mục đích.

- **Tác dụng**

- Loại bỏ các điểm ảnh mà vẫn đảm bảo độ phân giải.
- Hiệu quả trong việc giảm đi điểm nhiễu trong ma trận lọc (Kernel) lớn hay bằng 1 nửa số điểm trong ma trận lọc (Kernel).

- Ví dụ

$$I_{src} = \begin{bmatrix} 2 & 4 & 3 & 7 & 2 \\ 5 & 7 & 2 & 1 & 4 \\ 7 & 6 & 2 & 8 & 2 \\ 5 & 6 & 7 & 7 & 2 \\ 8 & 2 & 1 & 6 & 2 \end{bmatrix}$$

$Idst(1,1) = (2, 2, 2, 2, 2, 2, 4, 7, 5) = 2.$

$Idst(3,3) = (1, 2, 2, 6, 6, 7, 7, 7, 8) = 6.$

Chú ý

$Idst(1,1)$, $Idst(1,5)$, $Idst(5,1)$, $Idst(5,5)$: có thể làm như cách trên hoặc không cần kiểm tra điểm biên.

- Chương trình cài đặt

```
#include <stdio.h>
#include "opencv2\core\core.hpp"
#include "opencv2\highgui\highgui.hpp"
#include "opencv2\imgproc\imgproc.hpp"
#include <cmath>
using namespace cv;
using namespace std;

int trungvi(int a1[], int n) { //ham sap xep gia tri giam dan cua anh lenh kien
    int tg;
    for (int i = 0; i < n - 1; i++) {
        for (int j = n - 1; j > i; j--) {
            if (a1[i] < a1[j]) {
                tg = a1[i]; a1[i] = a1[j]; a1[j] = tg;
            }
        }
    }
    if (n % 2 == 0) return (a1[n / 2] + a1[n / 2 - 1]) / 2;
    else return a1[(n + 1) / 2];
}

int main() {
```

```

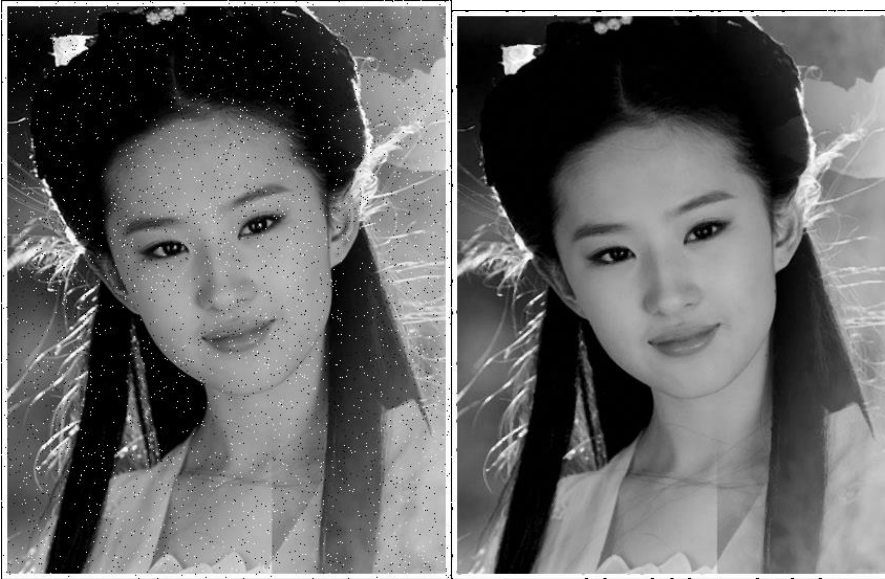
    Mat a = imread("d://ANHNNHIEU1.jpg", CV_LOAD_IMAGE_GRAYSCALE); //
neu n chia 2 bang 0 gia tri
    Mat b = a.clone();
    int nguong = 10; // tao ra anh moi la anh ao cua anh a truoc khi xu ly
    if (!a.data)
    {
        printf("khong the mo hay tim thay anh\n");
        return -1;
    }
    int a2[9]; // ma tran anh
    for (int i = 1; i < a.rows - 1; i++) {
        for (int j = 1; j < a.cols - 1; j++) {
            a2[0] = a.at<uchar>(i - 1, j - 1);
a2[1] = a.at<uchar>(i, j - 1);
a2[2] = a.at<uchar>(i + 1, j - 1);
            a2[3] = a.at<uchar>(i - 1, j);
a2[4] = a.at<uchar>(i, j);
a2[5] = a.at<uchar>(i + 1, j);
            a2[6] = a.at<uchar>(i - 1, j + 1);
a2[7] = a.at<uchar>(i, j + 1);
a2[8] = a.at<uchar>(i + 1, j + 1);
            if (abs(a.at<uchar>(i, j) - trungvi(a2, 9)) > nguong)
b.at<uchar>(i, j) = trungvi(a2, 9);

        }
    }

    imshow("anh_goc", a);
    imshow("anh_sau", b);
    waitKey(0);
    return 0;
}

```

- **Kết quả của chương trình**



2.1.4. Bộ lọc Sobel

Kỹ thuật sử dụng 2 ma trận lọc (Kernel) xấp xỉ đạo hàm theo hướng x và y:

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- **Sử dụng bộ lọc sobel trong OpenCV**

Là 1 phép lọc giúp tìm đường biên cho ảnh. Trong OpenCV để sử dụng Sobel cho 1 hình ảnh, sử dụng hàm sau:

```
cv::Sobel(cv::InputArray src, cv::OutputArray dst, int ddepth, int dx,
          int dy, int ksize = 3, double scale = (1,0), double delta = (0,0),
          int borderType = 4);
cv::Sobel(src, dst, src.depth(), 1, 0, 3);
```

- src: ảnh gốc.
- dst: ảnh sau khi thực hiện phép lọc số ảnh.
- ksize: kích thước của ma trận lọc. Giá trị mặc định là 3.
- ddepth: độ sâu của ảnh sau phép lọc: VD: CV_32F, CV_64F,...
- dx: đạo hàm theo hướng x. dx = 1 nếu đạo hàm theo hướng x.
- dy: đạo hàm theo hướng y. dy = 1 nếu đạo hàm theo hướng y.

- scale và delta: 2 thông số tùy chọn cho việc tính giá trị đạo hàm lưu giá trị ví sai vào ảnh sau phép lọc. Mặc định là 1 và 0.
- borderType: phương pháp để ước lượng và căn chỉnh các điểm ảnh nếu phép lọc chúng vượt ra khỏi giới hạn của ảnh. Giá trị mặc định là 4.

Ví dụ:

```
#include <stdio.h>
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace std;
using namespace cv;

int main()
{
    // Đọc ảnh mẫu
    Mat src = imread("d://lena1.JPG"), (CV_LOAD_IMAGE_COLOR);
    Mat dst = src.clone(); // sao chép ma trận B = A
    cv::Mat kernel = cv::Mat::ones(Size(3, 3), CV_32F) / (float)(9);

    cv::Sobel(src, dst, src.depth(), 1, 0, 3);
    imshow("Image Srd", src);
    imshow("Image Dst", dst);

    waitKey(0);
    return(0);
}
```



2.1.5. Bộ lọc Laplace

Là 1 phép lọc giúp tìm đường biên cục bộ cho ảnh. Tư tưởng là lấy đạo hàm bậc hai của các điểm. Ma trận lọc (Kernel) của bộ lọc Laplace có dạng:

$$M = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Trong thực tế dùng nhiều kiểu ma trận lọc (Kernel) khác nhau để xấp xỉ rời rạc đạo hàm bậc hai Laplace. 3 kiểu ma trận lọc (Kernel) thường dùng:

$$M_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad M_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad M_3 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

- **Sử dụng bộ lọc sobel trong OpenCV**

Trong OpenCV để sử dụng Laplace cho 1 hình ảnh, sử dụng hàm sau:

```
cv::Laplace(cv::InputArray src, cv::OutputArray dst, int ddepth, int ksize = 1,
            double scale = (1,0), double delta = (0,0), int borderType = 4);
```

Chú ý

- Có 1 phương pháp lọc dùng để tìm đường biên cục bộ khác là Gradient, phương pháp này làm việc khá tốt khi độ sáng thay đổi rõ nét, khi mức xám thay đổi chậm hoặc miền chuyển tiếp trải rộng thì phương pháp này tỏ ra kém hiệu quả. Vậy nên sử dụng phương pháp laplace để khắc phục nhược điểm này.
- Trong kỹ thuật lọc laplace, điểm biên được xác định bởi điểm cắt điểm không. Và điểm không là duy nhất do vậy kỹ thuật này cho đường biên rất mảnh (Rộng 1 pixel). Rất nhạy cảm với nhiễu do đạo hàm bậc 2 không ổn định.

- **Ví dụ:**

```
#include <stdio.h>
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace std;
using namespace cv;

int main()
{
    // Đọc ảnh màu
    Mat src = imread("d://lena1.JPG"), (CV_LOAD_IMAGE_COLOR);
    Mat dst = src.clone(); // sao chép ma trận B = A
    cv::Mat kernel = cv::Mat::ones(Size(3, 3), CV_32F) / (float)(9);

    cv::Laplacian(src, dst, src.depth(), 3);
    //cv::Laplacian()

    imshow("Image Srd", src);
    imshow("Image Dst", dst);

    waitKey(0);
    return(0);
}
```



2.1.6. Cài đặt bộ lọc trong OpenCV

Trong nội dung ở phía trên bài viết đã giới thiệu 1 số bộ lọc có sẵn trong OpenCV. Trong OpenCV có 1 hàm tính chập, nhờ đó có thể tự cài các bộ lọc cho riêng để phù hợp với mục đích xử lý ảnh. Hàm tính chập trong OpenCV như sau:

```
cv::Filter2D(cv::InputArray src, cv::OutputArray dst, int ddepth  
             cv::InputArray kernel, cv::Point anchor=cv::Point(-1, -1)  
             double delta = (0,0), int borderType = 4);
```

- src: ảnh ban đầu.
- dst: ảnh sau khi nhân chập
- kernel: mặt nạ nhân chập
- anchor: điểm neo để đặt mặt nạ nhân chập.

Điều quan trọng nhất với các tự cài đặt các bộ lọc đó chính là ma trận lọc (Kernel). Có thể dễ dàng tìm được các định ma trận lọc (Kernel). Trong các nội dung ở phần trên, các bộ lọc được cài đặt sẵn trong OpenCV có giới thiệu các ma trận lọc đi kèm cho từng phép lọc.

```
#include <stdio.h>  
#include "opencv2/core/core.hpp"  
#include "opencv2/highgui/highgui.hpp"  
#include "opencv2/imgproc/imgproc.hpp"  
using namespace std;  
using namespace cv;  
  
int main()  
{  
    // Đọc ảnh màu  
    Mat src = imread("d://lena1.JPG"), (CV_LOAD_IMAGE_COLOR);  
    Mat dst = src.clone(); // sao chép ma trận B = A  
    cv::Mat kernel = cv::Mat::ones(Size(3, 3), CV_32F) / (float)(9);  
  
    cv::filter2D(src, dst, src.depth(), kernel,  
                Point(-1, -1), 0, BORDER_DEFAULT);  
  
    imshow("Image Srd", src);  
    imshow("Image Dst", dst);
```

```

    waitKey(0);
    return(0);
}

```

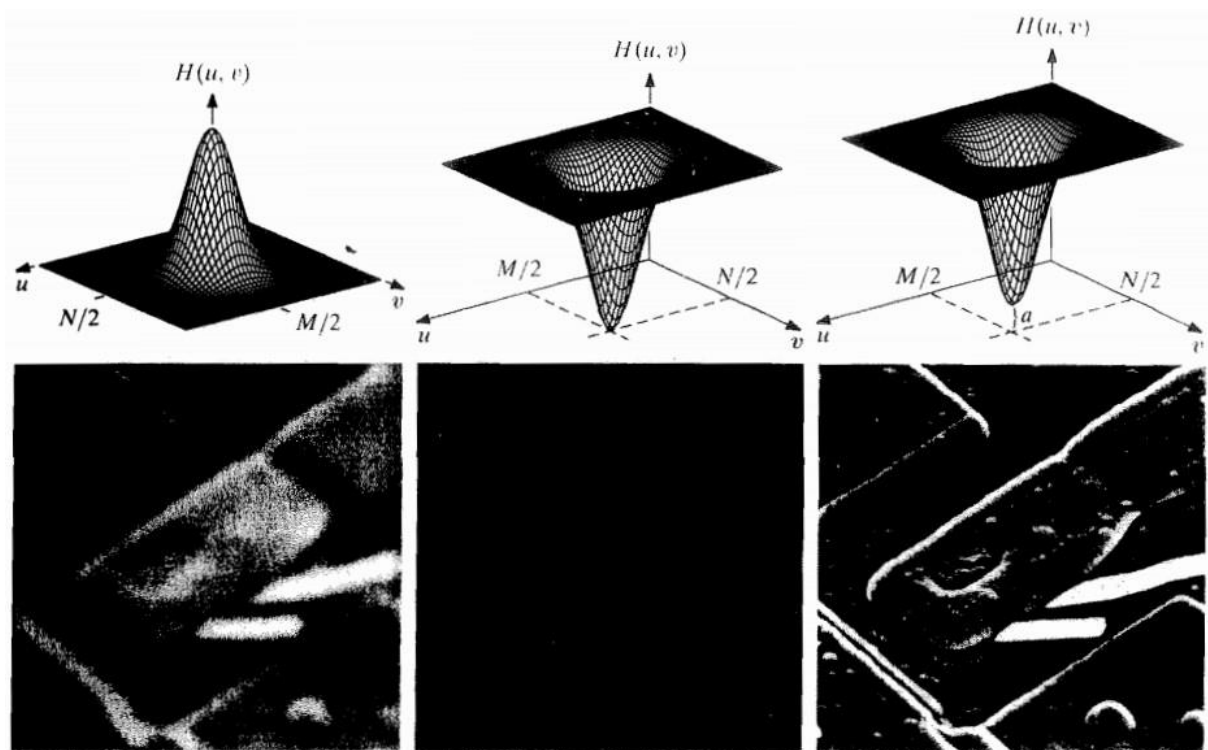
- Kết quả của chương trình



2.2. Lọc ảnh trong miền tần số

2.2.1. Khái niệm

Khi chụp ảnh, ta có thể thu được các ảnh có tần số thấp (sự thay đổi mức xám của ảnh ít, ví dụ như ảnh một bức tường) hay ảnh có tần số cao (ví dụ như biên của vật thể). Vì vậy, ta cần có bộ lọc $H(u, v)$ có thể làm giảm đi tần số cao trong khi đi qua các tần số thấp (gọi là lọc thông thấp) làm cho ảnh mờ đi. Ngược lại, ta cần có bộ lọc tổ tính chất ngược với lọc thông thấp (gọi là lọc thông cao) giúp tăng cường chi tiết hình dạng vật thể, đồng thời làm giảm độ tương phản ảnh. Hình ảnh sau mô tả điều này



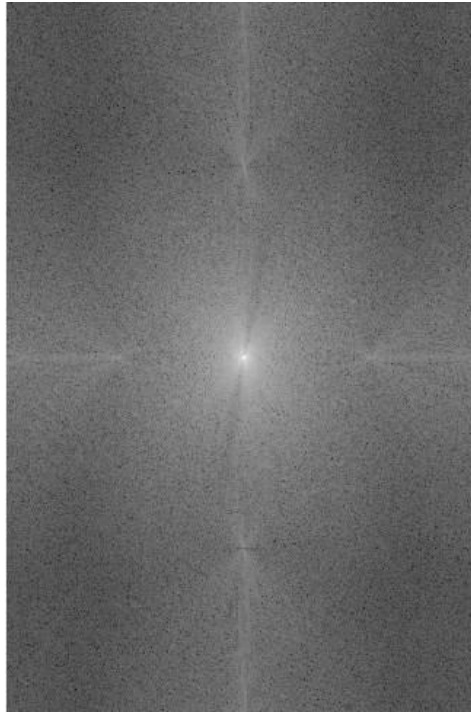
Hình 1.5 – 1. Ảnh trên là hàm đồ thị tần số ứng với ảnh bên dưới

Ta hãy xem một ảnh trong miền tần số trông như thế nào, ta quan sát ảnh sau:



Hình 1.5 – 2. Ảnh Mặt Trăng

Sử dụng hàm FFT, ta sẽ biểu diễn ảnh trên trong miền tần số như sau:



Hình 1.5 – 3. Ảnh Mặt Trăng khi chuyển qua miền tần số

Định lý tích chập cho ta mối quan hệ giữa miền không gian và miền tần số, cụ thể, thông qua tích chập, một ảnh trong miền không gian có thể chuyển qua miền tần số và ngược lại.

Quy trình lọc ảnh trong miền tần số như sau:

Ảnh -> Chuyển đổi Fourier > Lọc -> Chuyển đổi Fourier ngược -> Ảnh

- Bước 1: Xử lý ảnh trong miền không gian, tức tăng hoặc giảm độ sáng của ảnh.
- Bước 2: Lấy DFT của ảnh.
- Bước 3: Canh giữa DFT, tức mang DFT từ góc ảnh ra giữa ảnh
- Bước 4: Thực hiện tích chập với hàm lọc.
- Bước 5: Trượt DFT từ giữa ảnh ra góc.
- Bước 6: Lấy chuyển đổi ngược IDFT, tức chuyển ảnh từ miền tần số sang miền không gian.

Sau khi chuyển ảnh sang miền không gian, ta áp dụng một số bộ lọc trong quy trình lọc ảnh nhằm làm mờ ảnh, giảm nhiễu, làm nét ảnh.

- Các bộ lọc ảnh thông dụng:
 - Lọc thông thấp Ideal

- Lọc thông thấp Gauss
- Lọc thông thấp Butterworth
- Lọc thông cao Ideal
- Lọc thông cao Gauss
- Lọc thông cao Butterworth

2.2.2. Lọc thông thấp - Low Pass Filter

LPF - Low Pass Filter là 1 bộ lọc tuyến tính, thường được sử dụng để làm trơn nhiễu. Kỹ thuật lọc nhiễu ảnh này cần sử dụng 1 số ma trận lọc (Kernel) như sau:

$$M = \frac{1}{8} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Hoặc

$$M = \frac{1}{(b+2)^2} \begin{bmatrix} 1 & b & 1 \\ b & b^2 & b \\ 1 & b & 1 \end{bmatrix}$$

2.2.3. Lọc thông cao - High Pass Filter

HPF - High Pass Filter là 1 bộ lọc phi tuyến tính. Thường dùng trong việc làm trơn ảnh và tìm biên đối tượng có ở trong ảnh. 1 số ma trận lọc (Kernel) dùng trong lọc thông cao:

$$M_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad M_2 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad M_3 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Chú ý

Trong lọc thông cao, các ma trận lọc (Kernel) có tổng hệ số các giá trị của bộ lọc bằng 1.

2.2.4. Bộ lọc Blur

Là 1 phép lọc làm cho trơn ảnh và khử nhiễu hạt và là 1 bộ lọc trung bình. Ma trận lọc (Kernel) của bộ lọc Blur có dạng:

$$M = \frac{1}{\text{rows} * \text{cols}} \begin{bmatrix} 1 & 1 & 1 & 1 \dots 1 \\ 1 & 1 & 1 & 1 \dots 1 \\ 1 & 1 & 1 & 1 \dots 1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & 1 \dots 1 \end{bmatrix}$$

Trong OpenCV để sử dụng Blur cho 1 hình ảnh, sử dụng hàm sau:

```
cv::blur(cv::InputArray src, cv::InputArray dst, cv::Size ksize,
        cv::Point anchor = cv::Point(-1,-1), int borderType = 4)
```

- src: Là ảnh gốc.
- dst: Là ảnh sau khi thực hiện phép lọc số ảnh.
- ksize: Là kích thước của ma trận lọc.
- anchor: Là Anchor Point của ma trận lọc. Giá trị mặc định là (-1,-1).
- borderType: Là phương pháp để ước lượng và căn chỉnh các điểm ảnh nếu phép lọc chúng vượt ra khỏi giới hạn của ảnh. Giá trị mặc định là 4.

• Ví dụ

```
#include <stdio.h>
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace std;
using namespace cv;

int main()
{
    // Đọc ảnh mẫu
    Mat src = imread("d://lena1.JPG"), (CV_LOAD_IMAGE_COLOR);
    Mat dst = src.clone(); // sao chép ma trận B = A
    cv::Mat kernel = cv::Mat::ones(Size(3, 3), CV_32F) / (float)(9);

    cv::blur(src, dst, Size(3, 3), Point(-1, -1), 4);

    imshow("Image Srd", src);
    imshow("Image Dst", dst);
```

```
waitKey(0);  
return(0);  
}
```



Chương 3: Phát hiện đường biên

3.1. Các phương pháp phát hiện đường biên

Phát hiện biên là một công cụ quan trọng trong xử lý ảnh số. Nó làm giảm một cách đáng kể khối lượng dữ liệu cần tính toán, chỉ giữ lại một số ít những thông tin cần thiết đồng thời vẫn bảo toàn được những cấu trúc quan trọng trong bức ảnh.

Về mặt toán học người ta coi điểm biên của ảnh là điểm có sự biến đổi đột ngột về độ xám. Tập hợp các điểm biên tạo thành biên hay đường bao của ảnh.

Định nghĩa toán học của biên ở trên là cơ sở cho các kỹ thuật phát hiện biên. Điều quan trọng là sự biến thiên giữa các điểm ảnh là nhỏ, trong khi đó biến thiên độ sáng của điểm biên (khi qua biên) lại khá lớn. Xuất phát từ cơ sở này người ta thường sử dụng 2 phương pháp phát hiện biên sau:

- Phương pháp phát hiện biên trực tiếp: phương pháp này nhằm làm nổi đường biên dựa vào biến thiên về giá trị độ sáng của điểm ảnh. Kỹ thuật chủ yếu là dùng kỹ thuật đạo hàm. Nếu lấy đạo hàm bậc nhất của ảnh ta có phương pháp Gradient, nếu lấy đạo hàm bậc 2 ta có kỹ thuật Laplace.
- Phương pháp gián tiếp: Nếu bằng cách nào đấy ta phân ảnh thành các vùng thì đường phân ranh giữa các vùng đó chính là biên.

3.1.1. Phương pháp Gradient

Đạo hàm bậc nhất của ảnh theo hướng ngang và dọc được tính như sau:

$$\frac{\partial f(x,y)}{\partial x} = f_x \approx \frac{f(x+dx,y)-f(x,y)}{dx} \quad (4)$$

$$\frac{\partial f(x,y)}{\partial y} = f_y \approx \frac{f(x,y+dy)-f(x,y)}{dy} \quad (5)$$

Biên độ hay độ lớn của vector Gradient được tính theo công thức :

$$|\nabla f(x,y)| = \sqrt{(h_x \otimes f(x,y))^2 + (h_y \otimes f(x,y))^2} \quad (6)$$

Hướng của vector gradient được xác định theo công thức:

$$\psi(\nabla f(x,y)) = \arctan \left\{ \frac{h_y \otimes f(x,y)}{h_x \otimes f(x,y)} \right\} \quad (7)$$

Hướng của biên sẽ vuông góc với hướng của vector gradient.

• Toán tử Sobel

Toán tử Sobel sử dụng hai mặt nạ có kích thước $[3 \times 3]$ trong đó một mặt nạ chỉ đơn giản là sự quay của mặt nạ kia đi một góc 90° . Các mặt nạ này được thiết kế để tìm ra các

đường biên theo chiều đứng và chiều ngang một cách tốt nhất. Ảnh được lọc bằng cả hai toán tử, sau đó cộng hai kết quả với nhau.

Giá trị của toán tử Sobel như sau :

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (8)$$

- **Toán tử Prewitt**

Phương pháp Prewitt được thực hiện giống với Sobel. Phương pháp này cũng được được thiết kế để tìm ra các đường biên theo chiều đứng và chiều ngang một cách tốt nhất. Giá trị của toán tử Prewitt như sau:

$$H_x = \begin{bmatrix} -1 & -2 & -1 \\ -1 & 0 & 0 \\ -1 & 2 & 1 \end{bmatrix}, \quad H_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad (9)$$

- **Toán tử Robert**

Phương pháp Robert được được thiết kế để tìm ra các đường biên dạng đường chéo một cách tốt nhất. Giá trị của toán tử Prewitt như sau:

$$H_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad H_y = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad (10)$$

- **La bàn**

Toán tử la bàn đo gradient theo 8 hướng. Toán tử này thích hợp với các ảnh có đường biên đồng đều theo cả 8 hướng. Có nhiều toán tử la bàn khác nhau, dưới đây là các mặt nạ theo 8 hướng của toán tử Kish:

$$\begin{aligned} H_1 &= \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} & H_2 &= \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix} & H_3 &= \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \\ H_4 &= \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix} & H_5 &= \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} & H_6 &= \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix} \\ H_7 &= \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} & H_8 &= \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \end{aligned} \quad (11)$$

Trong đó $H_1, H_2, H_3, \dots, H_8$ tương ứng với 8 hướng: $0^\circ, 45^\circ, 90^\circ, 135^\circ, 180^\circ, 225^\circ, 315^\circ$.

Nếu ta kí hiệu $\nabla_i, i=1, 2, \dots, 8$ là gradient thu được theo 8 hướng bởi 8 mặt nạ, biên độ gradient tại (x, y) được tính như sau:

$$|\nabla f(x, y)| = \text{Max}(|\nabla_i(x, y)|, i = 1, 2, 3, 4, 5, 6, 7, 8) \quad (12)$$

3.1.2. Phương pháp Laplacian

Các phương pháp gradient ở trên làm việc khá tốt khi độ sáng thay đổi rõ nét. Khi mức xám thay đổi chậm, miền chuyển tiếp trải rộng, phương pháp hiệu quả hơn đó là phương pháp sử dụng đạo hàm bậc 2, gọi là phương pháp Laplace. Toán tử Laplace được định nghĩa như sau:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (12)$$

Trong không gian rời rạc đạo hàm bậc 2 có thể tính:

$$\frac{\partial^2 f}{\partial x^2} = 2f(x, y) - f(x-1, y) - f(x+1, y) \quad (13)$$

$$\frac{\partial^2 f}{\partial y^2} = 2f(x, y) - f(x, y-1) - f(x, y+1) \quad (14)$$

$$\nabla^2 f = -f(x-1, y) - f(x, y-1) + 4f(x, y) - f(x, y+1) - f(x+1, y) \quad (15)$$

Toán tử Laplace dùng nhiều kiểu mặt nạ khác nhau để xấp xỉ rời rạc đạo hàm bậc hai. Dưới đây là 3 kiểu mặt nạ thường dùng:

$$H_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad H_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad H_3 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix} \quad (16)$$

3.2. Một số chương trình phát hiện biên ảnh

3.2.1. Chương trình phát hiện biên sử dụng toán tử sobel

```
#include<iostream>
#include<cmath>
#include<opencv2/imgproc/imgproc.hpp>
#include<opencv2/highgui/highgui.hpp>
using namespace std;
using namespace cv;
int Hx(Mat image, int x, int y)
{
    return
        -image.at<uchar>(y - 1, x - 1)
        - 2 * image.at<uchar>(y, x - 1)
```

```

        - image.at<uchar>(y + 1, x - 1)
        + image.at<uchar>(y - 1, x + 1)
        + 2 * image.at<uchar>(y, x + 1)
        + image.at<uchar>(y + 1, x + 1);

    }

int Hy(Mat image, int x, int y)
{
    return
        -image.at<uchar>(y - 1, x - 1)
        - 2 * image.at<uchar>(y - 1, x)
        - image.at<uchar>(y - 1, x + 1)
        + image.at<uchar>(y + 1, x - 1)
        + 2 * image.at<uchar>(y + 1, x)
        + image.at<uchar>(y + 1, x + 1);
}

int main()
{
    Mat src, dst;
    int gx, gy, sum;
    src = imread("lena1.jpg", CV_LOAD_IMAGE_GRAYSCALE);
    dst = src.clone();
    for (int y = 0; y < src.rows; y++)
        for (int x = 0; x < src.cols; x++)
            dst.at<uchar>(y, x) = 0;
    for (int y = 1; y < src.rows - 1; y++)
    {
        for (int x = 1; x < src.cols - 1; x++)
        {
            gx = Hx(src, x, y);
            gy = Hy(src, x, y);
            sum = abs(gx) + abs(gy);
            if (sum > 255)
                sum = 255;
            if (sum < 0)

```

```

        sum = 0;;
        dst.at<uchar>(y, x) = sum;
    }
}
imshow("ANH CHUA XU LY", src);
imshow("ANH DA XU LY", dst);
waitKey();
}

```

- **Kết quả của chương trình như sau:**



3.2.2. Chương trình phát hiện sử dụng kỹ thuật laplace (Sử dụng mặt nạ H2)

$$H2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

```

#include <stdio.h>
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <cmath>
using namespace cv;
int h2(Mat C, int x, int y)
{
    return -C.at<uchar>(x - 1, y - 1) - C.at<uchar>(x - 1, y) - C.at<uchar>(x - 1,
y + 1)
        - C.at<uchar>(x, y - 1) + 8 * C.at<uchar>(x, y) - C.at<uchar>(x, y +
1)

```

```

        - C.at<uchar>(x + 1, y - 1) - C.at<uchar>(x + 1, y) - C.at<uchar>(x +
1, y + 1);
    }

int main()
{
    // Read image
    Mat A = imread("d://lena1.jpg", CV_LOAD_IMAGE_GRAYSCALE);
    Mat B = A.clone(); // sao chép 2 Mat
                        // Check for valid

    if (!A.data)
    {
        printf("khong tim thay anh.\n");
        return -1;
    }
#define KENH 3
    for (int i = 0; i < A.rows; i++) {
        for (int j = 0; j < A.cols; j++) {
            B.at< uchar >(i, j) = 0;
        }
    }

    int gtH;
    for (int i = 1; i < A.rows - 1; i++) {
        for (int j = 1; j < A.cols - 1; j++) {

            gtH = h2(A, i, j);
            if (gtH > 255) gtH = 255;
            if (gtH < 0) gtH = 0;
            B.at< uchar >(i, j) = gtH;
        }
    }
}

```

```

        imshow("Before", A);
        imshow("After", B);

        waitKey(0);

        return 0;
    }

```

Kết quả của chương trình như sau:



3.2.3. Phát hiện biên ảnh bằng thuật toán Candy

Phương pháp này sử dụng hai mức ngưỡng cao và thấp. Ban đầu ta dùng mức ngưỡng cao để tìm điểm bắt đầu của biên, sau đó chúng ta xác định hướng phát triển của biên dựa vào các điểm ảnh liên tiếp có giá trị lớn hơn mức ngưỡng thấp. Ta chỉ loại bỏ các điểm có giá trị nhỏ hơn mức ngưỡng thấp. Các đường biên yếu sẽ được chọn nếu chúng được liên kết với các đường biên khỏe.

Phương pháp Canny bao gồm các bước sau:

- Bước 1. Trước hết dùng bộ lọc Gaussian để làm mịn ảnh.
- Bước 2. Sau đó tính toán gradient của đường biên của ảnh đã được làm mịn.
- Bước 3. Tiếp theo là loại bỏ những điểm không phải là cực đại.
- Bước 4. Bước cuối cùng là loại bỏ những giá trị nhỏ hơn mức ngưỡng.

Phương pháp này hơn hẳn các phương pháp khác do ít bị tác động của nhiễu và cho khả năng phát hiện các biên yếu. Nhược điểm của phương pháp này là nếu chọn ngưỡng quá thấp sẽ tạo ra biên không đúng, ngược lại nếu chọn ngưỡng quá cao thì

nhiều thông tin quan trọng của biên sẽ bị loại bỏ. Căn cứ vào mức ngưỡng đã xác định trước, ta sẽ quyết định những điểm thuộc biên thực hoặc không thuộc biên. Nếu mức ngưỡng càng thấp, số đường biên được phát hiện càng nhiều (nhưng kèm theo là nhiễu và số các đường biên giả cũng xuất hiện càng nhiều). Ngược lại nếu ta đặt mức ngưỡng càng cao, ta có thể bị mất những đường biên mờ hoặc các đường biên sẽ bị đứt đoạn.

Phương pháp Canny có các ưu điểm sau:

- Cực đại hóa tỷ số tín hiệu trên nhiễu làm cho việc phát hiện các biên thực càng chính xác.
- Đạt được độ chính xác cao của đường biên thực.
- Làm giảm đến mức tối thiểu số các điểm nằm trên đường biên nhằm tạo ra các đường biên mỏng, rõ.

- **Chương trình phát hiện biên bằng phương pháp Candy:**

```
#include"opencv2\core\core.hpp"
#include"opencv2\imgproc\imgproc.hpp"
#include"opencv2\highgui\highgui.hpp"
#include<iostream>
#include<cmath>
using namespace std;
using namespace cv;
int main() {
    Mat a = imread("D://lena1.jpg",
CV_LOAD_IMAGE_GRAYSCALE); //docanhchuyenveanhxam
    Mat b = a.clone();
    Mat canny1;
    if (!a.data)
    {
        printf("khong the mo hay tim thay anh\n");
        return -1;
    }
    //phat hien bien candy
    GaussianBlur(b, b, Size(9, 9), 2);
    double t1 = 10, t2 = 70;
    Canny(b, canny1, t1, t2, 3, false);
```

```

    imshow("Anh goc", a);
    imshow("Bien anh candy ", canny1);
    waitKey(0);
    return 0;
}

```



3.3. Phát hiện đường thẳng với Hough transform

3.3. 1. Phương trình đường thẳng trong không gian ảnh

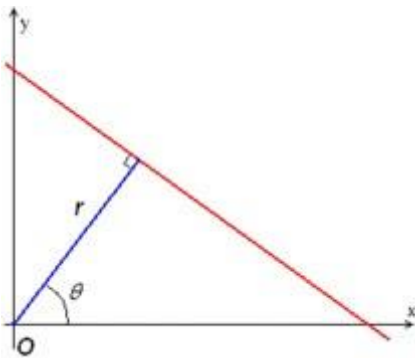
Trong hình học thì đường thẳng có phương trình:

$$y = m.x + c$$

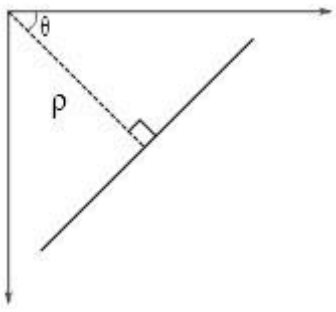
hoặc

$$\rho = x.\cos\theta + y.\sin\theta$$

Với ρ là khoảng cách từ đường thẳng tới gốc tọa độ, còn θ là góc giữa trục hoành và đoạn thẳng ngắn nhất nối tới gốc tọa độ (đơn vị là radian).



Hình minh họa trong hệ tọa độ Đề Cát



Hình minh họa sử dụng trong OpenCV

Lưu ý: Hệ tọa độ trên máy tính có gốc tọa độ ở góc trên bên trái

Có 2 cách để xác định 1 đường thẳng, người ta chọn cách thứ 2 vì chỉ cần 2 tham số (ρ, θ) tiện cho tính toán hơn. Vì θ được tính bằng radian nên có giá trị trong khoảng $[0; \pi]$ (hay là $[0; 3.14]$). Khi θ bằng 0 hoặc bằng 3.14 thì đường thẳng dựng đứng, còn θ bằng 1.57 ($\pi/2$) là nằm ngang.

Vì mỗi đường thẳng được xác định bởi 2 giá trị (ρ, θ) nên thuật toán tạo 1 mảng 2 chiều. Dòng ứng với ρ và cột ứng với θ , kích thước của mảng phụ thuộc vào bạn chọn, và tất nhiên là mảng càng lớn thì càng chính xác và tính toán càng lâu, còn mảng nhỏ thì nhanh hơn nhưng không chính xác bằng.

Khi sử dụng người dùng sẽ truyền giá trị ρ và θ họ mong muốn. Thuật toán sẽ vẽ $\rho \times \theta$ đường thẳng trên ảnh, với mỗi đường thẳng thuật toán đếm số pixel nằm trên đường thẳng đó, cứ mỗi pixel tìm thấy thì cộng thêm giá trị vào ô ứng với (ρ, θ) .

3.3. 2. Hàm houghlines trong opencv

Hàm `houghlines()` sử dụng ảnh nhị phân để xử lý.

```
void HoughLines(InputArray image, OutputArray lines, double
rho, double theta, int threshold, double srn=0, double stn=0
)
```

- **image** – ảnh input nhị phân (ảnh có thể bị thay đổi khi tính toán)
- **lines** – Output là vector chứa các đường thẳng. Mỗi đường thẳng chứa 2 giá trị (ρ, θ)
- **rho** – Khoảng cách của các đường thẳng tính bằng pixels
- **theta** – Khoảng cách góc của các đường thẳng tính bằng radians
- **threshold** – Số lần được tìm thấy phải vượt qua ngưỡng mới lấy
- **srn** – Dùng cho multi-scale
- **stn** – Cũng dùng cho multi-scale

3.3. 3. Chương trình tìm đường tròn, đường thẳng trong ảnh bằng biến đổi Hough

- **Thuật toán**

- Cho ảnh đầu vào
- Dùng hàm GaussianBlur để làm mờ ảnh
- Sử dụng Bộ lọc Canny để tìm biên
- Dùng hàm HoughLinesP để tìm đường thẳng
- Dùng hàm HoughCircles để tìm đường tròn
- Dùng vòng lặp for để vẽ đường thẳng, đường tròn lên ảnh.

- **Code chương trình**

```
#include <stdio.h>
#include <opencv2\core\core.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <iostream>
using namespace cv;
using namespace std;
void main()
{
    Mat src = imread("dongho.jpg", 1);
    Mat gray;
    cvtColor(src, gray, CV_BGR2GRAY);
    GaussianBlur(gray, gray, Size(9, 9), 2, 2);
    // dùng hàm gaussianBlur làm mờ hình ảnh

    Mat canny;
    Canny(gray, canny, 100, 200, 3, false); // Bộ lọc tìm biên
    vector<Vec4i> lines;
    HoughLinesP(canny, lines, 1, CV_PI / 180, 50, 60, 10);
    //dùng hàm HoughLinesP để tìm đường thẳng

    vector<Vec3f> circles;
    HoughCircles(gray, circles, CV_HOUGH_GRADIENT, 1, 100, 200, 100, 0, 0);
    //dùng hàm HoughCircles để tìm đường tròn

    for (int i = 0; i < lines.size(); i++)
    {
```

```

        Vec4i l = lines[i];
        line(src, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 0, 255), 2);
    }
    //dùng để vẽ đường thẳng

    for (int i = 0; i < circles.size(); i++)
    {
        Point center(cvRound(circles[i][0]), cvRound(circles[i][1])); // vẽ tâm vòng
tròn

        int radius = cvRound(circles[i][2]); // Bán kính
        circle(src, center, radius, Scalar(0, 0, 255), 2, 8, 0); // Vẽ đường viền vòng
tròn
    }
    // dùng để vẽ vòng tròn
    imshow("anh dong ho", gray);
    imshow("Anh sau khi tim thay duong thang - Duong tron", src);
    waitKey(0);
}

```

- **Ảnh kết quả**



3.4. Tìm contours trong ảnh

3.4.1. Contour là gì

Có thể hiểu contour là tập các điểm-liên-tục tạo thành một đường cong (curve) (boundary), và không có khoảng hở trong đường cong đó, đặc điểm chung trong một contour là các điểm có cùng /gần xấp xỉ một giá trị màu, hoặc cùng mật độ. Contour là một công cụ hữu ích được dùng để phân tích hình dạng đối tượng, phát hiện đối tượng và nhận dạng đối tượng.

3.4.2. Tìm contours trong ảnh với Opencv

Trong OpenCV, contour hoạt động trên ảnh đơn kênh, nhưng hiệu quả nhất là ảnh nhị phân. Việc tìm kiếm contours trong OpenCV khá đơn giản bằng hàm findContours(). Hàm này đưa ra danh sách các contours tìm được, với mỗi contour là một danh sách các tọa độ điểm.

```
Void findContours(InputOutputArray image, OutputArrayOfArrays  
contours, int mode, int method, Point offset=Point())
```

Trong đó:

- image: ảnh đầu vào (đơn kênh hoặc ảnh nhị phân)
- contours: danh sách contours tìm được
- mode: cách truy vấn các contours, thông thường mình đặt là **CV_RETR_TREE**
- method: phương thức ước lượng số đỉnh của từng contour. Có 2 phương thức thường dùng nhất là **CV_CHAIN_APPROX_NONE** (liệt kê toàn bộ các đỉnh của đa giác contour) và **CV_CHAIN_APPROX_SIMPLE** (hạn chế số đỉnh phải lưu trữ).
- Chương trình minh họa:

```
#include "opencv2/core.hpp"  
#include "opencv2/highgui.hpp"  
#include "opencv2/imgproc.hpp"  
#include <vector>  
#include <iostream>  
using namespace cv;  
int main()  
{  
    Mat img = imread("example.png");  
    imshow("Original image", img);  
    Mat grayImg;  
    cvtColor(img, grayImg, COLOR_BGR2GRAY);  
    imshow("Gray image", grayImg);  
    Mat binImg;  
    threshold(grayImg, binImg, 100, 255, CV_THRESH_BINARY);  
    imshow("Binary image", binImg);  
    std::vector< std::vector<Point> > contours;
```

```

        findContours(binImg, contours, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE);

        std::cout << "Contours number found: " << contours.size() << std::endl;
        for (int iContours = 0; iContours < contours.size(); iContours++)
        {
            std::cout << "Contour #" << iContours + 1 << ": { ";
            for (Point p : contours[iContours])
                std::cout << "( " << p.x << ", " << p.y << " ) ";
            std::cout << "}\n";
        }
        waitKey();
        return 0;
}

```

Kết

quả:



```

E:\workspace\Demo\y64\Debug\Project.exe
Contours number found: 61
Contour #1: { ( 635, 381 ) ( 635, 382 ) ( 634, 383 ) ( 634, 392 ) ( 633, 393 ) ( 633, 425 ) ( 633, 421 ) ( 635, 419 ) (
635, 415 ) ( 634, 414 ) ( 634, 399 ) ( 635, 398 ) }
Contour #2: { ( 635, 332 ) ( 635, 335 ) ( 634, 336 ) ( 634, 339 ) ( 635, 340 ) ( 635, 342 ) ( 634, 343 ) ( 635, 344 ) (
635, 345 ) ( 634, 346 ) ( 634, 355 ) ( 635, 356 ) ( 635, 367 ) }
Contour #3: { ( 635, 327 ) ( 635, 330 ) }
Contour #4: { ( 636, 307 ) ( 636, 318 ) }
Contour #5: { ( 9, 62 ) }
Contour #6: { ( 632, 34 ) ( 632, 35 ) }
Contour #7: { ( 632, 32 ) }
Contour #8: { ( 632, 0 ) ( 632, 1 ) ( 633, 2 ) ( 633, 3 ) ( 634, 4 ) ( 634, 5 ) ( 635, 6 ) ( 635, 8 ) ( 636, 9 ) ( 636,
14 ) ( 637, 15 ) ( 637, 19 ) ( 638, 20 ) ( 638, 46 ) ( 637, 47 ) ( 637, 76 ) ( 636, 77 ) ( 636, 81 ) ( 637, 82 ) ( 637,
84 ) ( 636, 85 ) ( 636, 112 ) ( 637, 113 ) ( 636, 114 ) ( 636, 167 ) ( 635, 168 ) ( 636, 169 ) ( 635, 170 ) ( 635, 172 ) (
636, 173 ) ( 635, 174 ) ( 635, 188 ) ( 636, 181 ) ( 636, 195 ) ( 635, 196 ) ( 635, 201 ) ( 636, 202 ) ( 636, 226 ) (
635, 227 ) ( 635, 233 ) ( 636, 234 ) ( 636, 241 ) ( 635, 242 ) ( 636, 243 ) ( 636, 254 ) ( 635, 255 ) ( 636, 256 ) ( 636,
257 ) ( 635, 258 ) ( 636, 259 ) ( 635, 260 ) ( 635, 261 ) ( 636, 264 ) ( 636, 284 ) ( 635, 285 ) ( 635, 294 ) ( 636, 2
95 ) ( 636, 385 ) ( 636, 381 ) ( 637, 300 ) ( 637, 273 ) ( 638, 272 ) ( 638, 269 ) ( 637, 268 ) ( 637, 256 ) ( 638, 255
) ( 638, 215 ) ( 637, 214 ) ( 637, 212 ) ( 636, 211 ) ( 636, 166 ) ( 637, 165 ) ( 637, 162 ) ( 636, 161 ) ( 636, 150 ) (
637, 158 ) ( 636, 157 ) ( 636, 146 ) ( 637, 145 ) ( 637, 57 ) ( 638, 56 ) ( 638, 50 ) ( 639, 40 ) ( 639, 16 ) ( 638, 15
) ( 638, 10 ) ( 637, 9 ) ( 637, 6 ) ( 635, 4 ) ( 635, 2 ) ( 633, 0 ) }
Contour #9: { ( 622, 0 ) ( 622, 1 ) ( 624, 3 ) ( 624, 4 ) ( 625, 4 ) ( 626, 5 ) ( 626, 8 ) ( 627, 9 ) ( 627, 10 ) ( 628,
11 ) ( 628, 13 ) ( 629, 14 ) ( 629, 16 ) ( 630, 17 ) ( 629, 18 ) ( 629, 19 ) ( 630, 20 ) ( 630, 28 ) ( 631, 28 ) ( 632,
29 ) ( 633, 29 ) ( 632, 29 ) ( 631, 28 ) ( 632, 27 ) ( 632, 26 ) ( 633, 25 ) ( 633, 20 ) ( 632, 19 ) ( 632, 14 ) ( 631,
13 ) ( 631, 11 ) ( 630, 11 ) ( 629, 10 ) ( 629, 6 ) ( 628, 5 ) ( 627, 5 ) ( 625, 3 ) ( 625, 2 ) ( 624, 1 ) ( 624, 0 ) }
Contour #10: { ( 21, 0 ) ( 19, 2 ) ( 18, 2 ) ( 15, 5 ) ( 14, 5 ) ( 14, 7 ) ( 11, 10 ) ( 11, 13 ) ( 10, 14 ) ( 10, 17 ) (
9, 18 ) ( 8, 18 ) ( 9, 18 ) ( 10, 19 ) ( 10, 20 ) ( 9, 21 ) ( 8, 21 ) ( 10, 23 ) ( 9, 24 ) ( 7, 24 ) ( 7, 25 ) ( 8, 25
) ( 9, 26 ) ( 8, 27 ) ( 8, 28 ) ( 7, 29 ) ( 8, 29 ) ( 9, 30 ) ( 7, 32 ) ( 7, 33 ) ( 9, 33 ) ( 11, 35 ) ( 11, 36 ) ( 10,
17 ) ( 9, 36 ) ( 7, 36 ) ( 8, 37 ) ( 8, 40 ) ( 7, 41 ) ( 8, 42 ) ( 7, 43 ) ( 8, 42 ) ( 9, 42 ) ( 10, 43 ) ( 10, 44 ) ( 1
1, 45 ) ( 10, 46 ) ( 9, 46 ) ( 10, 47 ) ( 10, 49 ) ( 11, 50 ) ( 11, 52 ) ( 10, 53 ) ( 8, 53 ) ( 9, 54 ) ( 10, 53 ) ( 11,

```

3.4.3. Vẽ contours trong ảnh với Opencv

Để trực quan hóa các tọa độ contours thu được từ hàm **findContours()**, chúng ta sử dụng hàm drawContours() để vẽ lên ảnh tất cả các contours tìm được ở phần trên.

```
void drawContours(InputOutputArray image, InputArrayOfArrays contours, int
contourIdx, const Scalar& color, int thickness=1, int lineType=8, InputArray
hierarchy=noArray(), int maxLevel=INT_MAX, Point offset=Point())
```

Trong đó:

- image: ảnh đầu vào
- contours: danh sách các contour tìm được từ hàm findContours()
- contourIdx: thứ tự của contour cần vẽ trong biến contours
- color: màu vẽ

chương trình minh họa:

```
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <vector>
#include <iostream>
using namespace cv;
int main()
{
```

Kết quả:



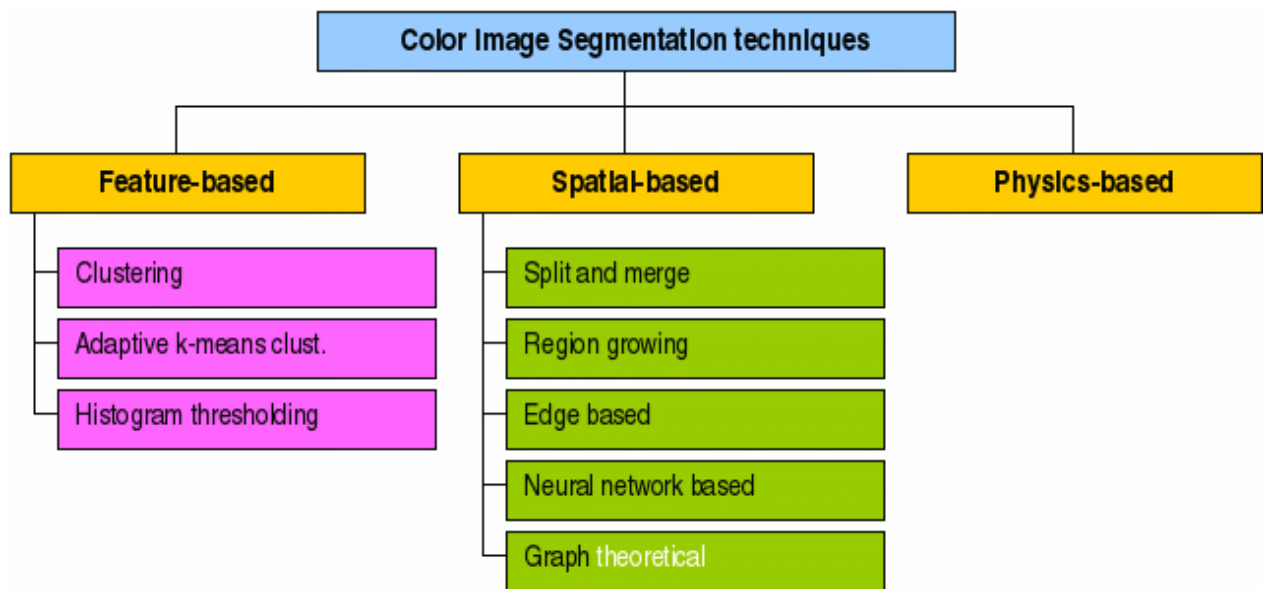
CHƯƠNG 4: PHÂN ĐOẠN ẢNH

4.1. Giới thiệu các phương pháp phân đoạn ảnh

Phân đoạn ảnh là một thao tác ở mức thấp trong toàn bộ quá trình xử lý ảnh. Quá trình này thực hiện việc phân vùng ảnh thành các vùng rời rạc và đồng nhất với nhau hay nói cách khác là xác định các biên của các vùng ảnh đó. Các vùng ảnh đồng nhất này thông thường sẽ tương ứng với toàn bộ hay từng phần của các đối tượng thật sự bên trong ảnh. Vì thế, trong hầu hết các ứng dụng của lĩnh vực xử lý ảnh (image processing), thị giác máy tính, phân đoạn ảnh luôn đóng một vai trò cơ bản và thường là bước tiền xử lý đầu tiên trong toàn bộ quá trình trước khi thực hiện các thao tác khác ở mức cao hơn như nhận dạng đối tượng, biểu diễn đối tượng, nén ảnh dựa trên đối tượng, hay truy vấn ảnh dựa vào nội dung ... Vào những thời gian đầu, các phương pháp phân vùng ảnh được đưa ra chủ yếu làm việc trên các ảnh mức xám do các hạn chế về phương tiện thu thập và lưu trữ. Ngày nay, cùng với sự phát triển về các phương tiện thu nhận và biểu diễn ảnh, các ảnh màu đã hầu như thay thế hoàn toàn các ảnh mức xám trong việc biểu diễn và lưu trữ thông tin do các ưu thế vượt trội hơn hẳn so với ảnh mức xám. Do đó, các kỹ thuật, thuật giải mới thực hiện việc phân vùng ảnh trên các loại ảnh màu liên tục được phát triển để đáp ứng các nhu cầu mới. Các thuật giải, kỹ thuật này thường được phát triển dựa trên nền tảng các thuật giải phân vùng ảnh mức xám đã có sẵn.

Phân đoạn ảnh là chia ảnh thành các vùng không trùng lắp. Mỗi vùng gồm một nhóm pixel liên thông và đồng nhất theo một tiêu chí nào đó. Tiêu chí này phụ thuộc vào mục tiêu của quá trình phân đoạn. Ví dụ như đồng nhất về màu sắc, mức xám, kết cấu, độ sâu của các layer... Sau khi phân đoạn mỗi pixel chỉ thuộc về một vùng duy nhất. Để đánh giá chất lượng của quá trình phân đoạn là rất khó. Vì vậy trước khi phân đoạn ảnh cần xác định rõ mục tiêu của quá trình phân đoạn là gì. Xét một cách tổng quát, ta có thể chia các hướng tiếp cận phân đoạn ảnh thành ba nhóm chính như sau:

- Các kỹ thuật phân đoạn ảnh dựa trên không gian đặc trưng.
- Các kỹ thuật dựa trên không gian ảnh.
- Các kỹ thuật dựa trên các mô hình vật lý.



4.2. Chọn ngưỡng cố định

Đây là một phương pháp chọn ngưỡng độc lập với dữ liệu ảnh. Nếu chúng ta biết trước là chương trình ứng dụng sẽ làm việc với các ảnh có độ tương phản khá cao, trong đó các đối tượng quan tâm rất tối còn nền gần như là đồng nhất và rất sáng thì việc chọn ngưỡng $T=128$ (xét trên thang độ sáng từ 0 tới 255) là một giá trị chọn khá chính xác. Chính xác ở đây hiểu theo nghĩa là số các điểm ảnh bị phân lớp sai là cực tiểu.

4.3. Chọn ngưỡng dựa trên lược đồ

Trong hầu hết các trường hợp, ngưỡng được chọn từ lược đồ sáng của vùng hay ảnh cần phân đoạn. Có rất nhiều kỹ thuật chọn ngưỡng tự động xuất phát từ lược đồ xám $\{h[b] | b=0,1,2 \dots 2^B - 1\}$ đã được đưa ra. Những kỹ thuật phổ biến sẽ được trình bày dưới đây. Nhưng kỹ thuật này có thể tận dụng những lợi thế do sự làm trơn dữ liệu lược đồ ban đầu mang lại nhằm loại bỏ những giao động nhỏ về độ sáng. Tuy nhiên các thuật toán làm trơn cần phải cẩn thận, không được làm dịch chuyển các vị trí đỉnh của lược đồ. Nhận xét này dẫn tới thuật toán làm trơn dưới đây :

$$h_{smooth}[b] = \frac{1}{W} \sum_{w=-(W-1)/2}^{(W-1)/2} h_{raw}[b-w] \quad W \text{ lẻ}$$

Trong đó, W thường được chọn là 3 hoặc 5

4.3. 1. Thuật toán đẳng hiệu

Đây là kỹ thuật chọn ngưỡng theo kiểu lặp do Ridler và Calvard đưa ra. Thuật toán được mô tả như sau:

- B1 : Chọn giá trị ngưỡng khởi động $\theta_0 = 2^{B-1}$
- B2 : Tính các trung bình mẫu ($m_{f,o}$) của những điểm ảnh thuộc đối tượng và ($m_{b,o}$) của những điểm ảnh nền.
- B3 : Tính ngưỡng trung gian theo công thức :

$$\theta_k = \frac{m_{f,k-1} + m_{b,k-1}}{2} \text{ với } k = 1, 2, \dots$$

- B4 : nếu $\theta_k = \theta_{k-1}$: kết thúc và dừng thuật toán

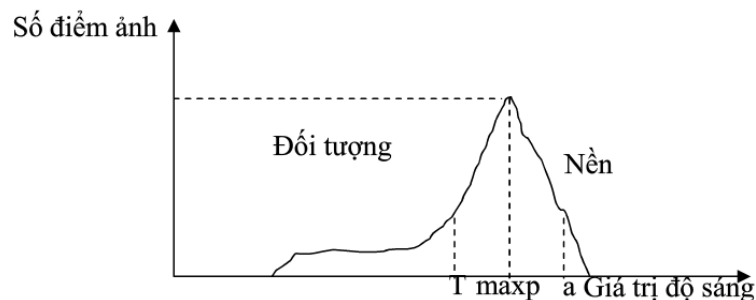
Ngược lại : tiếp tục bước 2.

4.3. 2. Thuật toán đối xứng nền

Kỹ thuật này dựa trên sự giả định là tồn tại hai đỉnh phân biệt trong lược đồ nằm đối xứng qua đỉnh có giá trị lớn nhất trong phần lược đồ thuộc về các điểm ảnh nền. Kỹ thuật này có thể tận dụng ưu điểm của việc làm trơn được mô tả trong chương trình ????. Đỉnh cực đại maxp tìm được nhờ tiến hành tìm giá trị cực đại trong lược đồ. Sau đó thuật toán sẽ áp dụng ở phía không phải là điểm ảnh thuộc đối tượng ứng với giá trị cực đại đó nhằm tìm ra giá trị độ sáng a ứng với giá trị phần trăm p% mà $P(a) = p\%$, trong đó P(a) là hàm phân phối xác suất về độ sáng được định nghĩa như sau :

Định nghĩa : [Hàm phân phối xác suất về độ sáng]

Hàm phân phối xác suất P(a) thể hiện xác suất chọn được một giá trị độ sáng từ một vùng ánh sáng cho trước, sao cho giá trị này không vượt qua một giá trị này cho trước, sao cho giá trị này không vượt quá giá trị sáng cho trước a. Khi a biến thiên từ $-\infty$ đến $+\infty$, P(a) sẽ nhận các giá trị từ 0 đến 1, P(a) là hàm đơn điệu không giảm theo a, do vậy $dP/da \geq 0$



Hình 1. Hình minh họa thuật toán đối xứng nền

Ở đây ta đang giả thiết là ảnh có các đối tượng tối trên nền sáng. Giả sử mức là 5% thì có nghĩa là ta phải ở bên phải đỉnh maxp một giá trị a sao cho $P(a) = 95\%$. Do tính đối xứng đã giả định ở trên, chúng ta sử dụng độ dịch chuyển về phía trái của điểm cực đại tìm giá trị ngưỡng T :

$$T = \text{maxp} - (a - \text{maxp})$$

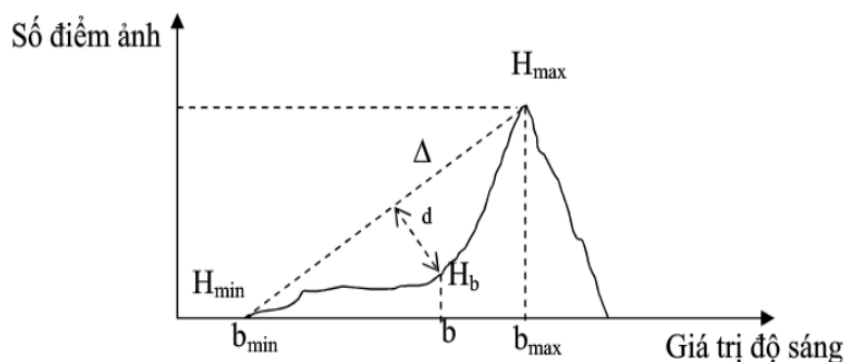
Kỹ thuật này dễ dàng điều chỉnh được cho phù hợp với tình huống ảnh có các đối tượng sáng trên một nền tối.

4.3. 3. Thuật toán tam giác

Khi một ảnh có các điểm ảnh thuộc đối tượng tạo nên một đỉnh yếu trong lược đồ ảnh thì thuật toán tam giác hoạt động rất hiệu quả. Thuật toán này do Zack đề xuất và được mô tả như sau:

- Bước 1 : xây dựng đường thẳng Δ là đường nối 2 điểm là (H_{\max}, b_{\max}) và (H_{\min}, b_{\min})
Trong đó H_{\max} là điểm Histogram ứng với độ sáng nhỏ nhất b_{\min} .
- Bước 2: Tính khoảng cách d từ H_b của lược đồ (ứng với điểm sáng b) đến Δ
Trong đó $b \in [b_{\max}, b_{\min}]$.
- Bước 3: Chọn ngưỡng $T = \text{Max}\{H_b\}$

Minh họa thuật toán tam giác bởi hình vẽ như sau :



Hình 2. Minh họa thuật toán tam giác

4.3. 4. Chọn ngưỡng đối với Bimodal Histogram

Ngưỡng T được chọn ở tại vị trí cực tiểu địa phương của Histogram nằm giữa hai đỉnh của Histogram. Điểm cực đại địa phương của Histogram có thể dễ dàng được phát hiện bằng cách sử dụng biến đổi chóp mũ (top hat) do Meyer đưa ra : phụ thuộc vào tình huống

chúng ta phải làm việc là đối với đối tượng sáng trên nền tối hay đối tượng tối trên nền sáng mà phép biến đổi top hat sẽ có một trong hai dạng sau:

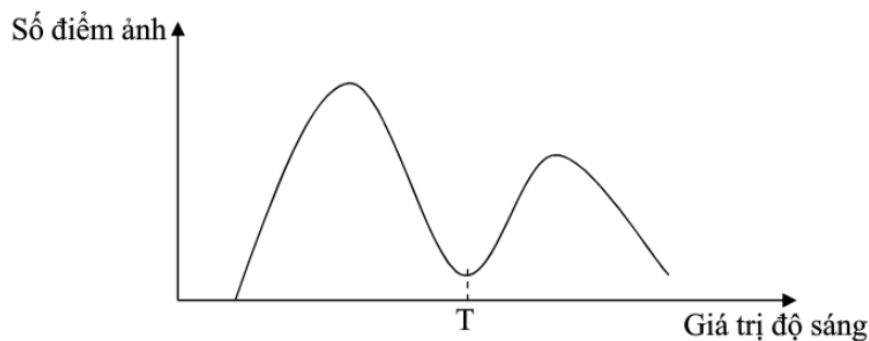
a/Các đối tượng sáng

$$TopHat(A, B) = A - (A \circ B) = A - \max_B(\min_B(A))$$

b/Các đối tượng tối

$$TopHat(A, B) = A - (A \circ B) = A - \min_B(\max_B(A))$$

Việc tính toán giá trị cực tiểu địa phương của Histogram thì khó nếu histogram nhiều. Do đó, trong trường hợp này nên làm trơn histogram, ví dụ sử dụng thuật toán.



Hình 3 . Bimodal Histogram

Trong một số ứng dụng nhất định, cường độ của đối tượng hay nền tối thay đổi khá chậm. Trong trường hợp này, histogram ảnh có thể không chứa hai thùy phân biệt rõ ràng, vì vậy có thể sử dụng ngưỡng thay đổi theo không gian. Hình ảnh được chia thành các khối vuông, histogram và ngưỡng được tính cho mỗi khối vuông tương ứng.

4.4. Phương pháp phân đoạn dựa trên ngưỡng cục bộ thích nghi

Số ngưỡng cục bộ và giá trị của chúng không được chỉ định trước mà được trích lọc thông qua quá trình kiểm tra các thông tin cục bộ. Giải thuật gồm các bước tuần tự như sau:

- Áp dụng giải thuật Watershed chia ảnh thành rất nhiều vùng con.

- Trộn các vùng và đồng thời phát hiện ngưỡng cục bộ. Ngưỡng được tính từ thông tin cục bộ của vùng và các vùng lân cận

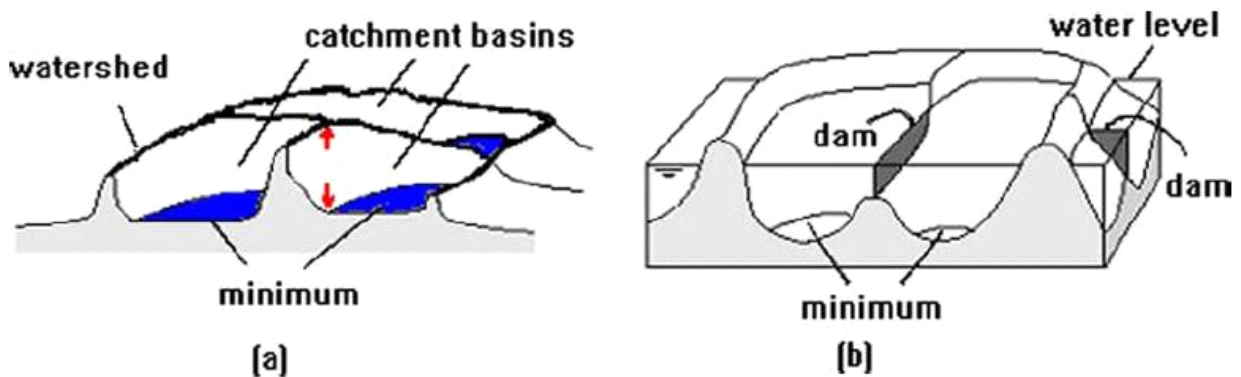
Giải thuật này cho kết quả tương đối tin cậy trên nhiều loại ảnh khác nhau

4.4. 1. Phân đoạn bằng Watershed

Dữ liệu đầu vào của giải thuật Watershed là một ảnh xám. Vì vậy, trước tiên ta biến đổi ảnh đầu vào I thành ảnh xám. Sau đó, dùng giải thuật tìm cạnh Canny [20] để lấy cường độ gradient, kí hiệu là I_G . Với ảnh gradient nhận được, ta hình liên tưởng đến một lược đồ địa hình, vùng có độ xám cao hơn là vùng trũng hơn và ngược lại. Tại mỗi pixel, việc đánh giá sẽ dựa vào giá trị mức xám của pixel đó.

Giải thuật định nghĩa hai thuật ngữ là vùng chứa nước (catchment basin) và đập ngăn nước (dams). Mỗi catchment basin được kết hợp với giá trị M nhỏ nhất. M là tập hợp các pixel liên thông mà một giọt nước rơi xuống từ pixel bất kì thuộc catchment basin này cứ rơi cho đến khi nó đạt được giá trị nhỏ nhất M. Trên đường rơi xuống, giọt nước chỉ đi qua những pixel thuộc về catchment basin này.

Dam thực chất là những đường phân nước, chúng tập hợp các pixel làm nhiệm vụ phân cách các catchment basin. Vì vậy, giọt nước rơi từ một bên của dams sẽ đạt trị nhỏ nhất của một catchment basin, trong khi đó giọt nước rơi từ cạnh khác của dam lại đạt trị nhỏ nhất trong catchment basin khác.

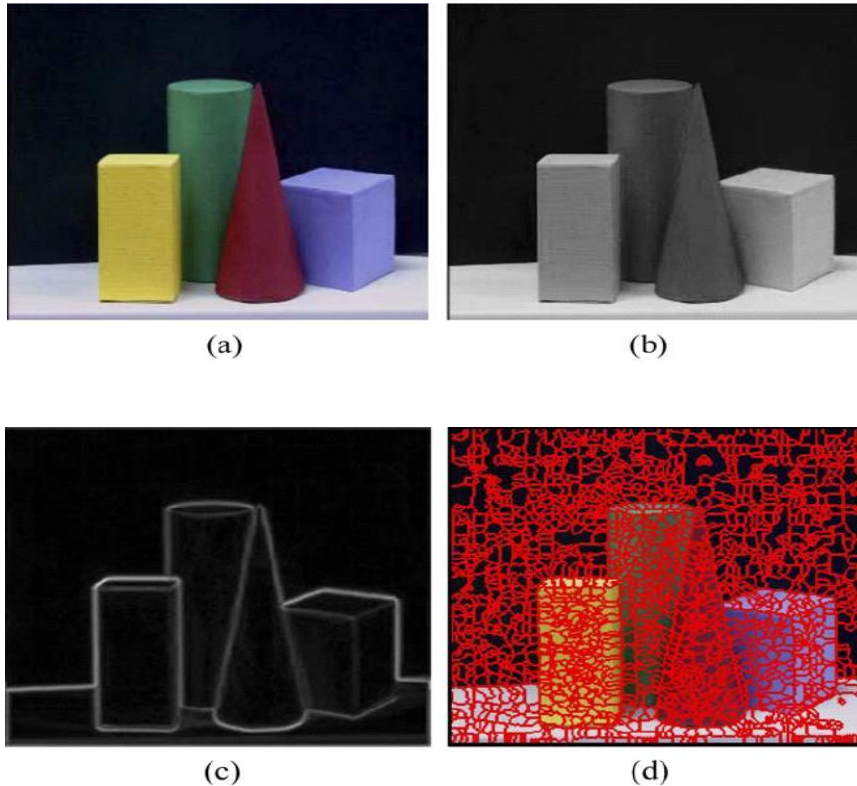


Áp dụng giải thuật watershed, phiên bản của Vincent và Soille.. Phiên bản này mô phỏng việc ngấm nước dần dần bề mặt địa hình của ảnh từ vùng thấp nhất cho đến khi mọi pixel của ảnh đều được ngấm trong nước. Giải thuật gồm hai bước: sắp thứ tự và làm ngập nước.

Ở bước thứ nhất, ta sắp xếp các pixel theo thứ tự tăng dần của cường độ xám. Kế đến, trong bước làm ngập nước, giải thuật quét các pixel theo trình tự đã sắp xếp để xây dựng các catchment basin. Mỗi catchment basin có một nhãn phân biệt. Bạn hãy thử hình dung ta đem nhúng nước một bề mặt địa hình, bắt đầu tại điểm thấp nhất của mặt địa rồi cho nước dâng dần lên. Khi nước trong các vũng cạnh nhau có thể hoà vào nhau tại một điểm, tại đó ta xây dựng một đập chắn nước, rồi lại tiếp tục cho nước dâng lên. Quá trình xây đập chắn giữa các vũng và cho nước dâng cứ lặp đi lặp lại cho đến khi mọi điểm của bề mặt địa hình đều được ngấm nước.

Trở lại giải thuật, ta làm tương tự, tại một điểm mà nước trong các catchment basin có thể hoà vào nhau, ta xây dựng một đập chắn nước – dam. Cứ như thế, lặp quá trình cho nước dâng lên và xây dựng dam tại những điểm nước của các catchment basin có thể hoà lẫn vào nhau cho đến khi mọi điểm ảnh đều nằm trong nước. Khi đó, ta nhận được ảnh gồm vô số vùng con, mỗi vùng con tương ứng với một catchment basin, còn biên của mỗi vùng chính là dam. Bạn xem hình 4 minh họa quá phân ảnh ban đầu (a) thành vô số vùng con (d). Trước tiên ảnh gốc 4a được biến đổi thành ảnh xám 4b. Kế đến, áp dụng giải thuật tìm cạnh Canny trên ảnh xám gradient ở hình 4b, ta được ảnh 4c chỉ gồm các đường nét. Đồng thời, áp dụng giải thuật watershed trên ảnh xám ta được hình 4d, chứa vô số vùng con.

Như vậy khi áp dụng giải thuật watershed vào ảnh I_G , ta nhận được ảnh kết quả gồm n vùng không trùng lặp. Do các vùng này sẽ được trộn trong giai đoạn trộn tiếp theo nên chúng tôi đặt đánh dấu chúng bằng kí hiệu $R_i^{m_i}$, $i = 1, \dots, n$, $m_i = 1, \dots, M_i$, với n là số lượng vùng và M_i là số lần trộn của $R_i^{m_i}$ trong quá trình trộn. R_i^0 , $i=1, \dots, n$ là tập các vùng khởi tạo, hay nói cách khác chúng là kết quả của giải thuật watershed trước khi quá trình trộn lặp của giai đoạn hai bắt đầu.



Hình 4 : Hình minh họa

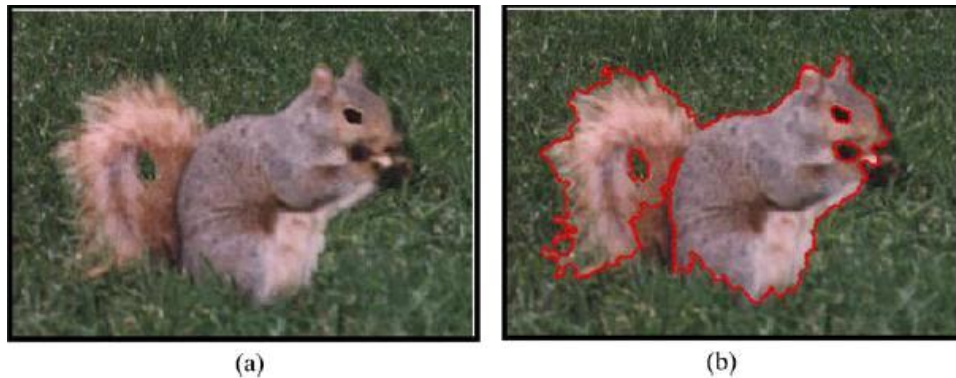
(a) Ảnh gốc ban đầu. (b) Ảnh xám. (c) Ảnh xám gradient sau khi đã áp dụng giải thuật tìm cạnh Canny. (d) Ảnh phân đoạn nhận được từ việc áp dụng giải

4.4. 2. Tìm ngưỡng cục bộ thích nghi

Mặc dù phần mô tả quá trình trộn đã hoàn chỉnh nhưng ta vẫn chưa xác định được khi nào thì giải thuật dừng. Hay nói cách khác, ta vẫn chưa biết cách xác định vùng nào không trộn được và thời điểm nào thì không trộn. Như vậy, chúng ta cần có cơ chế tự động rút trích thông tin về ngưỡng cục bộ thông qua việc theo dõi sự thay đổi của mỗi vùng trong quá trình trộn. Các ngưỡng này sẽ cho biết có thể trộn một vùng hay không. Như thế, các ngưỡng này giúp hình thành phân vùng hoàn chỉnh cuối cùng.

Như chúng ta đã biết quá trình phân đoạn là thao tác cục bộ, nên không phải mọi bước trộn cục bộ đều dừng đồng thời. Do đó việc sử dụng ngưỡng toàn cục là không đủ vì các vùng thường tách biệt với xung quanh nó bởi những ngưỡng khác nhau vào những lần xử lý khác nhau. Tuy nhiên trong một vài trường hợp thì ngưỡng toàn cục lại phù hợp. Ví dụ ở hình 5 mô tả một trường hợp ngoại lệ, chỉ dùng một ngưỡng toàn cục mà vẫn cho kết quả phân đoạn chính xác. Lý do là ảnh ví dụ chỉ chứa một đối tượng đồng nhất về màu

sắc, đồng thời phần nền cũng có màu đồng nhất. Trong trường hợp này chỉ cần một ngưỡng cho quá trình trộn là đủ. Quá trình trộn sẽ dừng khi trọng số của các cạnh khảo sát lớn hơn ngưỡng chọn trước, cụ thể trong ví dụ này là 100. Bạn xem kết quả phân đoạn bằng ngưỡng trên ở hình 5b. Trong thực tế, các ảnh phân tích thường chứa nhiều hơn hai vùng nên rất khó phân đoạn nếu chỉ dùng một ngưỡng toàn cục.

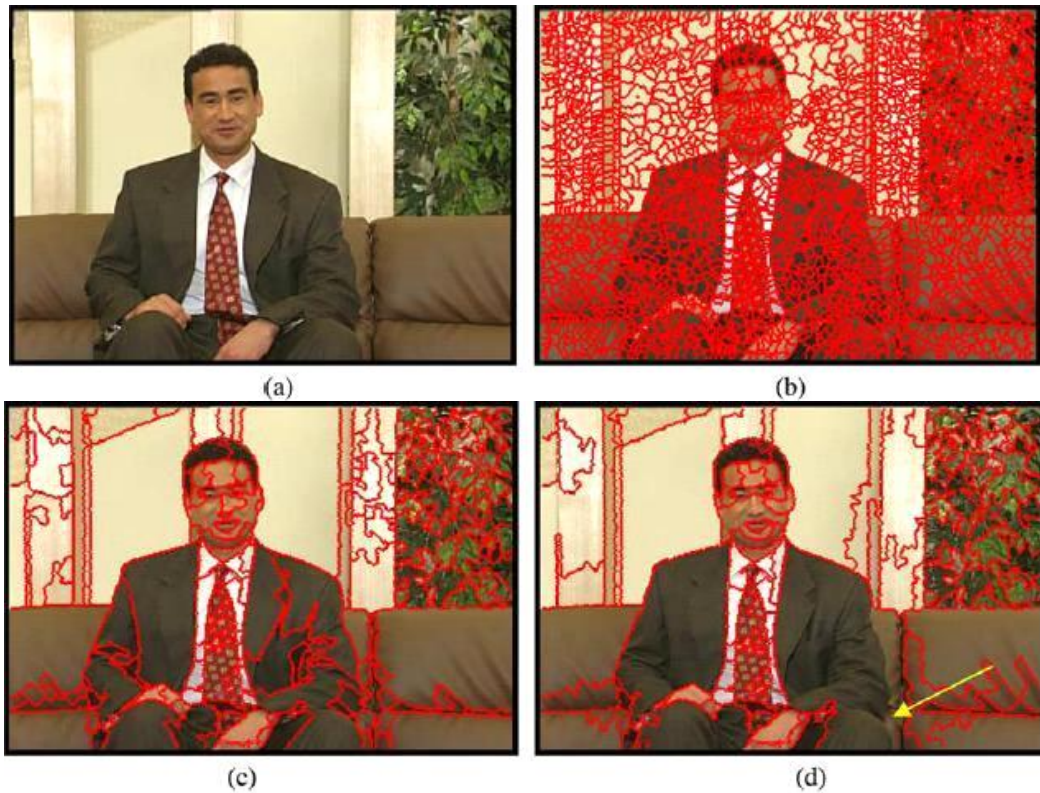


Hình 5. (a) Ảnh gốc. (b) Kết quả phân đoạn bằng ngưỡng toàn cục 100.

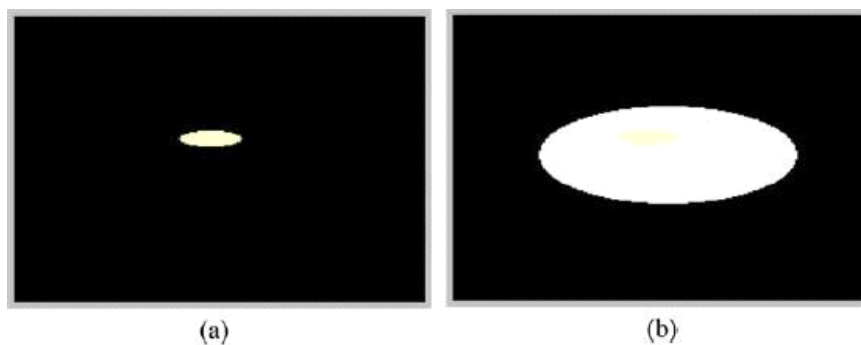
Bạn sẽ cảm nhận được nhu cầu dùng ngưỡng cục bộ thay cho ngưỡng toàn cục khi xem hình 6. Ta có hình gốc 6a, hình 6b là kết quả của giải thuật watershed. Với ngưỡng toàn cục $t = 20$ ta được kết quả phân đoạn hình 6c, còn hình 6d là kết quả tương ứng với ngưỡng toàn cục $t = 30$. Trong hình 6c, mọi vùng đều đồng nhất và có thể lớn hơn. Tuy nhiên, khi ngưỡng tăng lên 30 như ở hình 6d, các vùng nhìn bằng mắt thường là đồng nhất như mặt và ghế lại bị phân quá nhỏ. Trong khi đó, vùng chỉ ra bởi mũi tên vàng vẫn chưa đồng nhất. Để phân nó thành nhiều vùng đồng nhất thì ngưỡng phải nhỏ hơn 30, khi đó việc trộn hai vùng không đồng nhất là áo khoác của người đàn ông và cái ghế sẽ không được thực hiện.

Chúng ta đã nhận biết được nhu cầu cần thiết tính ngưỡng cục bộ, nhưng tính ngưỡng thế nào và dựa vào yếu tố gì thì cần xem xét tiếp. Việc tính ngưỡng cục bộ phải dựa vào các thông tin cục bộ, liên quan đến vùng đang xét và những vùng lân cận xung quanh nó. Thế nhưng tại sao phải xét vùng lân cận? Ta phải xét các vùng lân cận vì một vùng thường bị ảnh hưởng bởi các vùng xung quanh nó. Bạn xem ví dụ hình 8 để thấy mối quan hệ khăng khít giữa một vùng và các vùng lân cận nó, cùng một vùng nhưng nếu đặt vào giữa những vùng lân cận khác nhau thì cảm nhận thị giác sẽ rất khác nhau. Trong hình 7a, đối tượng hình ellipse màu vàng nổi bật trên nền màu đen, khác hẳn với hình 7b, cũng đối tượng

ellipse màu vàng này nhưng gần như hòa vào màu nền trắng xung quanh nó, rất khó nhận biết.



Hình 6. (a) Ảnh gốc (b) Sau khi áp dụng giải thuật watershed.
(c) Sau khi hoàn thành quá trình trộn dùng một ngưỡng toàn cục $t=20$.
(d) Sau khi trộn dùng một ngưỡng toàn cục $t=30$.



Hình 7. Vùng sáng elip hiển thị khác nhau khi do nền khác nhau.

4.5. Phân đoạn ảnh sử dụng openCV

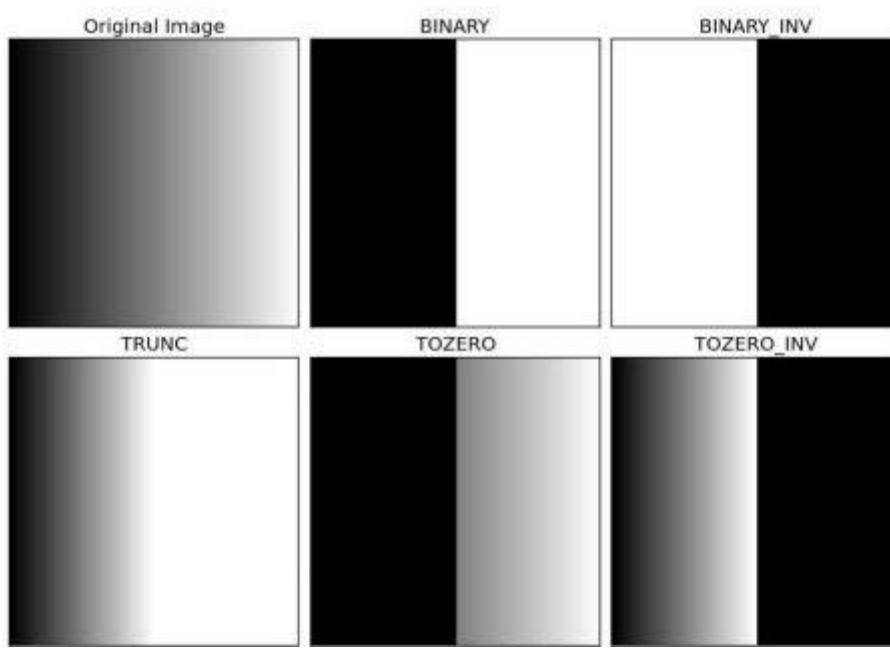
Nếu pixel có giá trị lớn hơn giá trị ngưỡng thì nó được gán 1 giá trị (thường là 1), ngược lại nhỏ hơn giá trị ngưỡng thì nó được gán 1 giá trị khác (thường là 0).

`double threshold(Mat src, Mat dst, double thresh, double maxval, int type)`

Hàm sử dụng là **threshold**, tham số đầu tiên là 1 ảnh xám, tham số thứ 2 là giá trị ngưỡng, tham số thứ 3 maxval là giá trị được gán nếu giá pixel lớn hơn giá trị ngưỡng, tham số thứ 4 là loại phân ngưỡng. Tùy theo các loại phân ngưỡng mà pixel được gán giá trị khác nhau:

- **THRESH_BINARY**
 - Nếu giá trị pixel lớn hơn ngưỡng thì gán bằng maxval
 - Ngược lại bằng gán bằng 0
- **THRESH_BINARY_INV**
 - Nếu giá trị pixel lớn hơn ngưỡng thì gán bằng 0
 - Ngược lại bằng gán bằng maxval
- **THRESH_TRUNC**
 - Nếu giá trị pixel lớn hơn ngưỡng thì gán giá trị bằng ngưỡng
 - Ngược lại giữ nguyên giá trị
- **THRESH_TOZERO**
 - Nếu giá trị pixel lớn hơn ngưỡng thì giữ nguyên giá trị
 - Ngược lại gán bằng 0
- **THRESH_TOZERO_INV**
 - Nếu giá trị pixel lớn hơn ngưỡng thì gán giá trị bằng 0
 - Ngược lại giữ nguyên

Hình minh họa trực quan với ngưỡng bằng 127:



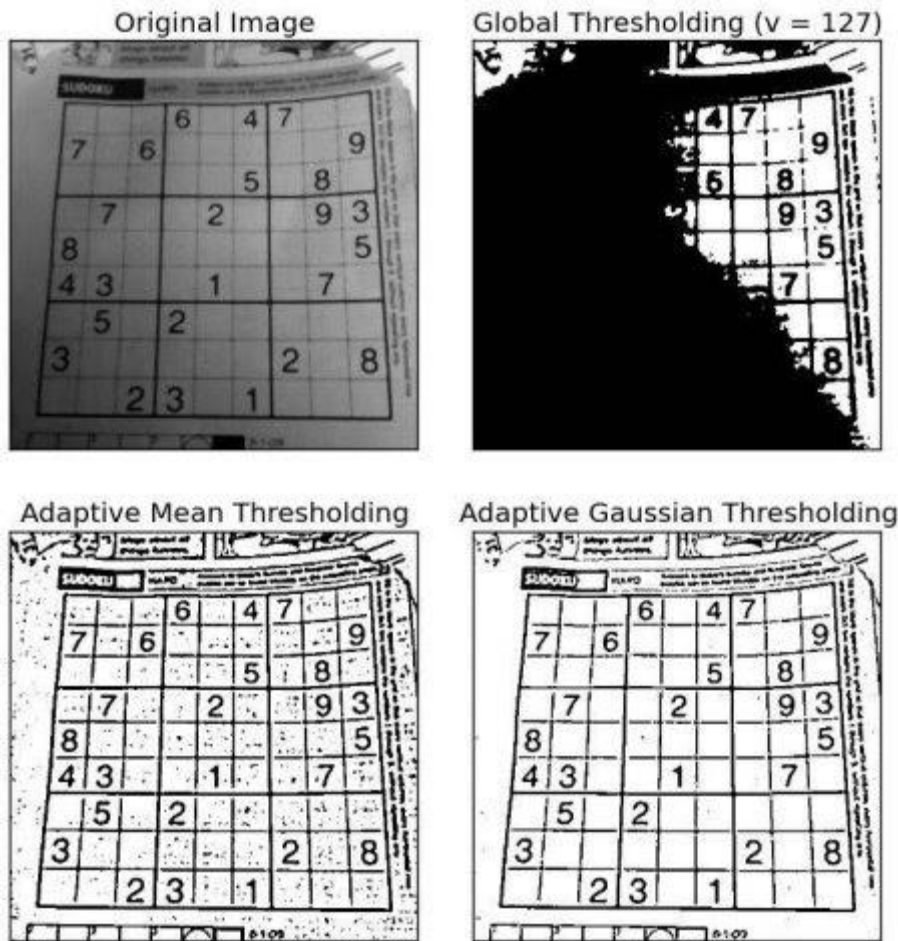
4.5.1. Adaptive Thresholding (phân ngưỡng thích nghi)

Phương pháp phân ngưỡng ở trên không phù hợp cho nhiều trường hợp, như là ánh sáng không đồng đều trên ảnh. Trong trường hợp đó chúng ta dùng hàm `adaptiveThreshold()`. Phương thức này tính giá trị trung bình của các n điểm xung quanh pixel đó rồi trừ cho C chứ không lấy ngưỡng cố định (n thường là số lẻ, còn C là số nguyên bất kỳ).

```
void adaptiveThreshold(Mat src, Matdst, double maxValue, int adaptiveMethod, int thresholdType, int blockSize, double C)
```

Ngoài các tham số giống như phân ngưỡng thường, `adaptiveThreshold` có thêm các tham số:

- Phương thức:
 - `ADAPTIVE_THRESH_MEAN_C`: giá trị của pixel phụ thuộc vào các pixel lân cận
 - `ADAPTIVE_THRESH_GAUSSIAN_C`: giá trị của pixel cũng phụ thuộc vào các pixel lân cận, tuy nhiên được khử nhiễu
- Block Size: số pixel lân cận dùng để tính toán
- C : hằng số trừ đi giá trị trung bình



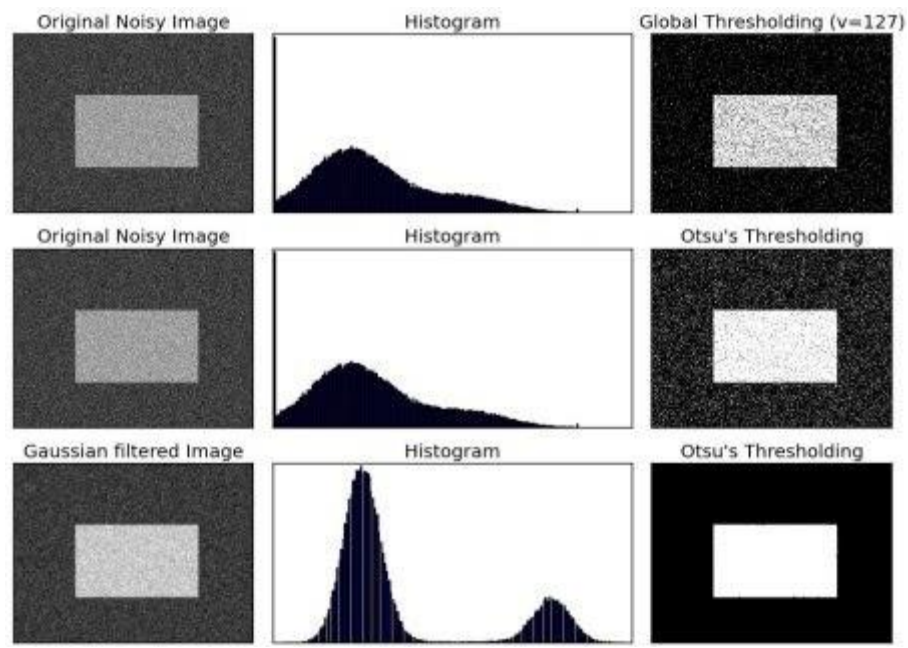
Hình minh họa cho các phương pháp phân ngưỡng khác nhau.

4.5.2. Nhị phân hóa bằng thuật toán Otsu

Trong phương thức phân ngưỡng thường, chúng lấy ngưỡng tùy ý, nhưng giá trị ngưỡng đó cho kết quả tốt hay ko? Câu trả lời là thử mới biết.

Nhưng với ảnh có 2 đỉnh trong histogram, tức là ảnh có 2 vùng sáng nổi bật. Như hình có nhiều bên dưới chẳng hạn, khi chưa lọc nhiễu thì chỉ có 1 đỉnh, sau khi lọc nhiễu có tới 2 đỉnh. Như vậy ngưỡng tốt nhất là giá trị trung bình của 2 đỉnh đó. Thuật toán này chỉ hiệu quả với ảnh có 2 đỉnh, do đó phải sử dụng hàm GaussianBlur để có được kết quả mong muốn.

```
double cvThreshold(Mat src, Mat dst, double thresh, double
maxValue, CV_THRESH_BINARY | CV_THRESH_OTSU);
```



Giải thích 1 chút tại sao ảnh sau khi GaussianBlur lại có 2 đỉnh:

- Ảnh gốc có nhiễu, các giá trị của pixel chênh lệch không nhiều nên histogram có dạng đồi thoải thoải
- Ảnh sau khi lọc nhiễu có 2 đỉnh, đỉnh cao là những pixel sáng nhất nằm trong hình nhật nhỏ. Đỉnh thấp là những pixel sáng hơn nằm bên ngoài.

CHƯƠNG 5: TRÍCH ĐẶC TRUNG VÀ ĐỐI SÁNH ẢNH

5.1. Các đặc trưng cơ bản

Trích chọn đặc trưng ảnh là tìm ra điểm đặc trưng của đối tượng so với đối tượng khác tùy theo mục đích nhận dạng trong quá trình xử lý ảnh. Ví dụ như ta muốn phân biệt đâu là ảnh cà chua chín thì dựa vào màu sắc chẳng hạn. Có thể dựa vào một số đặc điểm sau:

- **Đặc điểm không gian:** Phân bố mức xám, phân bố xác suất, biên độ, điểm uốn....
- **Đặc điểm biến đổi:** Các đặc điểm loại này được trích chọn bằng việc thực hiện lọc vùng (zonal filtering). Các bộ vùng được gọi là “mặt nạ đặc điểm” (feature mask) thường là các khe hẹp với hình dạng khác nhau (chữ nhật, tam giác, cung tròn v. v...)
- **Đặc điểm biên và đường biên:** Đặc trưng cho đường biên của đối tượng và do vậy rất hữu ích trong việc trích chọn các thuộc tính bất biến được dùng khi nhận dạng đối tượng. Các đặc điểm này có thể được trích chọn nhờ toán tử gradient, toán tử Laplace, toán tử “chéo không” (zero crossing) v. v... Việc trích chọn hiệu quả các đặc điểm giúp cho việc nhận dạng các đối tượng ảnh chính xác, với tốc độ tính toán cao và dung lượng nhớ lưu trữ giảm xuống.

Các đặc trưng cơ bản của ảnh bao gồm:

- Đặc trưng về màu sắc
- Đặc trưng về kết cấu
- Đặc trưng về hình dáng
- Đặc trưng cục bộ SIFT

5.2. Đặc trưng màu sắc

5.2.1. Mô tả các đặc trưng về màu sắc

Tìm kiếm ảnh theo lược đồ màu là phương pháp phổ biến và được sử dụng nhiều nhất trong các hệ thống tìm kiếm ảnh theo nội dung. Đây là phương pháp đơn giản, tốc độ tìm kiếm tương đối nhanh tuy nhiên kết quả tìm kiếm có độ chính xác không cao. Đây có thể xem là bước lọc đầu tiên cho những bước tìm kiếm sau. Một số lược đồ màu được sử dụng như: lược đồ màu RGB, lược đồ màu HSI, lược đồ HSI cải tiến.

Trong đó, lược đồ màu RGB được sử dụng phổ biến nhất[18][20].

- Lược đồ màu RGB:

Đối với ảnh 256 màu, lược đồ màu của ảnh tương đương với lược đồ màu của ảnh xám. Đối với ảnh 24 bit màu, lược đồ miêu tả khả năng kết nối về cường độ của ba kênh màu R, G, B. Lược đồ màu này được định nghĩa như sau:

$$h_{R,G,B}[r, g, b] = N * Prob\{R = r, G = g, B = b\}$$

Trong đó N là số lượng điểm có trong ảnh.

Lược đồ màu này được tính bằng cách rời rạc hóa từng màu trong ảnh, sau đó đếm số điểm ảnh của mỗi màu. Khi mà số lượng màu là có hạn, để thuận tiện hơn, người ta thường chuyển đổi ba kênh màu thành một biến giá trị duy nhất. Một cách khác để tính lược đồ màu của ảnh RGB là ta phân ra làm 3 lược đồ riêng biệt $h_R[], h_G[], h_B[]$. Khi đó, mỗi lược đồ được tính bằng cách đếm kênh màu tương ứng trong mỗi điểm ảnh.

5.2. 2. Độ đo tương đồng về màu sắc

Một số độ đo tương đồng được sử dụng như: Độ đo khoảng cách Öclit, độ đo Jensen-Shannon divergence (JSD).

Gọi $h(I)$ và $h(M)$ tương ứng là 2 lược đồ màu của hai ảnh I và ảnh M. Khi đó các loại độ đo màu được định nghĩa là một số nguyên (hoặc số thực) theo các loại độ đo tương ứng như sau:

- Khoảng cách Euclid:

Đây là khoảng cách Euclid thông thường giữa các K bin:

$$Intersection(h(I), h(M)) = \sum_{j=1}^K \sqrt{(h(I) - h(M))^2}$$

Hoặc:

$$Intersection(h(I), h(M)) = \sum_{j=1}^K |h(I) - h(M)|$$

- Độ đo Jensen-Shannon divergence (JSD) :

Độ đo Jensen-Shannon divergence sử dụng lược đồ màu RGB để tính toán độ tương đồng về màu sắc giữa 2 ảnh :

$$d_{JSD}(H, H') = \sum_{m=1}^M H_m \log \frac{2H_m}{H_m + H'_m} + H'_m \log \frac{2H'_m}{H'_m + H_m}$$

Trong đó : H và H' là 2 biểu đồ màu được so sánh, H_m là bin thứ m của biểu đồ H.

5.3. Đặc trưng kết cấu

5.3.1. Mô tả các đặc trưng kết cấu

Hiện tại, vẫn chưa có một định nghĩa chính thức cụ thể về kết cấu. Kết cấu là một đối tượng dùng để phân hoạch ảnh ra thành những vùng quan tâm để phân lớp những vùng đó.

Kết cấu cung cấp thông tin về sự sắp xếp về mặt không gian của màu sắc và cường độ một ảnh. Kết cấu được đặc trưng bởi sự phân bố không gian của những mức cường độ trong một khu vực láng giềng với nhau. Kết cấu gồm các kết cấu gốc hay nhiều kết cấu gộp lại đôi khi gọi là texel.

Một số phương pháp dùng để trích xuất các đặc trưng kết cấu như.

- Kim tự tháp "có thể lái được" (the steerable pyramid)
- Biến đổi đường viền (the cotourlet transform)
- Biến đổi sóng Gabor (The Gabor Wavelet transform)
- Biểu diễn ma trận đồng hiện (co-occurrence matrix)
- Hệ thống bộ lọc định hướng phức tạp (The complex directional filter bank)

5.3.2. Độ đo tương đồng cho kết cấu ảnh

Để đo độ tương đồng theo kết cấu giữa các ảnh, người ta thường sử dụng độ đo Oclit. Kết cấu được trích xuất từ các bức ảnh sẽ được biểu diễn thành các vector nhiều chiều và khoảng cách Oclit được dùng để đo độ tương đồng giữa các đặc trưng của ảnh truy vấn với đặc trưng của ảnh trong cơ sở dữ liệu.

5. 4. Đặc trưng hình dạng

5.4.1. Mô tả các đặc trưng hình dạng

Màu sắc và kết cấu là những thuộc tính có khái niệm toàn cục trong một ảnh. Trong khi đó, hình dạng không phải là một thuộc tính của ảnh. Nói tới hình dạng không phải là nhắc đến hình dạng của một ảnh. Thay vì vậy, hình dạng có khuynh hướng chỉ đến một khu vực đặc biệt trong ảnh, hay hình dạng chỉ là biên của một đối tượng nào đó trong ảnh.

Trong tìm kiếm ảnh theo nội dung, hình dạng là một cấp cao hơn so với màu sắc và kết cấu. Nó đòi hỏi sự phân biệt giữa các vùng để tiến hành xử lý về độ đo của hình dạng. Các hệ thống tìm kiếm ảnh theo nội dung thường khai thác hai nhóm biểu diễn hình dạng sau :

- Biểu diễn hình dạng theo đường biên (cotour-based descriptor) : Biểu diễn các đường biên bao bên ngoài
- Biểu diễn theo vùng (region-based descriptor): Biểu diễn một vùng toàn vẹn

5.4.2. Độ đo tương đồng cho hình dạng

Độ đo về hình dạng rất nhiều trong phạm vi lý thuyết của bộ môn xử lý ảnh. Chúng trải rộng từ những độ đo toàn cục dạng thô với sự trợ giúp của việc nhận dạng đối tượng, cho tới những độ đo chi tiết tự động tìm kiếm những hình dạng đặc biệt. Lược đồ hình dạng là một ví dụ của độ đo đơn giản. Kỹ thuật dùng đường biên hiệu quả hơn phương pháp trước, chúng tìm kiếm những hình dạng đối tượng gần giống với đường biên nhất. Phương pháp vẽ phác họa là phương pháp có nhiều đặc trưng rõ ràng hơn, không chỉ tìm kiếm những đường biên đối tượng đơn, mà còn đối với tập những đối tượng đã được phân đoạn trong một ảnh mà người dùng vẽ hay cung cấp.

5.5. Đặc trưng Haar-like

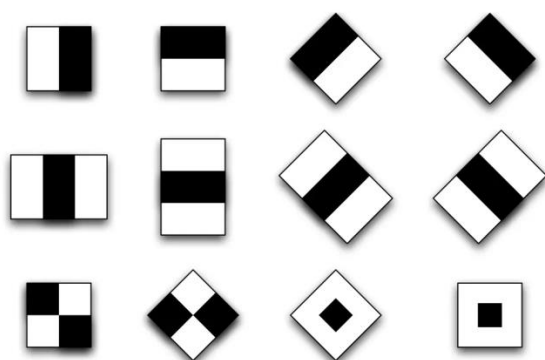
5.5.1. Giới thiệu

Một kỹ thuật trích chọn đặc trưng dựa trên phân bố mức xám là phép biến đổi Haar-Like do Viola và Jones công bố. Đặc trưng HaarLike [11] là một loại đặc trưng thường được dùng cho bài toán nhận dạng trên ảnh. Đặc trưng Haar-Like dựa trên ý tưởng tính độ chênh lệch giữa các giá trị mức xám của các điểm ảnh trong các vùng kề nhau trong ảnh xám, mỗi đặc trưng là sự kết hợp của các hình chữ nhật “trắng” hay “đen” theo một trật tự, kích thước nào đó.

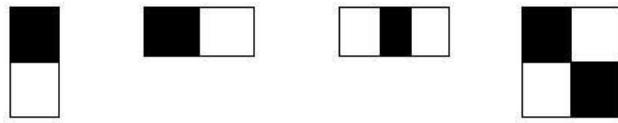
Lợi ích của đặc trưng Haar-Like là nó diễn đạt được tri thức về các đối tượng trong ảnh vì nó biểu diễn mối liên hệ giữa các bộ phận của đối tượng, điều mà bản thân từng điểm ảnh không diễn đạt được. Giá trị của đặc trưng Haar-Like là sự chênh lệch giữa tổng giá trị các điểm ảnh của các vùng đen và các vùng trắng.

5.5.2. Các đặc trưng Haar-Like

Các đặc trưng Haar-Like là những hình chữ nhật được phân thành các vùng khác nhau như hình:

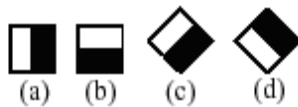


Đặc trưng do Viola và Jones công bố gồm 4 đặc trưng cơ bản để xác định khuôn mặt người. Mỗi đặc trưng Haar-Like là sự kết hợp của hai hay ba hình chữ nhật trắng hay đen như trong hình sau:

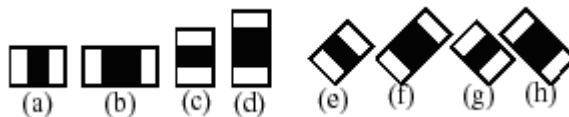


Để sử dụng các đặc trưng này vào việc xác định khuôn mặt người, 4 đặc trưng Haar-Like cơ bản được mở rộng ra và được chia làm 3 tập đặc trưng như sau:

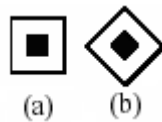
- Đặc trưng cạnh(edge feature)



- Đặc trưng đường(line feature)



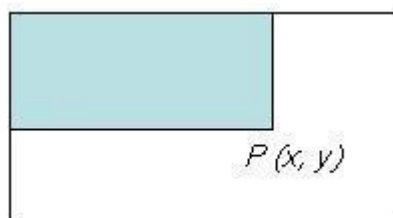
- Đặc trưng xung quanh tâm(center-surround features)



Dùng các đặc trưng trên, ta có thể tính được các giá trị của đặc trưng Haar-Like là sự chênh lệch giữa tổng của các pixel của vùng đen và vùng trắng như trong công thức sau:

$F(x) = \text{Tổng vùng đen (các mức xám của pixel)} - \text{Tổng vùng trắng (các mức xám của pixel)}$

Viola và Joines đưa ra một khái niệm gọi là Integral Image, là một mảng 2 chiều với kích thước bằng với kích thước của ảnh cần tính đặc trưng Haar-Like, với mỗi phần tử của mảng này được tính bằng cách tính tổng của điểm ảnh phía trên (dòng-1) và bên trái (cột-1) của nó.



Công thức tính Integral Image:

$$P(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

Sau khi tính được Integral Image, việc tính tổng các giá trị mức xám của một vùng bất kỳ nào đó trên ảnh thực hiện rất đơn giản theo cách sau:

Giả sử ta cần tính tổng giá trị mức xám của vùng D như hình dưới, ta có thể tính được như sau:

$$D = A + B + C + D - (A+B) - (A+C) + A$$

Với $A + B + C + D$ chính là giá trị tại điểm P4 trên Integral Image, tương tự như vậy $A+B$ là giá trị tại điểm P2, $A+C$ là giá trị tại điểm P3, và A là giá trị tại điểm P1. Vậy ta có thể viết lại biểu thức tính D ở trên như sau:

$$D = \underbrace{(x_4, y_4)}_{A+B+C+D} - \underbrace{(x_2, y_2)}_{(A+B)} - \underbrace{(x_3, y_3)}_{(A+C)} + \underbrace{(x_1, y_1)}_A$$

5.6. Đặc trưng cục bộ SIFT

5.6.1. Phát hiện đặc trưng cục bộ bất biến SIFT

Trong phương pháp trích rút các đặc trưng cục bộ bất biến SIFT của ảnh, các đặc trưng này bất biến với việc thay đổi tỉ lệ ảnh, quay ảnh, đôi khi là thay đổi điểm nhìn và thêm nhiễu ảnh hay thay đổi cường độ chiếu sáng của ảnh. Phương pháp được lựa chọn có tên là Scale-Invariant Feature Transform (SIFT) và đặc trưng trích rút được gọi là đặc trưng SIFT (SIFT Feature). Các đặc trưng SIFT này được trích rút ra từ các điểm hấp dẫn cục bộ (Local Interest Point)

- **Điểm hấp dẫn (Interest Point (Keypoint)):** Là vị trí (điểm ảnh) "hấp dẫn" trên ảnh. "Hấp dẫn" ở đây có nghĩa là điểm đó có thể có các đặc trưng bất biến với việc quay ảnh, co giãn ảnh hay thay đổi cường độ chiếu sáng của ảnh.

Phương pháp trích rút các đặc trưng bất biến SIFT được tiếp cận theo phương pháp thác lọc, theo đó phương pháp được thực hiện lần lượt theo các bước sau:

- **Phát hiện các điểm cực trị Scale-Space** (Scale-Space extrema detection): Bước đầu tiên này tiến hành tìm kiếm các điểm hấp dẫn trên tất cả các tỉ lệ và vị trí của ảnh. Nó sử dụng hàm different-of-Gaussian để xác định tất cả các điểm hấp dẫn tiềm năng mà bất biến với quy mô và hướng của ảnh.
- **Định vị các điểm hấp dẫn** (keypoint localization): Một hàm kiểm tra sẽ được đưa ra để quyết định xem các điểm hấp dẫn tiềm năng có được lựa chọn hay không?

- **Xác định hướng cho các điểm hấp dẫn** (Orientation assignment): Xác định hướng cho các điểm hấp dẫn được chọn
- **Mô tả các điểm hấp dẫn** (Keypoint descriptor): Các điểm hấp dẫn sau khi được xác định hướng sẽ được mô tả dưới dạng các vector đặc trưng nhiều chiều.
- **Phát hiện điểm cực trị Scale-space**

Các điểm hấp dẫn với đặc trưng SIFT tương thích với các cực trị địa phương của bộ lọc difference-of-Gaussian (DoG) ở các tỉ lệ khác nhau. Định nghĩa không gian tỉ lệ của một hình ảnh là hàm $(x, y, k) \rightarrow L(x, y, k\sigma)$ được mô tả như sau:

$$L(x, y, k\sigma) = G(x, y, k\sigma) \otimes I(x, y) \quad (5)$$

Trong đó: $G(x, y, k\sigma)$: Tỉ lệ Gaussian (variable scale Gaussian)

$I(x, y)$: Ảnh đầu vào

Ta có:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad (6)$$

Để phát hiện được các điểm hấp dẫn, ta đi tìm các cực trị của hàm DoG được định nghĩa:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \quad (7)$$

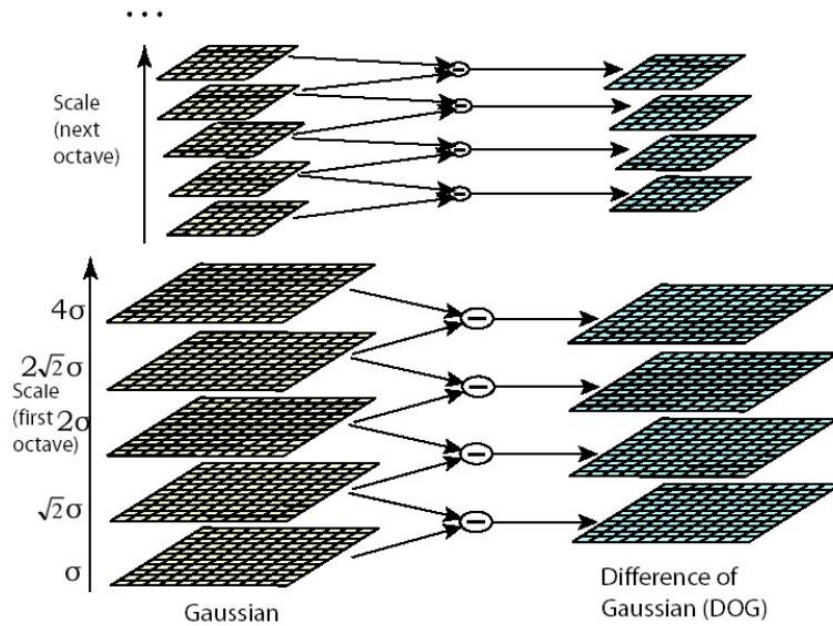
Giá trị hàm DoG được tính xấp xỉ dựa vào giá trị scale-normalized Laplacian of Gaussian ($\sigma^2 \nabla^2 G$) thông qua các phương trình (5)(6)(7)

$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G \quad (8)$$

$$\sigma \nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \quad (9)$$

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 \nabla^2 G \quad (10)$$

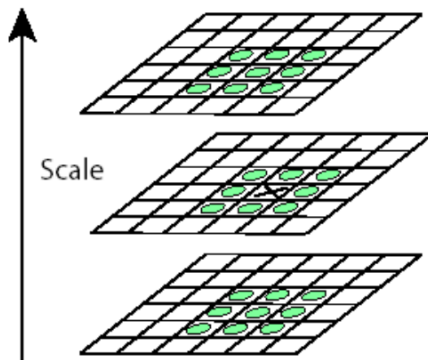
Như vậy, bước đầu tiên của giải thuật SIFT phát hiện các điểm hấp dẫn với bộ lọc Gaussian ở các tỉ lệ khác nhau và các ảnh DoG từ sự khác nhau của các ảnh kề mờ.



Hình 8. Biểu đồ mô phỏng việc tính toán các DoG ảnh từ các ảnh mờ

Các ảnh cuộn được nhóm thành các octave (mỗi octave tương ứng với giá trị gấp đôi của σ). Giá trị của k được chọn sao cho số lượng ảnh mờ (blurred images) cho mỗi octave là cố định. Điều này đảm bảo cho số lượng các ảnh DoG cho mỗi octave không thay đổi.

Các điểm hấp dẫn được xác định là các cực đại hoặc cực tiểu của các ảnh DoG qua các tỉ lệ. Mỗi điểm ảnh trong DoG được so sánh với 8 điểm ảnh láng giềng của nó ở cùng tỉ lệ đó và 9 láng giềng kề ở các tỉ lệ ngay trước và sau nó. Nếu điểm ảnh đó đạt giá trị cực tiểu hoặc cực đại thì sẽ được chọn làm các điểm hấp dẫn ứng viên.



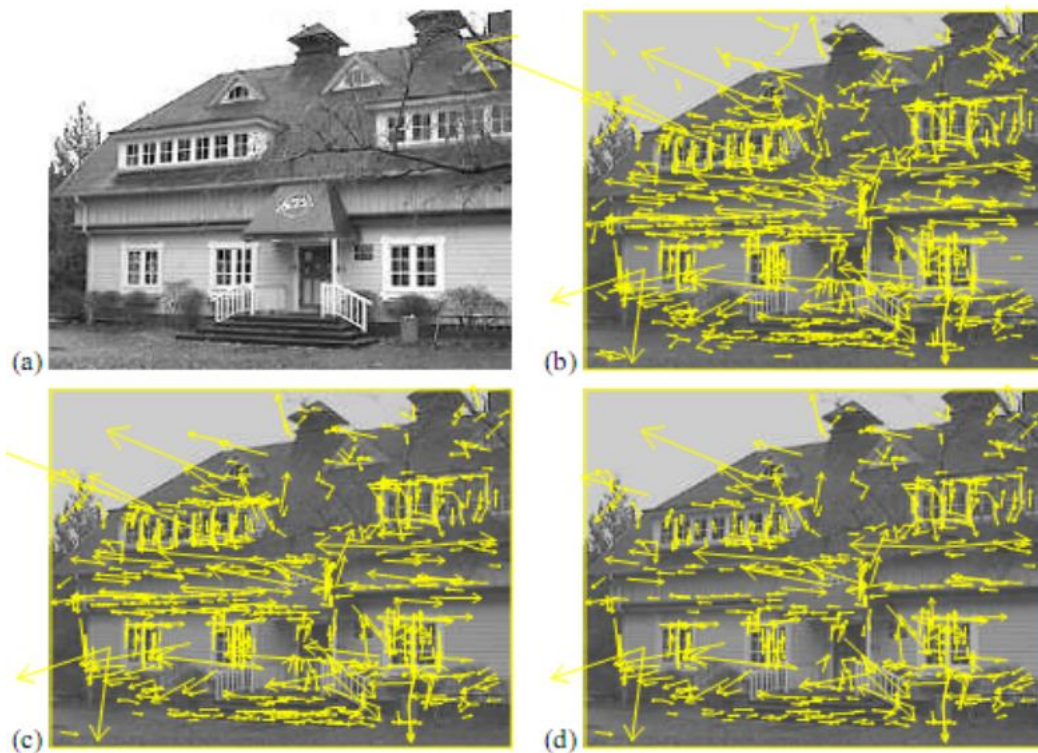
Hình 9. Mỗi điểm ảnh được so sánh với 26 láng giềng của nó

- **Định vị điểm hấp dẫn**

Mỗi điểm hấp dẫn ứng viên sau khi được chọn sẽ được đánh giá xem có được giữ lại hay không:

- Loại bỏ các điểm hấp dẫn có độ tương phản thấp
- Một số điểm hấp dẫn dọc theo các cạnh không giữ được tính ổn định khi ảnh bị nhiễu cũng bị loại bỏ.

Các điểm hấp dẫn còn lại sẽ được xác định hướng.



Hình 10. Quá trình lựa chọn các điểm hấp dẫn

a. Ảnh gốc, b. Các điểm hấp dẫn được phát hiện, c. Ảnh sau khi loại bỏ các điểm hấp dẫn có độ tương phản thấp, d. Ảnh sau loại bỏ các điểm hấp dẫn dọc theo cạnh.

- **Xác định hướng cho điểm hấp dẫn:**

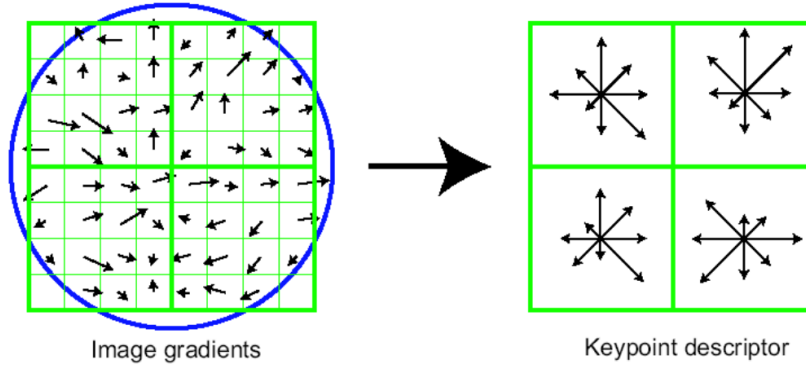
Để xác định hướng cho các điểm hấp dẫn, người ta tính toán biểu đồ hướng Gradient trong vùng láng giềng của điểm hấp dẫn. Độ lớn và hướng của các điểm hấp dẫn được xác định theo công thức:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \quad (11)$$

$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1)) / (L(x+1, y) - L(x-1, y))) \quad (12)$$

- **Biểu diễn vector cho điểm hấp dẫn**

Điểm hấp dẫn sau khi được xác định hướng sẽ được biểu diễn dưới dạng các vector 4x4x8=128 chiều.



Hình 11. Biểu diễn các vector đặc trưng

5.6.2. Độ đo tương đồng cho đặc trưng cục bộ bất biến

Một số độ đo tương đồng cho ảnh sử dụng đặc trưng SIFT như sau:

- Độ đo Cosin:

$$d(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|} \quad (13)$$

- Khoảng cách góc :

$$d(x, y) = \cos^{-1}(x \cdot y) \quad (14)$$

- Độ đo Euclide :

$$d(x, y) = \sqrt{\sum_{i=1}^n |x_i - y_i|^2} \quad (15)$$

- Độ đo Jensen-Shannon divergence :

$$d_{JSD}(H, H') = \sum_{m=1}^M H_m \log \frac{2H_m}{H_m + H'_m} + H'_m \log \frac{2H'_m}{H'_m + H_m} \quad (16)$$

Với H, H' là 2 biểu đồ biểu diễn các vector đặc trưng SIFT.

CHƯƠNG 6: NHẬN DẠNG

6.1. Nhận dạng đối tượng

Phát hiện đối tượng là để trả lời câu hỏi: “Đối tượng cần tìm có ở trong ảnh hay không?” và “Nếu có thì nằm ở vị trí nào?”

Trong các phương pháp phát hiện đối tượng hiện nay có Haar cascade là phương pháp phổ biến nhất. Quy trình bao gồm:

Bước 1: Dùng tool của Opencv cung cấp để rút trích đặc trưng của đối tượng (gọi là haar-like) ghi vào tập dữ liệu (*.xml). Đó là các đặc trưng đường chéo, đường ngang, đường dọc của đối tượng.

Bước 2: Viết chương trình để tìm đối tượng. Sau khi load các đặc trưng từ file xml đã huấn luyện, thuật toán của OpenCV sẽ lần lượt quét qua các vùng trên ảnh để tìm đối tượng. Đối tượng nào có đặc trưng giống với tập dữ liệu & thoả mãn các tham số thì xem như đó là đối tượng cần tìm.

6.1.1. Rút trích đặc trưng

Bước này còn được gọi là huấn luyện (training)

Bạn cần chuẩn bị 2 bộ ảnh, một bộ ảnh có chứa đối tượng gọi là ảnh dương. Bộ còn lại là những ảnh bất kỳ không chứa đối tượng gọi là ảnh âm. Số lượng ảnh âm & dương càng nhiều thì độ chính xác càng cao & thời gian huấn luyện càng lâu.

Khi training, chương trình sẽ rút ra đặc trưng rồi so sánh với các ảnh âm, nếu đặc trưng đó được tìm thấy trong ảnh âm thì đặc trưng đó không phù hợp.

Ví dụ: Để phát hiện biển số xe, bạn cần khoảng 2000 ảnh biển số & 1000 ảnh không chứa biển số. Ảnh không có biển số có thể là ảnh hoa lá cảnh, đồ vật bất kỳ...

Tiếp theo, chúng ta phải chỉ định rõ vị trí hình chữ nhật chứa đối tượng cần tìm và lưu vào file chỉ mục. Có 2 cách thực hiện:

- Crop ảnh chỉ lấy vùng chữ nhật chứa đối tượng: file chỉ mục chỉ chứa tên file ảnh
- Ghi lại thông số x y width height của vùng chữ nhật vào file: file chỉ mục gồm tên file ảnh, số vùng chữ nhật trong ảnh và các tọa độ x y width height

Khi đã có file chỉ mục, các bạn sẽ làm 2 bước để tạo tập dữ liệu (*.xml), thuật toán của 2 bước này được viết sẵn:

- Bước 1.1: dùng **opencv_createsamples.exe** tạo file vecto chứa ảnh thumbnail của vùng hình chữ nhật. Bước này nhanh, 2000 ảnh mất chưa tới 1 phút.
- Bước 1.2: dùng **opencv_traincascade.exe** lấy ảnh trong file vecto đặt ngẫu nhiên vào các ảnh âm để rút trích các đặc trưng. Bước này chậm, 2000 ảnh mất từ 1 ngày đến 1 tuần tùy theo độ chính xác các bạn yêu cầu.

Hoàn thành 2 bước này chúng ta sẽ có file xml

• Bước 1.1 Tạo file vecto

Đầu tiên các bạn tìm chương trình **opencv_createsamples.exe** trong lib Opencv theo đường dẫn **opencv300\build\bin\Release** và truyền các tham số sau:

```

C:\WINDOWS\system32\cmd.exe
E:\lib\opencv300\build\bin\Release>opencv_createsamples
Usage: opencv_createsamples
[-info <collection_file_name>]
[-img <image_file_name>]
[-vec <vec_file_name>]
[-bg <background_file_name>]
[-num <number_of_samples = 1000>]
[-bgcolor <background_color = 0>]
[-inv] [-randinv] [-bgthresh <background_color_threshold = 80>]
[-maxidev <max_intensity_deviation = 40>]
[-maxxangle <max_x_rotation_angle = 1.100000>]
[-maxyangle <max_y_rotation_angle = 1.100000>]
[-maxzangle <max_z_rotation_angle = 0.500000>]
[-show [<scale = 4.000000>]]
[-w <sample_width = 24>]
[-h <sample_height = 24>]
E:\lib\opencv300\build\bin\Release>_

```

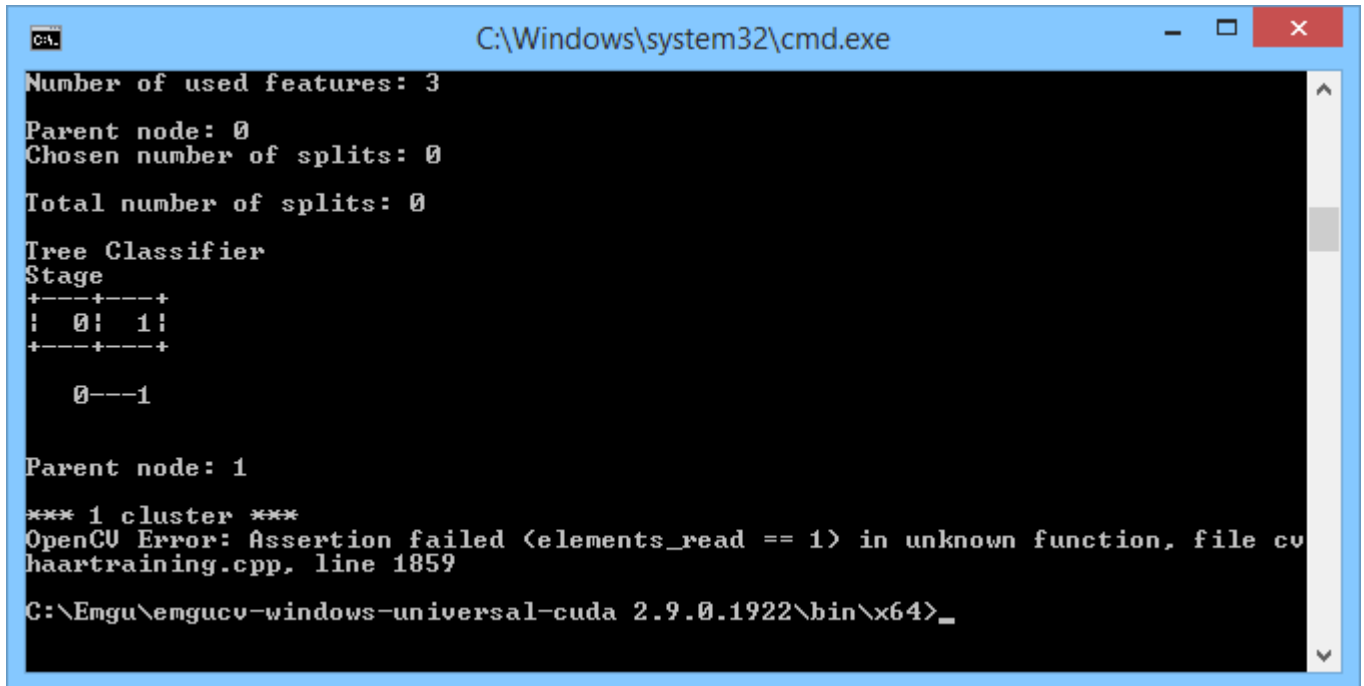
- **info** file chỉ mục mà bạn đã tạo
- **img** khi bạn chỉ có 1 ảnh dương (như là logo cty chẳng hạn) thì bỏ qua tham số **info** và truyền tên file ảnh vào đây
- **vec** file output cho bước tiếp theo
- **bg** file ảnh âm khi bạn chỉ có 1 ảnh dương
- **num** số lượng file ảnh dương
- **bgcolor** khi bạn không chỉ định ảnh âm thì có thể chỉ định 1 màu để làm ảnh âm
- **inv -randinv -bgthresh -maxidev -maxxangle -maxyangle -maxzangle** các tham số để tạo thêm các ảnh ngẫu nhiên bằng cách xoay trong không gian 3 chiều
- **show** hiện ảnh trong quá trình tạo mẫu
- **w** chiều rộng của mẫu được tạo
- **h** chiều cao của mẫu được tạo

Ví dụ: `opencv_createsamples.exe -vec D:\bienso.vec -inf`

`D:\plate_image\positive\location.txt -num 2000 -w 40 -h 30`

OpenCV sẽ tạo file `D:\bienso.vec` có chứa 2000 ảnh thumbnail, mỗi ảnh có kích thước 40 x 30 pixels.

- **Bước 1.2 huấn luyện**



```
C:\Windows\system32\cmd.exe

Number of used features: 3
Parent node: 0
Chosen number of splits: 0
Total number of splits: 0

Tree Classifier
Stage
+---+---+
| 0! 1!
+---+---+

      0---1

Parent node: 1

*** 1 cluster ***
OpenCV Error: Assertion failed (elements_read == 1) in unknown function, file cv
haartraining.cpp, line 1859

C:\Emgu\emgucv-windows-universal-cuda 2.9.0.1922\bin\x64>_
```

```

C:\WINDOWS\system32\cmd.exe

E:\lib\opencv300\build\bin\Release>opencv_traincascade.exe
Usage: opencv_traincascade.exe
  -data <cascade_dir_name>
  -vec <vec_file_name>
  -bg <background_file_name>
  [-numPos <number_of_positive_samples = 2000>]
  [-numNeg <number_of_negative_samples = 1000>]
  [-numStages <number_of_stages = 20>]
  [-precalcValBufSize <precalculated_vals_buffer_size_in_Mb = 1024>]
  [-precalcIdxBufSize <precalculated_idx_buffer_size_in_Mb = 1024>]
  [-baseFormatSave]
  [-numThreads <max_number_of_threads = 5>]
  [-acceptanceRatioBreakValue <value> = -1]
--cascadeParams--
  [-stageType <BOOST(default)>]
  [-featureType <{HAAR(default), LBP, HOG}>]
  [-w <sampleWidth = 24>]
  [-h <sampleHeight = 24>]
--boostParams--
  [-bt <{DAB, RAB, LB, GAB(default)}>]
  [-minHitRate <min_hit_rate> = 0.995]
  [-maxFalseAlarmRate <max_false_alarm_rate = 0.5>]
  [-weightTrimRate <weight_trim_rate = 0.95>]
  [-maxDepth <max_depth_of_weak_tree = 1>]
  [-maxWeakCount <max_weak_tree_count = 100>]
--haarFeatureParams--
  [-mode <BASIC(default) | CORE | ALL
--lbpFeatureParams--
--HOGFeatureParams--

E:\lib\opencv300\build\bin\Release>_

```

Chúng ta lấy file **opencv_traincascade.exe** truyền các tham số sau (do nhiều tham số nên mình chỉ ghi các tham số quan trọng):

- **data** Thư mục rỗng dành để chứa các step và các file tạm trong quá trình huấn luyện, file xml thu được cũng nằm trong này
- **vec** file vec bạn tạo ở bước 1.1
- **bg** file text chứa danh sách các file ảnh âm
- **numPos** số lượng ảnh dương
- **numNeg** số lượng ảnh âm
- **numStage** số lượng bước huấn luyện bạn muốn, càng lớn (14 -> 20) thì thời gian huấn luyện càng lâu và kết quả càng chính xác
- **w** chiều rộng của mẫu được tạo (ở bước 1.1 bao nhiêu thì ở đây cũng vậy)
- **h** chiều cao của mẫu được tạo (ở bước 1.1 bao nhiêu thì ở đây cũng vậy)

- **minHitRate** tỉ lệ chính xác của ảnh dương.
- **maxFalseAlarmRate** tỉ lệ cho phép sai số tối đa của ảnh âm.

Ví dụ: `opencv_traincascade.exe -data D:\output -vec D:\bienso.vec -bg D:\plate_image\negative\list.txt -numPos 2000 -numNeg 1000 -numStage 20 -minHitRate 0.995 -maxFalseAlarmRate 0.5 -w 40 -h 30`

Chương trình sẽ tạo file `D:\output\cascade.xml` chứa các đặc trưng của biển số xe. `minHitRate=0.995` nghĩa là phải đúng 995/1000 ảnh. `maxFalseAlarmRate=0.5` nghĩa là cứ mỗi 1000 ảnh chỉ được detect nhầm tối đa 500 ảnh.

6.1.2. Sử dụng tập dữ liệu để phát hiện đối tượng

Trước khi thực hiện thuật toán phát hiện đối tượng, ảnh phải chuyển sang ảnh xám (gray). Đồng thời cũng cân bằng sáng (equalize hist) để ảnh được đồng nhất về độ tương phản, không bị ảnh hưởng bởi ánh sáng môi trường.

OpenCV sẽ quét qua toàn bộ ảnh để so sánh với tập dữ liệu. Lần đầu tiên sẽ lấy hình chữ nhật kích thước bằng `max-size` được chỉ định (hoặc bằng kích thước ảnh). Sau đó thu nhỏ lại theo tỉ lệ được chỉ định rồi lại so sánh với tập dữ liệu, cứ như thế cho đến khi vùng quét bằng kích thước `min-size` thì dừng.



Để sử dụng, bạn cần load data từ file xml, sau đó dùng hàm **`detectMultiScale(const Mat& image, vector& objects, double scaleFactor=1.1, int minNeighbors=3, int flags=0, Size minSize=Size(), Size maxSize=Size())`** để tìm đối tượng trong ảnh. Các tham số truyền vào bao gồm:

- **image**: ảnh input
- **objects**: kết quả trả về là các hình chữ nhật
- **scaleFactor** tỉ lệ vùng chữ nhật được thu nhỏ sau mỗi lần quét (default là 1.1)
- **minNeighbors** số khung hình chữ nhật trùng nhau tối thiểu để được xem là 1 đối tượng (default là 3)

- **flags** phương thức quét
- **minSize** và **maxSize**: kích cỡ tối thiểu, kích cỡ tối đa.

Hình bên dưới là test detect nhằm biển số xe hơi. Mặc dù đó không phải là biển số nhưng có đủ đặc trưng khi huấn luyện thì cũng được xem là biển số.



6.1.3. Ví dụ nhận diện biển số xe máy sử dụng OpenCV

- **Các công cụ cần thiết**
 1. OpenCV 2.4.9 Đây là phiên bản ổn định (stable), các bạn có thể dùng phiên bản cao hơn.
 2. Tập dữ liệu dương: Sử dụng bộ ảnh của công ty Green Parking. Trong file zip chứa 1749 ảnh
 3. Tập dữ liệu âm: bao gồm 3000 hình ảnh bất kỳ không chứa biển số xe
 4. Object locator: dùng để khoanh vùng đối tượng
 5. Cascade analytics: dùng để đánh giá độ chính xác sau khi huấn luyện xong

- **Các bước thực hiện như sau:**

Bước 1: Khoanh vùng đối tượng

Chúng ta sử dụng tool Object locator vẽ hình chữ nhật bao lấy đối tượng, sau khi làm xong chúng ta sẽ được file location.txt nằm trong thư mục chứa ảnh.

Bước 2: Tạo tập ảnh samples

Dùng lệnh sau để tạo tập ảnh:

```
opencv_createsamples.exe -info E:\GreenParking\location.txt -vec GreenParking.vec -w 30 -h 20 -num 1749
```

Ý nghĩa lệnh: bạn tạo file vecto tên là GreenParking.vec chứa 1749 ảnh crop biên số, mỗi ảnh có kích thước 30×20 **pixels**.

Bước 3: Huấn luyện

Tạo folder để chứa dữ liệu các bước huấn luyện, ví dụ folder E:\train

Dùng lệnh sau để huấn luyện

```
opencv_traincascade.exe -data E:\train -vec GreenParking.vec -bg E:\negatives\bg.txt -numPos 1700 -numNeg 3000 -w 30 -h 20 -featureType LBP
```

Ý nghĩa lệnh: huấn luyện để tạo file E:\train\cascade.xml, với 1700 ảnh dương và 3000 ảnh âm, với kích thước bằng kích thước samples đã được tạo, sử dụng LBP feature vì kết quả chính xác tương đương Haar feature nhưng lại nhanh hơn rất nhiều, phù hợp cho việc thử nghiệm.

Khi huấn luyện chương trình sẽ hiển thị như thế này

```

Administrator: C:\Windows\system32\cmd.exe - opencv_traincascade -data x86v12 -vec GreenParking_expand.vec -bg E:\HaarCascade\tutorial-haartraining\d...
+-----+
| 8 | 1 | 0.492 |
+-----+
END>
Training until now has taken 0 days 0 hours 37 minutes 29 seconds.

===== TRAINING 15-stage =====
<BEGIN
POS count : consumed 3000 : 3000
NEG count : acceptanceRatio 3000 : 4.82016e-006
Precalculation time: 2.459
+-----+
| N | HR | FA |
+-----+
| 1 | 1 | 1 |
+-----+
| 2 | 1 | 1 |
+-----+
| 3 | 1 | 1 |
+-----+
| 4 | 1 | 1 |
+-----+
| 5 | 1 | 0.873667 |
+-----+
| 6 | 1 | 0.873667 |
+-----+
| 7 | 1 | 0.675667 |
+-----+
| 8 | 1 | 0.408 |
+-----+
END>
Training until now has taken 0 days 1 hours 10 minutes 27 seconds.

===== TRAINING 16-stage =====
<BEGIN
POS count : consumed 3000 : 3000
NEG current samples: 2372

```

Chú ý:

- Khi tạo mẫu thì 1749 nhưng khi huấn luyện chỉ dùng 1700, điều đó giúp cho chương trình chạy không bị crash. Số lượng ảnh phải giảm đi tùy vào bộ ảnh.
- Với câu lệnh như vậy thì thời gian huấn luyện trên máy CPU Intel core i5 2500 mất hơn 3 giờ

- **Kiểm tra độ chính xác bằng tool**

Sau khi huấn luyện xong bạn sẽ được file E:\train\cascade.xml, dùng chương trình Cascade analytics để đánh giá độ chính xác.

6.2. Nhận dạng mặt người

6.2.1. Nhận diện khuôn mặt: Bước 1: face detection

Phát hiện khuôn mặt (detect) là bước đầu tiên của bài toán nhận diện khuôn mặt (recognize). Về nguyên lý thì phát hiện khuôn mặt giống như phát hiện bất kỳ đối tượng nào trong ảnh. Do vấn đề này đã được nghiên cứu và ứng dụng quá nhiều nên chúng ta không nên huấn luyện lại mà nên sử dụng file xml đã được huấn luyện sẵn.

Các file xml khuôn mặt được cung cấp sẵn trong OpenCV được huấn luyện từ hình ảnh khuôn mặt người phương Tây, các file xml này cũng đã làm tốt việc phát hiện khuôn mặt người Việt Nam.

Chương trình này load data xml khuôn mặt, sau đó detect trên ảnh/video/camera. Viết bằng Visual C++ 11 sử dụng thư viện Opencv.

- **Lý thuyết**

Nhận diện khuôn mặt gồm 3 bước:

Bước 1. Tiền xử lý ảnh: trích xuất ảnh từ video, webcam. Sau đó biến đổi thành ảnh xám (gray)

Bước 2. Tìm khuôn mặt trong ảnh xem có khuôn mặt trong ảnh không? Nếu có thì có bao nhiêu khuôn mặt? Sau đó crop để qua bước 3

Bước 3. Nhận diện khuôn mặt và trả về kết quả. Có thể vẽ hình vuông bao lấy khuôn mặt hoặc vẽ text tên người

- **Giới thiệu về Cascade model**

Đặc trưng haar-like là tập hợp những đặc trưng đường ngang, đường dọc, đường chéo,... của một đối tượng. Các đặc trưng này được lưu trong 1 file xml, dựa vào file này mà có thể biết được ảnh có chứa đối tượng cần tìm hay không.

Ngoài HAAR ra thì có đặc trưng LBP và HOG cũng được áp dụng cho nhận diện hình ảnh.

OpenCV cung cấp sẵn các model như lbpcascade_frontalface.xml để phát hiện khuôn mặt. Các bạn tải file này về rồi dùng tool Cascade analytics để thử.

- **Các bước của chương trình**

Đầu tiên chương trình load các file cascade model được chỉ định file config.ini. Nếu load cascade thành công thì load file video cũng được chỉ định trong config.ini.

Sau đó tiến hành tách ra từng frame rồi detect khuôn mặt, nếu có khuôn mặt thì vẽ hình vuông lên trên khuôn mặt.

6.2.2. Nhận diện khuôn mặt – Bước 2: face recognition

Sau khi đã phát hiện được khuôn mặt chúng ta tiến hành nhận dạng (recognition).

Để nhận dạng ta sử dụng lib **face**, là lib được đóng góp bởi người dùng OpenCV. Trong lib **face** hỗ trợ 3 phương pháp nhận diện là Eigen, Fisher và LBPH. Các tham số đó tốt nhất là để trong file config để dễ dàng thay đổi mà không cần build lại.

- **Thuật toán**

Nhận diện khuôn mặt là học có giám sát, tức là chúng ta phải dán nhãn cho ảnh khuôn mặt. Từ đó chương trình sẽ tìm ra sự khác biệt của các khuôn mặt rồi lưu lại model. Với model đã học chúng ta dùng để nhận diện khuôn mặt.

Cách mình làm như sau:

- Thu thập khuôn mặt, crop rồi lưu vào folder, mỗi người là mỗi folder
- Khi chạy chương trình thì training ảnh khuôn mặt
- Với mỗi khung hình tiến hành tìm kiếm khuôn mặt
- Nếu tìm thấy khuôn mặt thì nhận diện rồi in kết quả

6.2.3. Nhận diện khuôn mặt – Bước 3: tối ưu hoá

Trong thực tế tình trạng nhận diện sai xảy ra khá nhiều, để khắc phục có một vài cách như sau:

1. Thay đổi file cascade và tối ưu các tham số
 2. Tìm đôi mắt trong khuôn mặt đã nhận được
 3. Kiểm tra màu da
- **Thay đổi file cascade và tối ưu các tham số**

Theo như lý thuyết thì độ chính xác phát hiện khuôn mặt phụ thuộc rất nhiều vào file cascade và các tham số. Những file cascade được cung cấp sẵn theo lib OpenCV để phát hiện khuôn mặt có sự khác biệt nhau. Mình thử với ảnh 80 ảnh chứa 77 khuôn mặt thì phát hiện được số lượng khuôn mặt như sau (các tham số của hàm detectMultiScale để default):

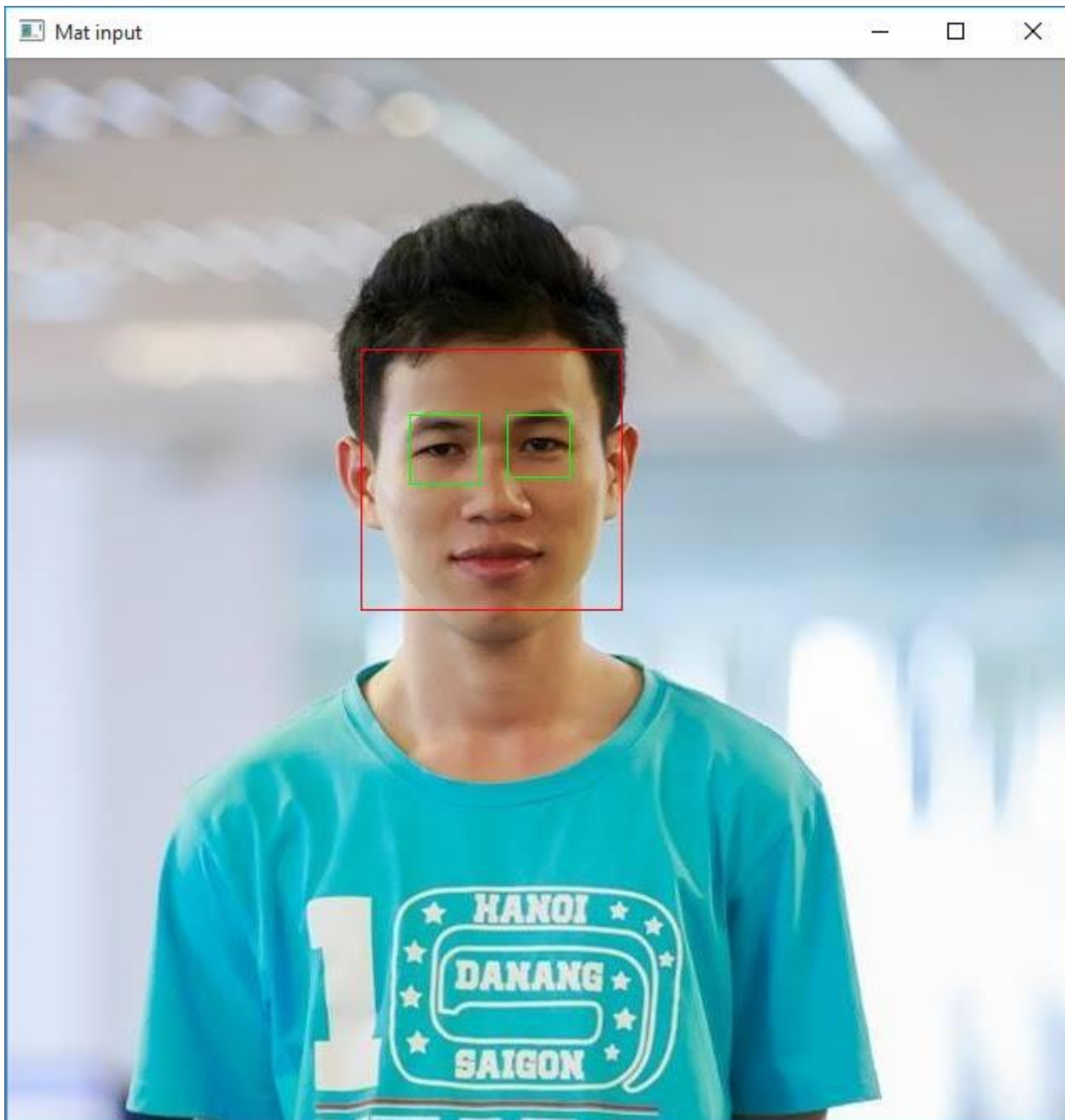
haarcascade_frontalface_alt.xml	86	khuôn	mặt/80	ảnh
haarcascade_frontalface_alt2.xml	95	khuôn	mặt/80	ảnh
haarcascade_frontalface_alt_tree.xml	74	khuôn	mặt/80	ảnh
haarcascade_frontalface_default.xml	155	khuôn	mặt/80	ảnh

Như vậy, có thể dùng **haarcascade_frontalface_alt.xml**

hoặc **haarcascade_frontalface_alt_tree.xml** để đạt kết quả tốt nhất. Tuy nhiên vẫn còn tùy thuộc tập dữ liệu và tham số truyền vào. Do đó cần test cẩn thận bộ dữ liệu sẽ sử dụng.

- **Tìm đôi mắt trong khuôn mặt đã nhận được**

Trong khuôn mặt đã phát hiện được có thể dùng cascade để tìm đôi mắt để tăng độ chính xác. Hơn nữa, biết được vị trí của đôi mắt có thể xoay khuôn mặt về đúng góc độ để dùng nhận diện khuôn mặt sẽ tốt hơn.



Trong thư viện OpenCV có đi kèm các file cascade sau có thể giúp detect được đôi mắt

[haarcascade_eye.xml](#)

[haarcascade_eye_tree_eyeglasses.xml](#)

- **Kiểm tra màu da**

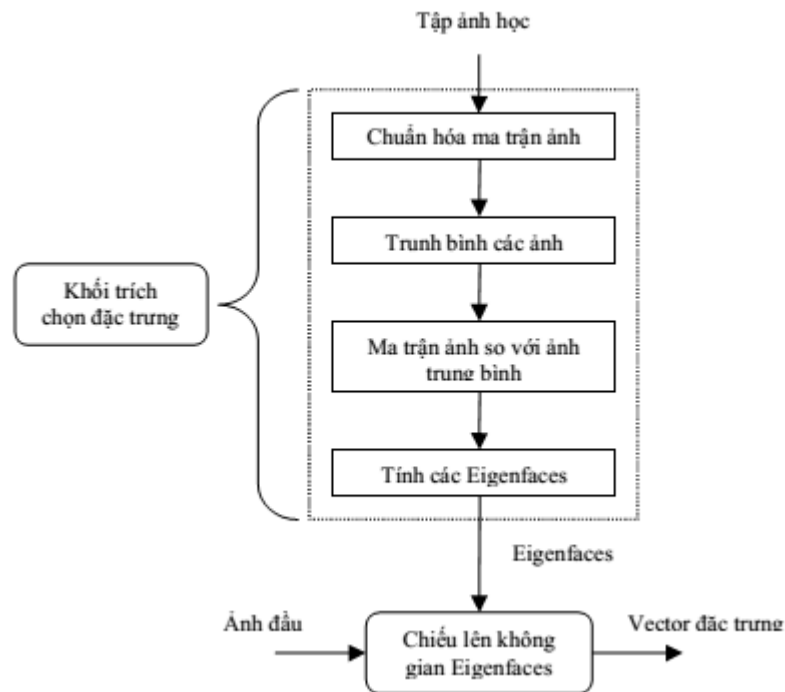
Có thể dùng màu HSV để kiểm tra đó có phải là da người không. Điểm yếu của cách này là chỉ nhận được màu da vàng, còn da quá trắng hoặc quá đen sẽ khó mà nhận biết:

1. Crop khuôn mặt
2. Chuyển ảnh khuôn mặt thành HSV rồi giới hạn trong khoảng (0, 48, 80) và (20, 255, 255)
3. Lấy giá trị trung bình của kênh màu H, nếu giá trị trung bình lớn hơn 100 thì đó là khuôn mặt



6.3. Eigenfaces

Eigen Faces là phương pháp áp dụng trực tiếp phép phân tích các thành phần chính PCA, nó đã được áp dụng rất nhiều vào biểu diễn, phát hiện và nhận dạng mặt. Ưu điểm của phương pháp này là biểu diễn được toàn bộ ảnh và có độ nén rất tốt (loại bỏ nhiễu và dư thừa).



Hình 12. Sơ đồ khối trích chọn đặc trưng sử dụng Eigen Faces.

6.3.1. Thuật toán PCA và ứng dụng trong nhận dạng mặt người.

- **Giới thiệu chung về thuật toán.**

Phân tích thành phần chính (*Principal Component Analysis*) gọi tắt là PCA là một thuật toán được sử dụng để tạo ra một ảnh mới từ ảnh ban đầu. Ảnh mới này có kích thước nhỏ hơn nhiều so với ảnh ban đầu nhưng vẫn mang những đặc trưng cơ bản nhất của ảnh cần nhận dạng.

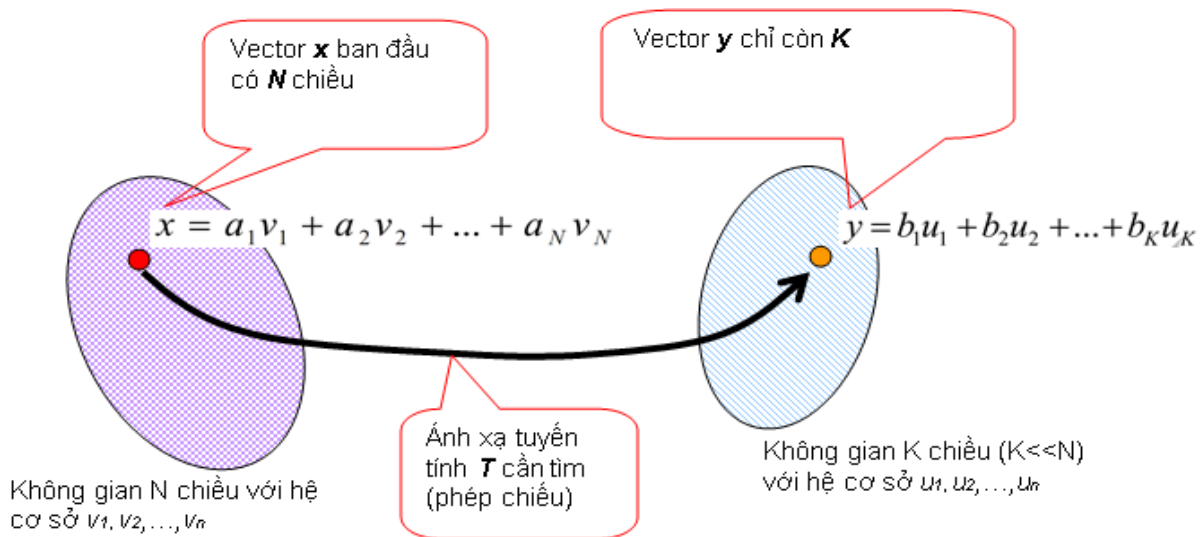
PCA không cần quan tâm đến việc tìm ra các đặc điểm cụ thể của thực thể cần nhận dạng và mối quan hệ giữa các đặc điểm đó. Tất cả các chi tiết đó đều được thể hiện ở ảnh mới được tạo ra từ PCA.

- **Ưu điểm của phương pháp PCA**

- Tìm được các đặc tính tiêu biểu của đối tượng cần nhận dạng mà không cần phải xác định các thành phần và mối quan hệ giữa các thành phần đó.
- Thuật toán có thể thực hiện tốt với các ảnh có độ phân giải cao, do PCA sẽ thu gộp ảnh thành một ảnh có kích thước nhỏ hơn.
- PCA có thể kết hợp với các phương pháp khác như mạng Neuron, Support Vector Machine... để mang lại hiệu quả nhận dạng cao hơn.

- **Nhược điểm của PCA:**

- PCA phân loại theo chiều lớn nhất của tập vector. Tuy nhiên, chiều phân bố lớn nhất không phải lúc nào cũng mang lại hiệu quả tốt nhất cho bài toán nhận dạng. Đây là nhược điểm cơ bản của PCA.
- PCA rất nhạy với nhiễu.
- **Ứng dụng trong bài toán nhận dạng mặt người**
 - Trong bài toán nhận dạng mặt người, PCA là thuật toán nhận dạng ảnh dựa trên những nét tổng thể của khuôn mặt, ta sẽ áp dụng thuật toán này để thực hiện công việc tìm một khuôn mặt giống với khuôn mặt cho trước với kích thước nhỏ hơn và chỉ mang những nét đặc trưng của khuôn mặt.
 - Đặc trưng PCA: Mục tiêu của phương pháp PCA là “giảm số chiều” của một tập vector sao cho vẫn đảm bảo được “tối đa thông tin quan trọng nhất”. Phương pháp PCA sẽ giữ lại K thuộc tính mới (Feature extraction) từ N các thuộc tính ban đầu (feature selection) ($K < N$).



- Mục tiêu của phương pháp PCA đó là tìm phép biến đổi tuyến tính T thỏa mãn:

$$y = T.x$$

sao cho trung bình bình phương lỗi (MSE) là nhỏ nhất (sai số nhỏ nhất do giảm số chiều).

❖ Cách tìm được T:

Gọi ψ là vector trung bình của tất cả các vector x (kích thước $(N \times P) \times 1$) trong tập mẫu gồm M ảnh.

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

$$C = \frac{1}{M} \sum_k^M (x_k - \psi) \cdot (x_k - \psi)^T$$

Ma trận C có kích thước $(N \times P) \times (N \times P)$.

“**Ma trận hiệp phương sai** của tập hợp m biến ngẫu nhiên là một ma trận vuông hạng $(m \times m)$, trong đó các phần tử nằm trên đường chéo (từ trái sang phải, từ trên xuống dưới) lần lượt là phương sai tương ứng của các biến này (ta chú ý rằng $\text{Var}(X) = \text{Cov}(X, X)$), trong khi các phần tử còn lại (không nằm trên đường chéo) là các phương sai của đôi một hai biến ngẫu nhiên khác nhau trong tập hợp..”^[11]

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 mãn 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* . Sau đó, chỉ giữ lại K vector riêng ứng với K trị riêng lớn nhất để làm cơ sở cho không gian mới này.

6.3.2. Tìm EigenFaces

Ban đầu ta có một tập ảnh khuôn mặt gọi là tập ảnh huấn luyện (*training set*). Giả sử mỗi ảnh có kích thước $N \times P$, ta coi mỗi bức ảnh này là một vector trong không gian $N \times P$ chiều. Bây giờ, mỗi khuôn mặt là một vector, ta thấy những vector này không phân bố ngẫu nhiên trong không gian ảnh mà phân bố theo một quy luật tương đối này đó, ta có thể nói những vector này nằm trong một không gian con gọi là không gian khuôn mặt. Từ những vector trong tập huấn luyện, ta sẽ tìm một cơ sở trực chuẩn cho không gian khuôn mặt. Những vector thuộc cơ sở này có thể coi là những vector mang những nét tổng thể đặc trưng về khuôn mặt.

Các bước thực hiện tìm EigenFaces:

Bước 1: Giả sử tập huấn luyện có M ảnh: I_1, I_2, \dots, I_M , với khuôn mặt phải *chính diện* và tất cả các ảnh đều có *cùng kích thước* $N \times P$.



Hình 13. Tập ảnh huấn luyện.

Bước 2: Biểu diễn mỗi ảnh I_1, I_2, \dots, I_M , thành các vector T_1, T_2, \dots, T_M tương ứng. Mỗi vector T_i sẽ có kích thước $(N \times P) \times 1$.

Ví dụ: Để đơn giản, ta giả sử chỉ có 4 ảnh trong tập huấn luyện (kích thước mỗi ảnh là 3×3). Ta tính toán được các vector T_i có kích thước (9×1) :

$$T_1 = \begin{bmatrix} 225 \\ 229 \\ 48 \\ 251 \\ 33 \\ 238 \\ 0 \\ 255 \\ 217 \end{bmatrix} ; T_2 = \begin{bmatrix} 10 \\ 219 \\ 24 \\ 255 \\ 18 \\ 247 \\ 17 \\ 255 \\ 2 \end{bmatrix} ; T_3 = \begin{bmatrix} 196 \\ 35 \\ 234 \\ 232 \\ 59 \\ 244 \\ 243 \\ 57 \\ 226 \end{bmatrix} ; T_4 = \begin{bmatrix} 255 \\ 223 \\ 224 \\ 255 \\ 0 \\ 255 \\ 249 \\ 255 \\ 235 \end{bmatrix}$$

Bước 3: Tính vector khuôn mặt trung bình theo công thức:

$$\psi = \frac{1}{M} \sum_{i=1}^M T_i$$

Cụ thể, ta có:

$$\psi = \begin{bmatrix} 171.50 \\ 176.50 \\ 132.50 \\ 248.25 \\ 27.50 \\ 246.00 \\ 127.25 \\ 205.50 \\ 170 \end{bmatrix}$$

Bước 4: Tính sai số của các vector ảnh so với giá trị vector mặt trung bình.

$$\Phi_i = T_i - \psi$$

(i = 1 .. M)

Cụ thể ta có:

$$\Phi_1 = \begin{bmatrix} 53.50 \\ 52.50 \\ -84.50 \\ 2.75 \\ 5.50 \\ -8.00 \\ 127.25 \\ 49.50 \\ 47.00 \end{bmatrix}; \quad \Phi_2 = \begin{bmatrix} -161.50 \\ 42.50 \\ -108.50 \\ 6.75 \\ -9.50 \\ 1.00 \\ -110.25 \\ 49.50 \\ -168.00 \end{bmatrix}; \quad \Phi_3 = \begin{bmatrix} 24.50 \\ -141.50 \\ 101.50 \\ -16.25 \\ 31.50 \\ -2.00 \\ 115.75 \\ -148.50 \\ 56.00 \end{bmatrix}; \quad \Phi_4 = \begin{bmatrix} 83.50 \\ 46.50 \\ 91.50 \\ 6.75 \\ -27.50 \\ 9.00 \\ 121.75 \\ 49.50 \\ 65.00 \end{bmatrix}$$

Bước 5: Tính ma trận hiệp phương sai (covariance) C:

$$C = \frac{1}{M} \sum_n^M \Phi_n \Phi_n^T \xrightarrow{\frac{1}{M}} A \cdot A^T$$

Ma trận C sẽ có kích thước (NxP)x(NxP) = (3x3)x(3x3)

Trong đó:

- Ma trận A có kích thước (NxP)xM.

$$A = [\Phi_1,$$

$$\Phi_2, \dots, \Phi_M]$$

- Ma trận A^T là ma trận chuyển vị của ma trận A . (Để có được ma trận A^T , từ ma trận A , ta chuyển hàng thành cột, cột thành hàng. $\Rightarrow A^T$ có kích thước $M \times (N \times P)$).

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 $(3 \times 3) \times 4$

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

$$C = \begin{bmatrix} 36517 & -3639 & 23129 & -778 & 304 & 113 & 24000 & -4851 & 36446 \\ -3639 & 26747 & -19155 & 3045 & -5851 & 324 & -22083 & 28017 & -9574 \\ 23129 & -19155 & 37587 & -1997 & 1247 & 1188 & 45603 & -20097 & 25888 \\ -778 & 3045 & -1996 & 363 & -746.5 & 78 & -2153 & 3217 & -1476 \\ 304 & -5851 & 1247 & -747 & 1869 & -364 & 645 & -6237 & 1831 \\ 113 & 324 & 1188 & 78 & -364 & 150 & 1772 & 396 & -71 \\ 24000 & -22083 & 45603 & -2153 & 645.5 & 1772 & 56569 & -22919 & 26937 \\ -4851 & 28017 & -20097 & 3218 & -6237 & 396 & -22919 & 29403 & -1088 \\ 36446 & -9574 & 25888 & -1476 & 1831 & -71 & 26937 & -11088 & 37794 \end{bmatrix}$$

Ma trận C kích thước $(N \times P) \times (N \times P) = (9 \times 9)$.

Bước 6: Tính các vector riêng (*Eigenvectors*) của ma trận vuông $C = AA^T$.

Nhận thấy ma trận AA^T có kích thước $(N \times P) \times (N \times P)$, do kích thước ma trận này quá lớn nên ta không thể tìm các vector riêng và trị riêng tương ứng một cách trực tiếp, thay vào đó, ta sẽ tìm những vector riêng của ma trận $A^T A$ có kích thước $M \times M$.

Nếu v là một vector riêng của ma trận $A^T A$ và λ là trị riêng tương ứng, khi đó ta có:

$$(A^T A).v = \lambda.v \Leftrightarrow A.(A^T A).v = A.(\lambda.v) \Leftrightarrow (AA^T).(A.v) = \lambda.(A.v)$$

Tức $u = A.v$ là một vector riêng của ma trận AA^T .

Đặt $L = A^T A$, tìm V là tập hợp các vector riêng của L , D là tập hợp các trị riêng tương ứng.

V bao gồm K vectors riêng ứng với những trị riêng lớn hơn một giá trị nào đó hoặc ứng với K trị riêng lớn nhất trong D . ($K \ll NxP$)

Có 2 cách để xác định K :

Cách 1:

- Sắp xếp theo thứ tự 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:

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

$E = A.V$ là tập các vectors riêng của $A.A^T$. Do đây là những vectors riêng, mà nó lại có dạng khuôn mặt nên còn được gọi là Eigenfaces. E là ma trận $(NxP) \times K$, mỗi cột là một vector riêng.

Chuẩn hóa các vector cột trong E để thu được một cơ sở trực chuẩn của không gian khuôn mặt.

$$u_i (\|u_i\| = 1), \text{ nghĩa là: } u_i = \frac{u_i}{\|u_i\|}$$

Bước 6.1: Xét ma trận $L = A^T A$ có kích thước $M \times M = 4 \times 4$.

Cụ thể ta có:

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

Bước 6.2: Tính các vector riêng v_i (*eigenvectors*) của ma trận vuông $A^T A$ kích thước 4×4 .

- ✓ 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. Do đó, khi lập trình, ta có thể 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 khác 0 “*non=zero*”. Kết quả ta thu được 3 trị riêng (từ đó tính ra các vector riêng tương ứng).

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

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

$V = [v_1, v_2, v_3]$: Các vector riêng (*eigenvector*) của ma trận $A^T A$.

$D = [\lambda_1, \lambda_2, \lambda_3]$: Các trị riêng (*eigenvalues*) tương ứng với các vectors riêng.

Bước 6.3: Từ các vector riêng v_i (có kích thước $M \times 1 = 4 \times 1$), ta dễ dàng suy ra được các vector riêng u_i (kích thước $(N \times P) \times 1 = (3 \times 3) \times 1 = 9 \times 1$) cần tìm theo công thức:

$$u_i = A v_i \quad (*)$$

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

u_1, u_2, u_3 : Các vectors riêng (*eigenvectors*) của ma trận hiệp phương sai C ($A^T A$) cần tìm.

Bước 7: Chuẩn hóa các vectors của ma trận C để thu được một cơ sở trực chuẩn của không gian khuôn mặt.

$$u_i = \frac{u_i}{\|u_i\|}$$

Các vectors u_i sau khi chuẩn hóa:

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

Các vector này là các vector cơ sở của không gian mới. Ta gọi các vector này là các **EIGENFACES**.

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

6.3.3. So sánh khoảng cách-Compare Distance.

Xét ảnh H có cùng kích thước với những bức ảnh trong tập huấn luyện. H được xem là một vector trong không gian $N \times P$ chiều.

Đặt $G = H - \psi$, với ψ là vector ảnh trung bình.

Cho V là một không gian có tích vô hướng hữu hạn chiều và W là một không gian con của V . Giả sử W có một cơ sở trực chuẩn là $\{u_1, \dots, u_K\}$. Khi đó, hình chiếu trực giao của vector u bất kỳ lên W được xác định như sau:

$$pr_w u = u_0 = \sum_{i=1}^K (\vec{u}, \vec{u}_i) \cdot u_i$$

- Độ dài $\|u - u_0\|$ được gọi là khoảng cách từ u đến W .
- Tập hợp $c_i = (\vec{u}, \vec{u}_i)$, $i=1, \dots, K$ được gọi là tọa độ của u_0 trong không gian W .
- Tìm $C = E^T G$ là tọa độ hình chiếu G_f của G lên không gian khuôn mặt. C là vector

$$G_f = \sum_{i=1}^K c_i \cdot e_i \quad \text{cột} \quad K \times 1.$$

với $c_i = C(i, 1)$; $e_i = E(1, i)$.

Với A_i là một cột trong ma trận A (tương ứng với bức ảnh T_i trong tập huấn luyện). Ta tính $C_i = E^T \cdot A_i$ là tọa độ của hình chiếu A_{if} của A_i lên không gian khuôn mặt.

Tính: $S_i = \|C - C_i\|$ xem như khoảng cách từ H đến bức ảnh T_i trong tập huấn luyện.

Gọi $\beta = \text{eigenDistanceThreshold}$ là khoảng cách ngưỡng riêng, khi đó:

- Nếu $S_i < \beta$ thì T_i là bức ảnh của cùng một người với H (H đủ gần với T_i).