

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



HISTOGRAM OF ORIENTED GRADIENT
TRONG BÀI TOÁN PHÁT HIỆN NGƯỜI ĐI BỘ

Môn học: Thị giác máy tính

Giảng viên hướng dẫn: TS. Nguyễn Thị Oanh

Sinh viên thực hiện: Nguyễn Đức Thắng - MSHV: 20166769

Lê Hoàng Ngân - MSHV: 20162886

HÀ NỘI, 10/2020

Mục lục

1 Kiến thức cơ sở	2
1.1 Support Vector Machine (SVM)	2
1.1.1 Khoảng cách từ một điểm đến một siêu phẳng	2
1.1.2 Bài toán phân chia hai lớp dữ liệu	3
1.1.3 Xây dựng bài toán tối ưu cho SVM	4
1.2 Histogram of Oriented Gradient (HOG)	7
1.2.1 Tiền xử lý	9
1.2.2 Tính Gradient	9
1.2.3 Tính vector đặc trưng cho từng cell	11
1.2.4 Chuẩn hóa block	15
1.2.5 Tính toán vector đặc trưng HOG	18
2 Histogram of Oriented Gradient cho bài toán phát hiện người đi bộ	19
2.1 Phát biểu bài toán	19
2.2 Mô tả dữ liệu	21
2.2.1 Dữ liệu người đi bộ Penn Fudan	21
2.2.2 Bộ dữ liệu INRIA Person	22
2.3 Kết quả thực nghiệm	22
2.3.1 Cấu trúc các file mã nguồn	22
2.3.2 Huấn luyện	23
2.3.3 Kiểm thử	23

Lời mở đầu

Trong những năm gần đây, trí tuệ nhân tạo ngày càng phát triển và đi vào cuộc sống hàng ngày. Trong đó, thị giác máy tính (computer vision) là một nhánh của trí tuệ nhân tạo tập trung vào các ứng dụng trên ảnh và video. Trong trí tuệ nhân tạo thì thị giác máy tính là một trong những phần khó. Các bài toán ứng dụng của thị giác máy tính bao gồm: phân loại đối tượng trong ảnh, phát hiện đối tượng, chú thích cho ảnh...

Phát hiện người đi bộ là một trong những bài toán của thị giác máy tính, có vai trò quan trọng trong nhiều ứng dụng thực tiễn. Trong báo cáo này, với mục đích nghiên cứu về phân đoạn đối tượng trong thị giác máy tính, em thực hiện bài toán phát hiện người đi bộ sử dụng Histogram of Oriented Gradient (HOG).

Nội dung báo cáo này được chia làm 2 chương:

- **Chương 1:** Kiến thức cơ sở, tổng quan về Support Vector Machine (SVM) và Histogram of Oriented Gradient (HOG).
- **Chương 2:** Áp dụng HOG cho bài toán phát hiện người đi bộ.

Chân thành cảm ơn TS. Nguyễn Thị Oanh đã hướng dẫn, trang bị cho nhóm em các kiến thức cần thiết để thực hiện báo cáo này và cung cấp dữ liệu để nhóm có thể hoàn thành đề tài. Mặc dù đã cố gắng rất nhiều, nhưng với kiến thức và thời gian hạn chế nên không thể tránh khỏi những thiếu sót. Rất mong được sự góp ý của thầy cô và bạn bè để báo cáo này được hoàn thiện hơn.

Hà Nội, ngày 12 tháng 10 năm 2020

Chương 1

Kiến thức cơ sở

1.1 Support Vector Machine (SVM)

Support Vector Machine (SVM) là một thuật toán thuộc nhóm các bài toán học có giám sát (*Supervised Learning*), là một trong những thuật toán phân lớp phổ biến và hiệu quả.

1.1.1 Khoảng cách từ một điểm đến một siêu phẳng

Trong không gian hai chiều, khoảng cách từ một điểm có tọa độ (x_0, y_0) tới đường thẳng có phương trình $w_1x + w_2y + b = 0$ được xác định bởi công thức:

$$\frac{|w_1x_0 + w_2y_0 + b|}{\sqrt{w_1^2 + w_2^2}}$$

Trong không gian ba chiều, khoảng cách từ một điểm có tọa độ (x_0, y_0, z_0) tới một mặt phẳng có phương trình $w_1x + w_2y + w_3z + b = 0$ được xác định bởi công thức:

$$\frac{|w_1x_0 + w_2y_0 + w_3z_0 + b|}{\sqrt{w_1^2 + w_2^2 + w_3^2}}$$

Hơn nữa, khi bỏ dấu trị tuyệt đối ở tử số, ta có thể xác định được điểm đó nằm về phía nào của đường thẳng hay mặt phẳng đang xét. Những điểm làm cho biểu thức trong dấu giá trị tuyệt đối mang dấu dương nằm về cùng một phía (phía dương), những điểm làm cho giá trị này mang dấu âm nằm về phía còn lại (phía âm). Những điểm nằm trên đường thẳng/mặt phẳng sẽ làm cho tử số có giá trị bằng 0, tức khoảng cách bằng 0.

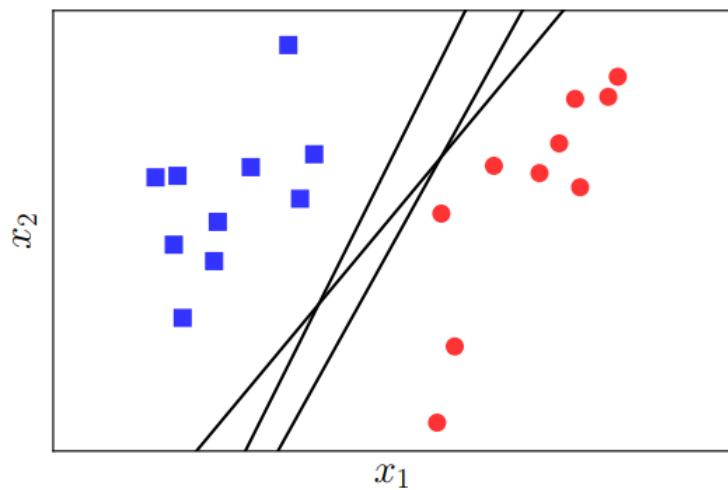
Các công thức này có thể được tổng quát lên cho trường hợp không gian d chiều. Khoảng cách từ một điểm (vector) có tọa độ $(x_{10}, x_{20}, \dots, x_{d0})$ tới siêu phẳng (*hyperplane*) có phương trình $w_1x_1 + w_2x_2 + \dots + w_dx_d + b = 0$ được xác định bởi:

$$\frac{|w_1x_{10} + w_2x_{20} + \dots + w_dx_{d0} + b|}{\sqrt{w_1^2 + w_2^2 + w_3^2 + \dots + w_d^2}} = \frac{|w^T x_0 + b|}{\|w\|_2}$$

Với $x_0 = [x_{10}, x_{20}, \dots, x_{d0}]^T$, $w = [w_1, w_2, \dots + w_d]^T$

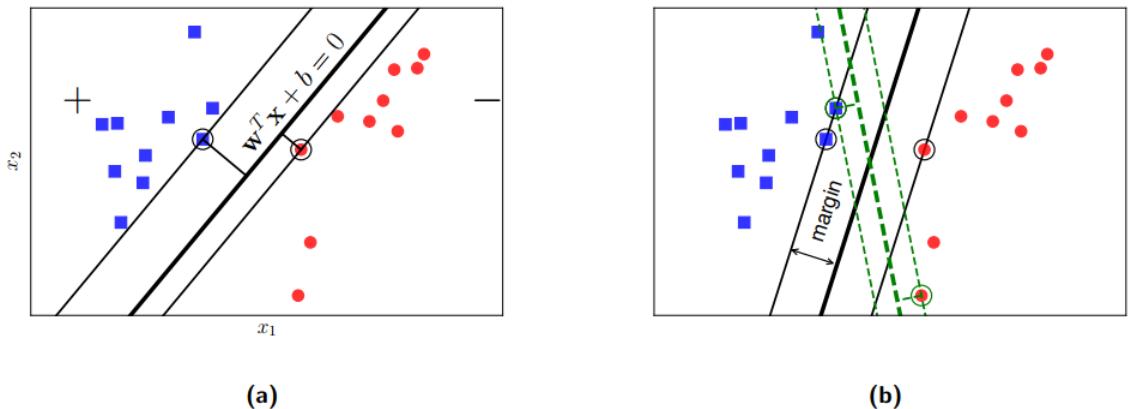
1.1.2 Bài toán phân chia hai lớp dữ liệu

Giả sử rằng có hai lớp dữ liệu được mô tả bởi các điểm (*feature vector*) trong không gian nhiều chiều, hơn nữa, hai lớp dữ liệu này là linearly separable, tức là tồn tại một siêu phẳng phân chia chính xác hai lớp đó.



Hình 1.1: Hai lớp dữ liệu đỏ và xanh là linearly separable. Có vô số các đường thẳng có thể phân tách chính xác hai lớp dữ liệu này

Câu hỏi đặt ra ở đây là: trong vô số các mặt phân chia, đâu là mặt tốt nhất? Trong ba đường thẳng minh họa trong hình 1.1, có hai đường thẳng lệch về phía lớp màu đỏ. Việc này có thể khiến cho các điểm màu đỏ trong tương lai bị phân lớp lỗi thành điểm màu xanh. Do vậy, việc định nghĩa siêu phẳng để phân lớp dữ liệu như nào được gọi là tối ưu? Để trả lời câu hỏi này, chúng ta cần tìm một tiêu chuẩn để xác định thế nào là tối ưu cho mỗi lớp. Nhìn bằng mắt thường chúng ta có thể thấy, đường tối ưu là đường tạo cho ta có cảm giác 2 lớp dữ liệu nằm cách xa nhau và cách xa đường đó nhất. Hình 1.2(a), lớp màu đỏ sẽ không được tối ưu vì đường phân chia gần nó hơn lớp màu xanh rất nhiều. Chúng ta cần một đường phân chia sao cho khoảng cách từ điểm gần nhất của mỗi lớp (các điểm được khoanh tròn) tới đường phân chia là như nhau. Khoảng cách như nhau này được gọi là *Margin* (biên, lề).



Hình 1.2: Ý tưởng của SVM: Margin của một lớp được định nghĩa là khoảng cách từ các điểm gần nhất của lớp đó tới mặt phân chia. Margin của hai lớp phải bằng nhau và lớn nhất có thể.

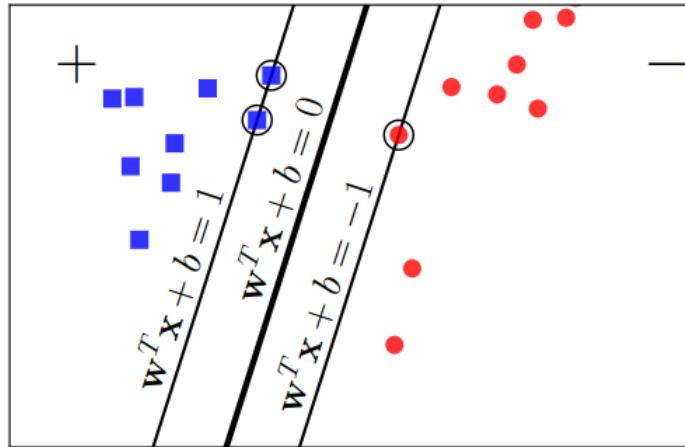
Xét hình 1.2(b), khi khoảng cách từ đường phân chia tới các điểm gần nhất của mỗi lớp là như nhau. Xét hai cách phân chia bởi đường nét liền màu đen và đường nét đứt màu lục, đường nào tối ưu hơn? Rõ ràng đó phải là đường nét màu đen vì nó tạo ra một margin rộng hơn.

Việc margin rộng hơn sẽ mang lại hiệu ứng phân lớp tốt hơn vì sự phân chia giữa hai lớp là rạch ròi hơn. Bài toán tối ưu trong SVM chính là bài toán tìm đường phân chia sao cho margin giữa hai lớp là lớn nhất. Đây cũng là lý do vì sao SVM còn được gọi là *Maximum Margin Classifier*.

1.1.3 Xây dựng bài toán tối ưu cho SVM

Giả sử rằng các cặp dữ liệu trong tập huấn luyện là $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$ với vector $x_i \in \mathbb{R}^d$ thể hiện đầu vào của một điểm dữ liệu và y_i là nhãn của điểm dữ liệu đó, d là số chiều của dữ liệu và N là số điểm của dữ liệu. Giả sử nhãn của mỗi điểm dữ liệu được xác định bởi $y_i = 1$ hoặc $y_i = -1$.

Dễ dẽ hình dung, chúng ta cùng làm với các ví dụ trong không gian hai chiều. Giả sử rằng các điểm màu xanh có nhãn là 1, các điểm tròn đỏ có nhãn là -1, và mặt $w^T x + b = w_1 x_1 + w_2 x_2 + b = 0$ có mặt phân chia giữa hai lớp (hình 1.3). Hơn nữa, lớp màu xanh nằm về phía dương, lớp màu đỏ nằm về phía âm của mặt phân chia. Nếu ngược lại, ta chỉ cần đổi dấu w và b . Ta cần xác định siêu phẳng được mô tả bởi các hệ số w và b .



Hình 1.3: Giả sử mặt phân chia có phương trình $w^T x + b = 0$. Không mất tính tổng quát, bằng cách nhân các hệ số w, b với các hằng số phù hợp, ta có thể giả sử rằng điểm gần nhất của lớp màu xanh tới mặt này thỏa mãn $w^T x + b = 1$. Khi đó, điểm gần nhất của lớp đỏ thỏa mãn $w^T x + b = -1$

Ta quan sát thấy một điểm quan trọng như sau: với cặp dữ liệu (x_n, y_n) bất kỳ, khoảng cách từ điểm đó tới mặt phân chia là:

$$\frac{y_n(w^T x_n + b)}{\|w\|_2}$$

Điều này có thể được nhận thấy vì theo giả sử ở trên, y_n luôn cùng dấu với $w^T x_n$. Từ đó suy ra y_n cùng dấu với $(w^T x_n + b)$, vì vậy tử số luôn là một đại lượng không âm. Với mặt phân chia này, margin được tính là khoảng cách gần nhất từ một điểm (trong cả hai lớp, vì cuối cùng margin của cả hai lớp sẽ như nhau) tới mặt đó, tức là:

$$margin = \min_n \frac{y_n(w^T)}{\|w\|_2}$$

Bài toán tối ưu của SVM chính là việc xác định w, b sao cho margin đạt giá trị lớn nhất:

$$(w, b) = \arg \max_{w,b} \left\{ \min_n \frac{y_n(w^T)}{\|w\|_2} \right\} = \arg \max_{w,b} \left\{ \frac{1}{\|w\|_2} \min_n y_n(w^T x_n + b) \right\} \quad (1.1)$$

Có một nhận xét quan trọng là nếu ta thay vector hệ số w bởi kw , và b bởi kb trong đó k là một hằng số dương bất kỳ thì mặt phân chia không thay đổi, tức là khoảng cách từ từng điểm đến mặt phân chia không đổi, tức margin không đổi. Vì vậy, ta có thể giả sử:

$$y_n(w^T x_n + b) = 1$$

Với những điểm nằm gần mặt phân chia nhất (được khoanh tròn trong hình 1.3). Như vậy, với mọi n ta có:

$$y_n(w^T x_n + b) \geq 1$$

Vậy bài toán tối ưu 1.1 có thể đưa về bài toán tối ưu có ràng buộc có dạng:

$$(w, b) = \arg \max_{w,b} \frac{1}{\|w\|_2} \quad (1.2)$$

$$\text{Vđk : } 1 - y_n(w^T x_n + b) \leq 0 \quad \forall n = 1, 2, \dots, N$$

Bằng một biến đổi đơn giản, ta có thể đưa bài toán này về dạng:

$$(w, b) = \arg \max_{w,b} \frac{1}{2} \|w\|_2^2 \quad (1.3)$$

$$\text{Vđk : } 1 - y_n(w^T x_n + b) \leq 0 \quad \forall n = 1, 2, \dots, N$$

Ở đây, chúng ta lấy nghịch đảo hàm mục tiêu, bình phương nó để được một hàm khả vi, và nhân với $1/2$ để biểu thức có đạo hàm đẹp hơn.

Trong bài toán 1.3, hàm mục tiêu là một norm, nên là một hàm lồi. Các hàm bất đẳng thức ràng buộc là các hàm tuyến tính theo w, b , nên chúng cũng là hàm lồi. Vậy bài toán 1.3 có hàm mục tiêu là lồi, và các hàm ràng buộc cũng là lồi, nên nó là một bài toán lồi. Hơn nữa, nó là một quadratic programming vì hàm mục tiêu là một quadratic form. Từ đây, có thể suy ra nghiệm cho SVM là duy nhất.

Đến đây thì bài toán có thể giải thông qua bài toán đối ngẫu của nó và sử dụng phương pháp nhân tử Lagrange. Lúc này, ta cần xác định các giá trị λ như sau:

$$\begin{aligned} \lambda &= \arg \max_{\lambda} \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n,m=1}^N \lambda_n \lambda_m y_n y_m x_n x_m \\ \text{Vđk: } &\begin{cases} \lambda_i \geq 0 \\ \sum_{n=1}^N \lambda_n y_n = 0, n \in [1, N] \end{cases} \end{aligned}$$

Việc giải λ có thể được thực hiện bằng phương pháp quy hoạch động bậc 2 (Quadratic Programming). Với Python, ta có thể sử dụng thư viện CVXPY. Sau khi tìm được λ thì ta xác định được các tham số:

$$\begin{aligned} w &= \sum_{n=1}^N \lambda_n y_n x_n \\ b_n &= y_n - \sum_{m=1}^N \lambda_m y_m x_m x_n \end{aligned}$$

Ở đây, (x_n, y_n) là một điểm dữ liệu bất kỳ nào đó nằm trên đường biên gốc. Điểm dữ liệu này còn được gọi là Support Vector. Tuy nhiên, thường người ta xác định b bằng phép lấy trung bình tổng của tất cả các b_n . Giả sử, ta có tập \mathbb{S} các Support Vector thì:

$$b = \frac{1}{|\mathbb{S}|} \sum_{i \in \mathbb{S}} \left(y_i - \sum_{j=1}^N \lambda_j y_j x_j x_i \right)$$

Khi đó, một điểm dữ liệu mới sẽ được phân loại dựa theo:

$$h(x) = \operatorname{sgn}\left(\sum_{i=1}^N \lambda_i y_i x_i^T x + b\right)$$

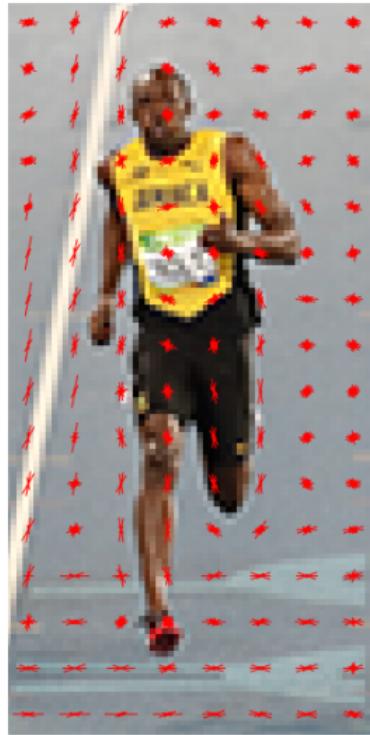
Như vậy, chỉ cần các điểm Support Vector trên đường biên gốc là ta có thể ước lượng được các tham số tối ưu cho bài toán. Việc này rất có lợi khi tính toán giúp phương pháp này tiết kiệm được tài nguyên thực thi.

Ngoài ra, khi triển khai trên Python, chúng ta có thể sử dụng trực tiếp hàm `sklearn.svm.SVC` của thư viện `scikit-learn` để giải quyết bài toán.

1.2 Histogram of Oriented Gradient (HOG)

Trích xuất đặc trưng (*feature extraction*) là một quá trình nhằm biến dữ liệu phức tạp đầu vào thành một cách biểu diễn dữ liệu đơn giản hơn, phù hợp hơn cho các thuật toán học máy. Dữ liệu sau khi xử lý đã được lược bỏ phần dữ liệu dư thừa, giữ lại những dữ liệu có ích cho bài toán cần xử lý. Một số thuật toán để trích xuất như HOG, SIFT. Trong báo cáo này, nhóm sẽ tập trung giới thiệu về Histogram of Oriented Gradient (HOG).

HOG (Histogram of Oriented Gradient) là một feature descriptor được sử dụng trong lĩnh vực thị giác máy tính và xử lý hình ảnh, dùng để phát hiện một đối tượng. Các khái niệm về HOG được nêu ra từ năm 1986 tuy nhiên cho đến năm 2005, HOG mới được sử dụng rộng rãi sau khi Navneet Dalal và Bill Triggs công bố những bổ sung về HOG. HOG tương tự như các biểu đồ edge orientation, scale-invariant feature transform descriptors (như SIFT, SURF,...). Mục đích của feature descriptor là trừu tượng hóa đối tượng bằng cách trích xuất ra những đặc trưng của đối tượng và bỏ đi những thông tin không hữu ích. Vì vậy, HOG được sử dụng chủ yếu để mô tả hình dạng và sự xuất hiện của một đối tượng trong ảnh.



Bản chất của phương pháp HOG là sử dụng thông tin về sự phân bố của các cường độ Gradient (*intensity gradient*) hoặc của hướng biên (*edge directions*) để mô tả các đối tượng cục bộ trong ảnh. Các toán tử HOG được cài đặt bằng cách chia nhỏ một bức ảnh thành các vùng con, được gọi là cell, với mỗi cell, ta sẽ tính toán một histogram về các hướng của gradient cho các điểm nằm trong cell. Ghép các histogram lại với nhau ta sẽ có một biểu diễn cho bức ảnh ban đầu. Để tăng cường hiệu năng nhận dạng, các histogram cục bộ có thể được chuẩn hóa về độ tương phản bằng cách tính một ngưỡng cường độ trong một vùng lớn hơn cell, gọi là các khối (block), và sử dụng giá trị ngưỡng đó để chuẩn hóa tất cả các cell trong block. Kết quả sau bước chuẩn hóa sẽ là một vector đặc trưng có tính bất biến cao hơn đối với các thay đổi về điều kiện ánh sáng.

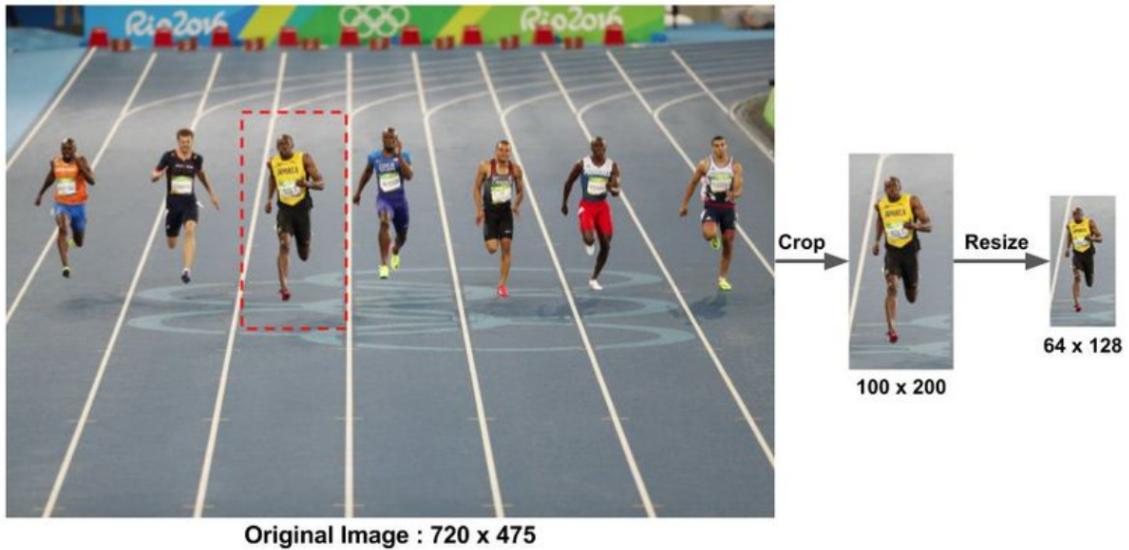
Có 5 bước cơ bản để xây dựng một vector HOG cho hình ảnh, bao gồm:

- Bước 1: Tiền xử lý
- Bước 2: Tính gradient
- Bước 3: Tính vector đặc trưng cho từng ô (cell)
- Bước 4: Chuẩn hóa block
- Bước 5: Tính toán vector HOG

Tiếp theo, bài báo cáo sẽ đi vào từng bước.

1.2.1 Tiền xử lý

Trong bài toán này, để thuận tiện cho việc chia đều hình ảnh thành các cell, block và tính toán đặc trưng ở các bước tiếp theo, chúng ta cần resize kích thước tất cả các hình ảnh trong tập dữ liệu về một kích thước chung.

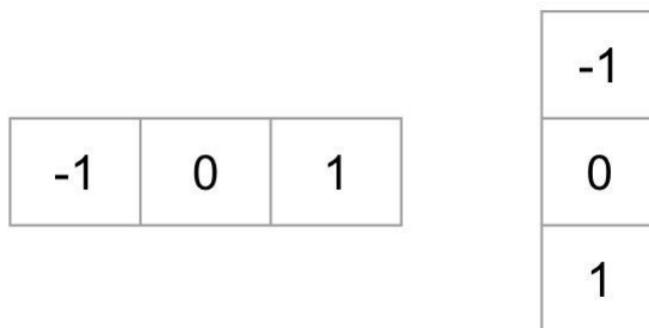


1.2.2 Tính Gradient

Bước này được thực hiện bằng hai phép nhân chập ảnh gốc với 2 chiều, tương ứng với các toán tử lấy đạo hàm theo 2 hướng Ox, Oy . Trong đó, 2 hướng tương ứng là:

$$D_x = [-1, 0, 1]$$

$$D_y = [1, 0, -1]^T$$



Gọi ảnh đầu vào là I , ta sẽ có 2 ảnh đạo hàm riêng theo Ox, Oy :

$$I_x = I * D_x$$

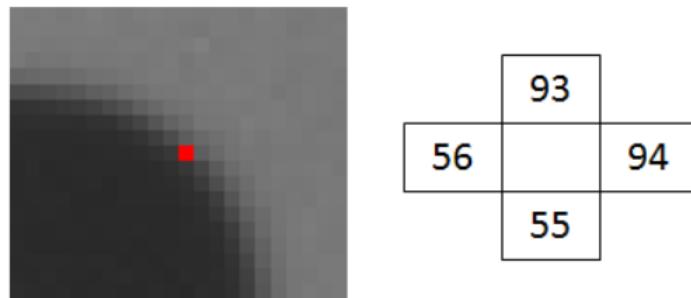
$$I_y = I * D_y$$

Khi đó, ta có thể tính được gradient bao gồm hai thành phần là: cường độ (*Gradient Magnitude*), và hướng (*Gradient Direction*) theo công thức:

$$\text{Cường độ : } |G| = \sqrt{I_x^2 + I_y^2}$$

$$\text{Hướng : } \theta = \arctan \frac{I_y}{I_x}$$

Để rõ hơn về vấn đề, ta xét một điểm ảnh:



Chúng ta sẽ áp dụng các công thức trên để tính được gradient tại điểm ảnh này:

$$I_x = I * D_x = [56, x, 94] * [-1, 0, 1]^T = [38]$$

$$I_y = I * D_y = [93, y, 55]^T * [1, 0, -1] = [38]$$

$$\text{Cường độ: } |G| = \sqrt{I_x^2 + I_y^2} = \sqrt{38^2 + 38^2} \approx 53.74$$

$$\text{Hướng: } \theta = \arctan \frac{I_y}{I_x} \approx 0.79$$

Đối với ảnh màu, gradient của 3 kênh (red, green, blue) được đánh giá. Độ lớn của gradient tại một điểm ảnh là giá trị lớn nhất của cường độ gradient của 3 kênh, và góc là góc tương ứng với gradient tối đa.

Sau bước tính gradient, kết quả thu được là:



Left : Absolute value of x-gradient. Center : Absolute value of y-gradient.
Right : Magnitude of gradient.

1.2.3 Tính vector đặc trưng cho từng cell

Để tính toán vector đặc trưng cho từng cell, chúng ta cần chia hình ảnh thành các block, mỗi block lại chia đều thành các cell. Để xác định được số block, chúng ta sẽ sử dụng công thức sau:

$$n_{block;image} = \left(\frac{W_{image} - W_{block} * W_{cell}}{W_{cell}} + 1 \right) * \left(\frac{H_{image} - H_{block} * H_{cell}}{H_{cell}} + 1 \right)$$

Trong đó:

$W_{image}, W_{block}, W_{cell}$: lần lượt là chiều rộng của ảnh, block, cell

$H_{image}, H_{block}, H_{cell}$: lần lượt là chiều dài của ảnh, block, cell

Các block có thể xếp chồng lên nhau. Sau khi xác định số block và kích thước mỗi block, cell, để tính toán vector đặc trưng cho từng cell, chúng ta cần:

- Chia không gian hướng thành p bin (số chiều vector đặc trưng của ô).
- Rời rạc hóa góc hướng nghiêng tại mỗi điểm ảnh vào trong các bin.

Giả sử hướng góc nghiêng tại pixel ở vị trí (x, y) có độ lớn là $\alpha(x, y)$.

- Trường hợp rời rạc hóa unsigned - HOG với $p = 9$:

$$B(x, y) = \text{round}\left(\frac{p * \alpha(x, y)}{\pi}\right) \mod p$$

- Trường hợp rời rạc hóa signed-HOG với $p = 18$:

$$B(x, y) = \text{round}\left(\frac{p * \alpha(x, y)}{2\pi}\right) \mod p$$

Giá trị bin được định lượng bởi tổng cường độ biến thiên của các pixels thuộc về bin đó. Sau khi tính toán đặc trưng cell, ta sẽ nối các vector đặc trưng cell để thu được vector đặc trưng block. Số chiều vector đặc trưng block được tính theo công thức:

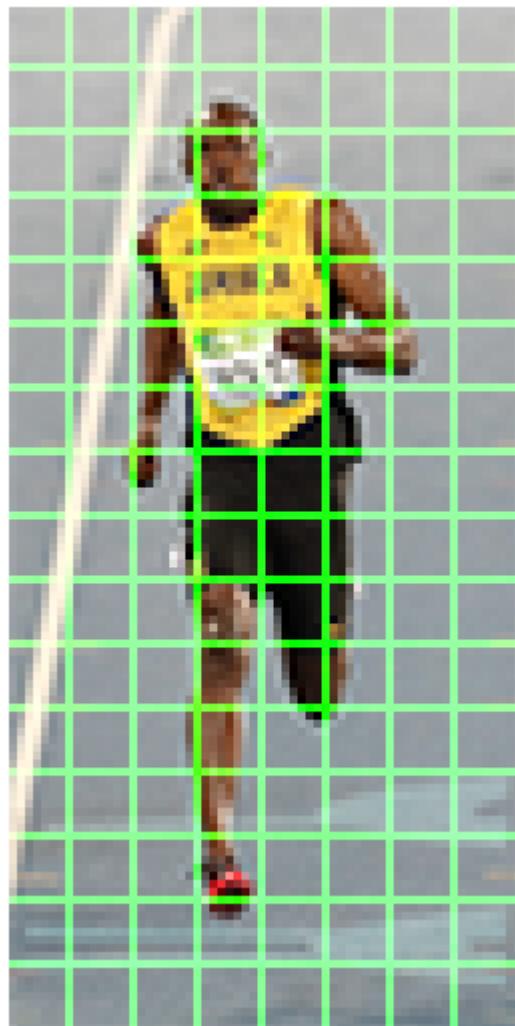
$$\text{size}_{block} = n * \text{size}_{cell}$$

Trong đó:

n là số cell trong block

size_{cell} là số chiều của vector đặc trưng cell ($\text{size}_{cell} = 9$ nếu sử dụng unsigned-HOG và $\text{size}_{cell} = 18$ nếu sử dụng signed-HOG)

Xét một ví dụ:

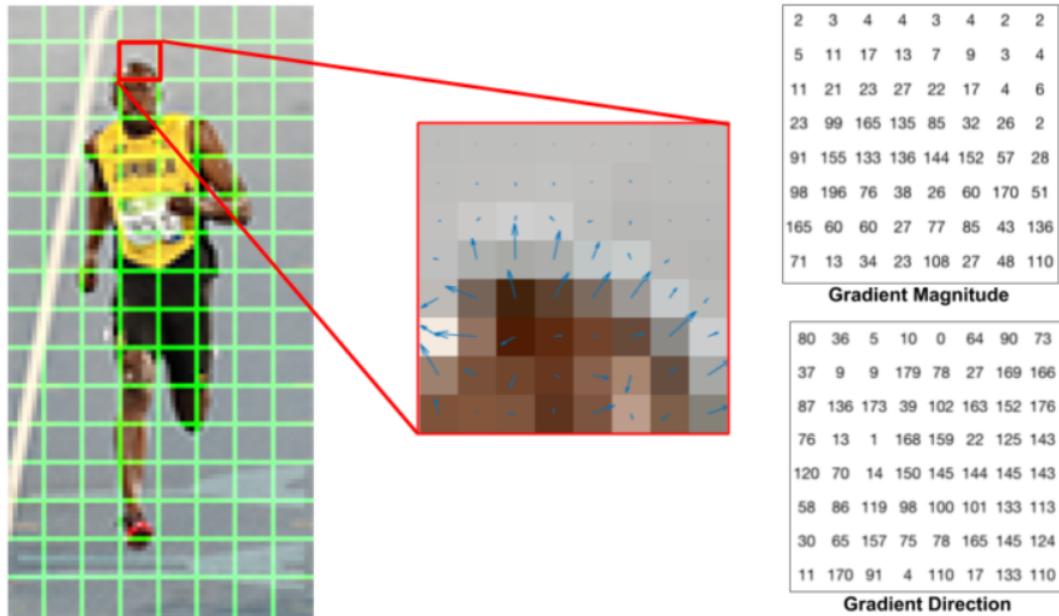


Trong trường hợp này, hình ảnh có kích thước là 64×128 , ta sẽ chia hình ảnh thành các block có kích thước 16×16 . Mỗi block sẽ bao gồm 4 cell, mỗi cell có kích thước là 8×8 .

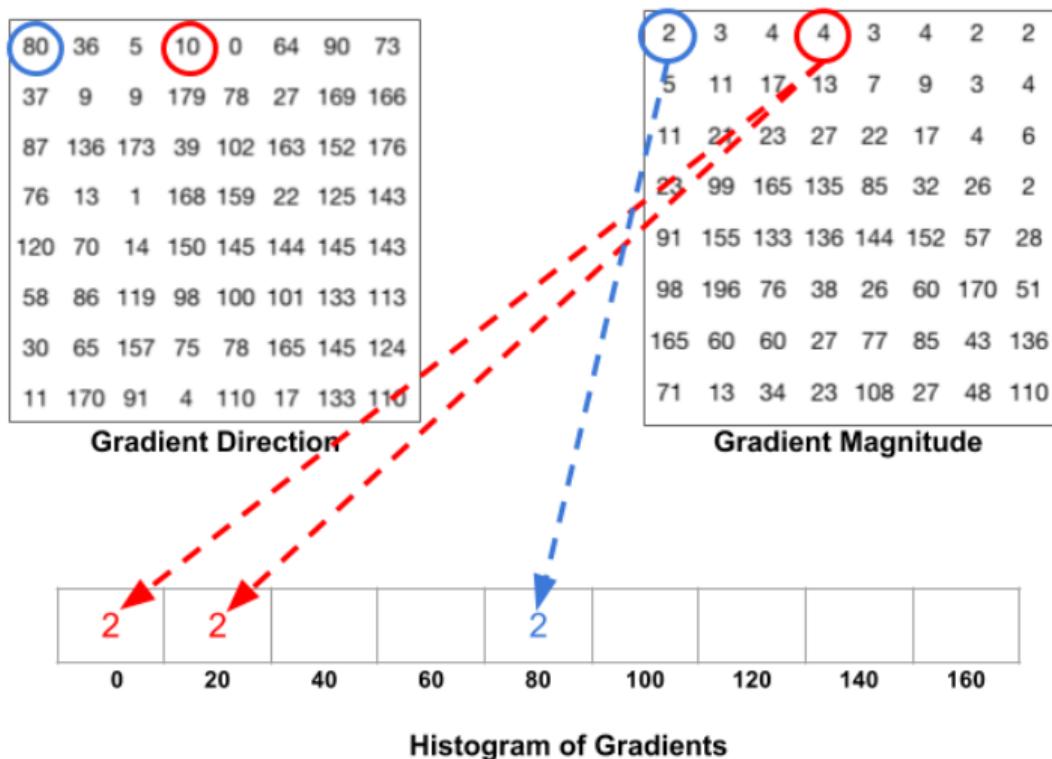
Tiếp theo, tiến hành tính toán đặc trưng HOG tại mỗi cell sử dụng không gian hướng 9 bin, trường hợp unsigned-HOG. Hướng gradient sẽ chạy trong khoảng từ 0° đến 180° , trung bình 20° mỗi bin. Cụ thể:

- Hướng $0^\circ - 20^\circ$: vote hướng thuộc đoạn này vào bin 0.
- Hướng $20^\circ - 40^\circ$: vote hướng thuộc đoạn này vào bin 1.
- Hướng $40^\circ - 60^\circ$: vote hướng thuộc đoạn này vào bin 2.
- Hướng $60^\circ - 80^\circ$: vote hướng thuộc đoạn này vào bin 3.
- Hướng $80^\circ - 100^\circ$: vote hướng thuộc đoạn này vào bin 4.
- Hướng $100^\circ - 120^\circ$: vote hướng thuộc đoạn này vào bin 5.
- Hướng $120^\circ - 140^\circ$: vote hướng thuộc đoạn này vào bin 6.
- Hướng $140^\circ - 160^\circ$: vote hướng thuộc đoạn này vào bin 7.
- Hướng $160^\circ - 180^\circ$: vote hướng thuộc đoạn này vào bin 8.

Khi có hướng rơi vào bin, ta không vote kiểu bình thường là giá trị trong bin tăng lên 1 đơn vị, mà ta sẽ tăng lên một giá trị bằng biên độ của hướng đó. Ví dụ: hướng 42° , biên độ 0.27, vote vào bin 2, giá trị bin 2 tăng lên 0.27. Hướng của các pixel trong cell vote vào bin nào thì giá trị của bin đó tăng dần lên. Khi đó, tại mỗi cell, xây dựng biểu đồ cường độ gradient bằng cách vote các pixel vào biểu đồ. Trọng số vote của mỗi pixel sẽ phụ thuộc vào hướng và cường độ gradient của pixel đó. Ví dụ:

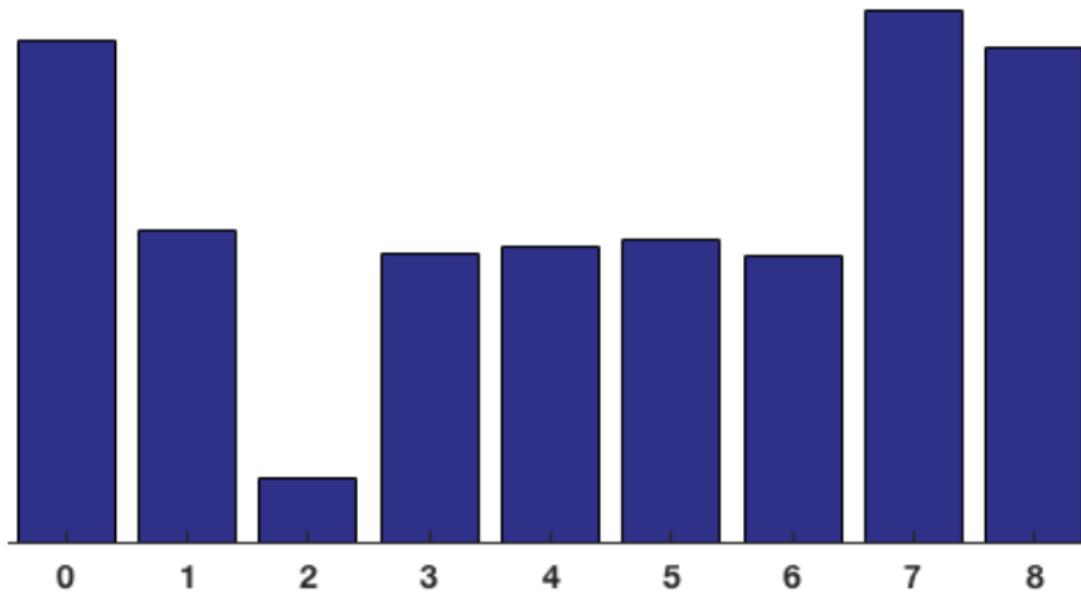


Center : The RGB patch and gradients represented using arrows. Right : The gradients in the same patch represented as numbers



Như trong hình trên, đầu tiên là pixel có bao quanh bởi màu xanh lam. Nó có hướng

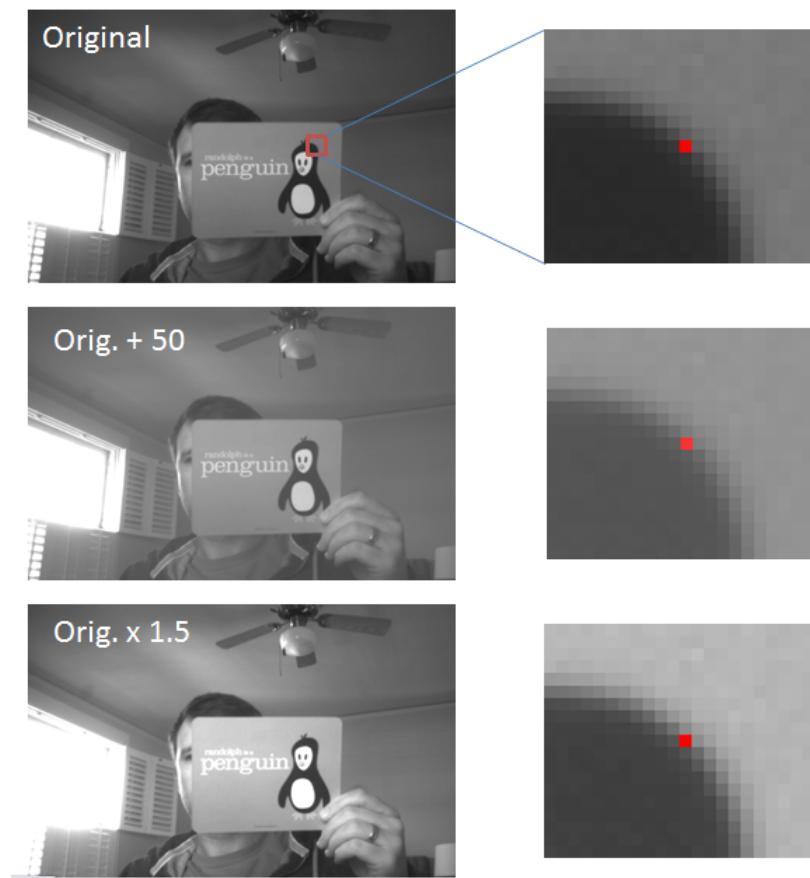
80° và cường độ là 2, vì vậy, ta thêm 2 vào bin thứ 5. Tiếp theo pixel có bao quanh màu đỏ. Nó có hướng 10° và cường độ là 4. Vì không có bin 10° , nên ta vote vào bin 0 và bin 1, mỗi bin thêm 2 đơn vị. Sau khi vote hết các pixel trong một cell kích thước 8×8 vào 9 bin, ta có thể thu được kết quả như sau:



1.2.4 Chuẩn hóa block

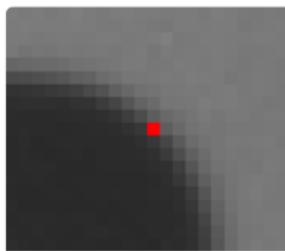
Hiện tại mỗi cell đang mang thông tin histogram trên vùng ảnh 8×8 , các thông tin này mang tính chất cục bộ. Để tăng cường hiệu quả nhận dạng, các histogram cục bộ sẽ được chuẩn hóa về độ tương phản bằng cách tính một ngưỡng cường độ trong một block và sử dụng giá trị đó để chuẩn hóa tất cả các ô trong block. Kết quả sau bước chuẩn hóa sẽ là một vector đặc trưng có tính bất biến cao hơn đối với các thay đổi về điều kiện ánh sáng.

Đầu tiên, hãy xem xét ảnh hưởng của việc chuẩn hóa tới các vector gradient trong ví dụ sau:



Trong hình ảnh trên, trường hợp đầu tiên là một ô của hình ảnh ban đầu. Trường hợp thứ hai, tất cả các giá trị pixel đã được tăng lên 50. Trong trường hợp thứ 3, tất cả các giá trị pixel được nhân với 1.5. Dễ dàng thấy được, trường hợp thứ 3 hiển thị độ tương phản gia tăng. Ảnh hưởng của phép nhân là làm các điểm ảnh sáng trở nên sáng hơn nhiều, trong khi các điểm ảnh tối chỉ sáng hơn một chút, do đó làm tăng độ tương phản giữa các phần sáng và tối của hình ảnh.

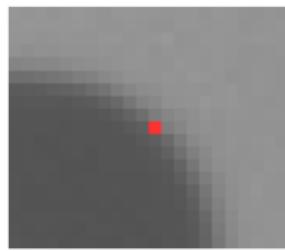
Hãy nhìn vào các giá trị pixel thực tế và sự thay đổi của vector gradient của 3 trường hợp trên trong hình ảnh sau:



	93	
56		94
	55	

$$\nabla f = \begin{bmatrix} 38 \\ 38 \end{bmatrix}$$

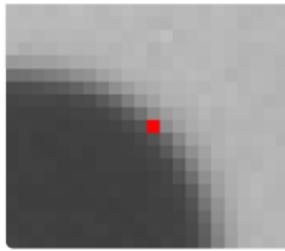
$$|\nabla f| = \sqrt{(38)^2 + (38)^2} = 53.74$$



	143	
106		144
	105	

$$\nabla f = \begin{bmatrix} 38 \\ 38 \end{bmatrix}$$

$$|\nabla f| = \sqrt{(38)^2 + (38)^2} = 53.74$$



	140	
84		141
	83	

$$\nabla f = \begin{bmatrix} 57 \\ 57 \end{bmatrix}$$

$$|\nabla f| = \sqrt{(57)^2 + (57)^2} = 80.61$$

Trong đó, các con số trong các cell là giá trị pixel của các điểm ảnh lân cận điểm ảnh được đánh dấu màu đỏ. ΔF là đạo hàm riêng hai hướng của điểm đánh ($[I_x, I_y]$). $|\Delta F|$ là giá trị cường độ điểm ảnh (Gradient Magnitude), tính theo công thức đã được giới thiệu ở trên.

Trong trường hợp 1 và 2, giá trị cường độ vector gradient của chúng tương đương nhau, nhưng trong trường hợp thứ 3, cường độ vector gradient đã tăng lên 1.5 lần. Nếu chia 3 vector bằng độ lớn tương ứng, ta sẽ nhận được các kết quả tương đương cho cả 3 trường hợp. Vì vậy, trong ví dụ trên, chúng ta thấy rằng bằng cách chia các vector gradient theo độ lớn của chúng, chúng ta có thể biến chúng thành bất biến để thay đổi độ tương phản.

Có nhiều phương pháp có thể chuẩn hóa block. Gọi v là vector cần chuẩn hóa chứa tất cả các histogram của một block. $\|v_k\|$ là giá trị chuẩn hóa của v theo các chuẩn $k = 1, 2$ và e là một hằng số có giá trị nhỏ. Khi đó, các giá trị chuẩn hóa có thể tính bằng một trong các công thức sau:

$$\text{L2-norm: } f = \frac{v}{\sqrt{\|v\|^2 + e^2}}$$

$$\text{L1-norm: } f = \frac{v}{\|v\| + e}$$

$$\text{L1-sqrt: } f = \sqrt{\frac{v}{\|v_1\| + e}}$$

Trong đó, với L1-norm, sau chuẩn hóa, tổng các giá trị của những phần tử trong vector bằng 1, với L2-norm, sau chuẩn hóa, độ dài vector bằng 1. Ghép các vector đặc trưng block sẽ thu được vector đặc trưng HOG cho ảnh. Số chiều vector đặc trưng ảnh tính theo công thức:

$$size_{image} = n * size_{block}$$

Trong đó:

n là số block trong ảnh $size_{block}$ là số chiều của vector đặc trưng block.

1.2.5 Tính toán vector đặc trưng HOG

- Với mỗi hình ảnh kích thước 64×128 , chia thành các block 16×16 chồng nhau, sẽ có 7 block ngang, 15 block dọc, nên sẽ có $7 \times 15 = 105$ block.
- Mỗi block có 4 cell, khi áp dụng biểu đồ 9 bin cho mỗi cell, mỗi block sẽ được đại diện bởi một vector có kích thước 36×1 .
- Vì vậy, khi nối tất cả các vector trong một block lại với nhau, ta sẽ thu được vector đặc trưng HOG của ảnh có kích thước $105 \times 36 \times 1 = 3780 \times 1$

Chương 2

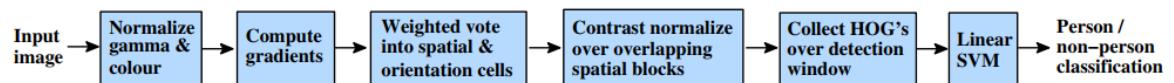
Histogram of Oriented Gradient cho bài toán phát hiện người đi bộ

2.1 Phát biểu bài toán

Phát hiện người đi bộ là vấn đề quan trọng trong nhiều bài toán ứng dụng của lĩnh vực xử lý ảnh, ví dụ như giám sát giao thông, phát hiện đột nhập, xe tự hành,... Để giải quyết bài toán, nhóm em đã tìm hiểu, nghiên cứu và thực hành sử dụng phương pháp HOG và mô hình SVM.

Toàn bộ mã nguồn cho báo cáo này lưu trữ tại:

<https://github.com/nducthang/Pedestrians-Detections-HOG>



Hình 2.1: Các bước xử lý bài toán

Chuẩn hóa màu và kích cỡ ảnh Trong bài báo cáo này, nhóm sử dụng ảnh xám để thực hiện. Ảnh đầu vào kích thước 64×128 ($w \times h$) pixel.

```

# Chuyển sang ảnh xám và resize ảnh
if len(img.shape) > 2:
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Lấy chiều cao và chiều rộng của ảnh
h, w = img.shape # 128, 64
if h != 128 or w != 64:
    img = cv2.resize(src=img, dsize=(64, 128))
    h, w = img.shape # 128, 64

```

Hình 2.2: Chuẩn hoá màu và kích cỡ ảnh

Tính Gradient của ảnh

```

""" TÍNH GRADIENT CỦA ẢNH """
x_kernel = np.array([[-1, 0, 1]])
y_kernel = np.array([[-1], [0], [1]])
dx = cv2.filter2D(img, cv2.CV_32F, x_kernel)
dy = cv2.filter2D(img, cv2.CV_32F, y_kernel)

# Độ lớn và hướng gradient của toàn bộ ảnh
maginutude = np.sqrt(np.square(dx) + np.square(dy))
orientation = np.arctan(np.divide(dy, dx + 0.00001)) # radient
orientation = np.degrees(orientation) # -90 -> 90
orientation += 90 # 0 -> 180

```

Hình 2.3: Tính gradient của ảnh

Vote hướng gradient từng cell vào histogram

```

""" VOTE VÀO HISTOGRAM """
num_cell_x = w // cell_size # 8
num_cell_y = h // cell_size # 16

# hist_tensor Lưu lại vote của từng cell
hist_tensor = np.zeros([num_cell_y, num_cell_x, bins])
for cx in range(num_cell_x):
    for cy in range(num_cell_y):
        ori = orientation[cy*cell_size:cy*cell_size +
                          cell_size, cx*cell_size:cx*cell_size+cell_size]
        mag = maginutude[cy*cell_size:cy*cell_size +
                          cell_size, cx*cell_size:cx*cell_size+cell_size]

        hist, _ = np.histogram(ori, bins=bins, range=(0, 180), weights=mag) # 1D vector, 9 elements
        hist_tensor[cy, cx, :] = hist

```

Hình 2.4: Vote hướng gradient từng cell vào histogram

Chuẩn hóa theo block

```

# Chuẩn hóa theo block
redundant_cell = block_size - 1 # số cell thừa
feature_tensor = np.zeros(
    [num_cell_y-redundant_cell, num_cell_x-redundant_cell, block_size*block_size*bins])
for by in range(num_cell_y-redundant_cell):
    for bx in range(num_cell_x-redundant_cell):
        by_from = by
        by_to = by + block_size
        bx_from = bx
        bx_to = bx + block_size
        v = hist_tensor[by_from:by_to, bx_from:bx_to, :].flatten()
        feature_tensor[by, bx, :] = v/(LA.norm(v, 2)+0.00001)
        # if np.isnan(feature_tensor[by, bx, :]).any():
        #     feature_tensor[by, bx, :] = v

# Vector đặc trưng đã chuẩn hóa
return feature_tensor.flatten() # 3780 feature

```

Hình 2.5: Chuẩn hóa theo block

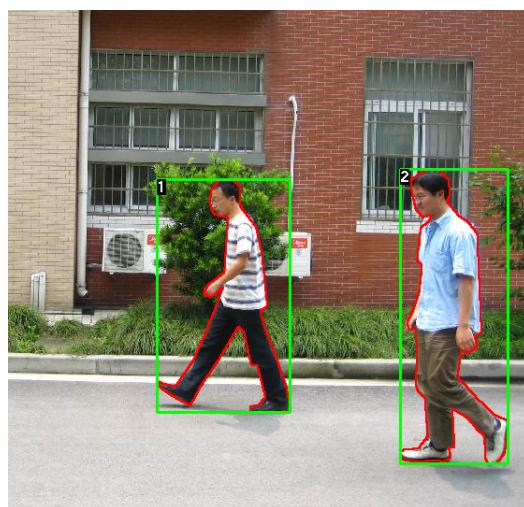
2.2 Mô tả dữ liệu

2.2.1 Dữ liệu người đi bộ Penn Fudan

Đây là cơ sở dữ liệu chứa các hình ảnh để phát hiện người đi bộ. Các hình ảnh được lấy từ các cảnh xung quanh khuôn viên trường và đường phố đô thị. Đối tượng quan tâm là người đi bộ, mỗi hình ảnh sẽ có ít nhất một người đi bộ trong đó.

Chiều cao của người đi bộ được gắn nhãn trong cơ sở dữ liệu này rơi vào [180.390] pixel. Tất cả những người đi bộ được dán nhãn đều ở dạng hình chữ nhật đúng.

Có 170 hình ảnh với 345 người đi bộ được dán nhãn, trong đó 96 hình ảnh được chụp từ xung quanh Đại học Pennsylvania, và 74 hình khác được chụp từ xung quanh Đại học Fudan.



Hình 2.6: Một mẫu dữ liệu người đi bộ trong bộ dữ liệu Penn Fudan

Nguồn: https://www.cis.upenn.edu/~jshi/ped_html/

2.2.2 Bộ dữ liệu INRIA Person

Bộ dữ liệu này được thu thập như một phần của công trình nghiên cứu về việc phát hiện người đứng thẳng trong hình ảnh và video. Do **đa phần** người trong các hình ảnh bộ dữ liệu này đang ở trạng thái đi bộ nên ta có thể tạm dùng bộ dữ liệu này để thực hiện cho bài toán.

Tập dữ liệu chứa hình ảnh từ một số nguồn khác nhau:

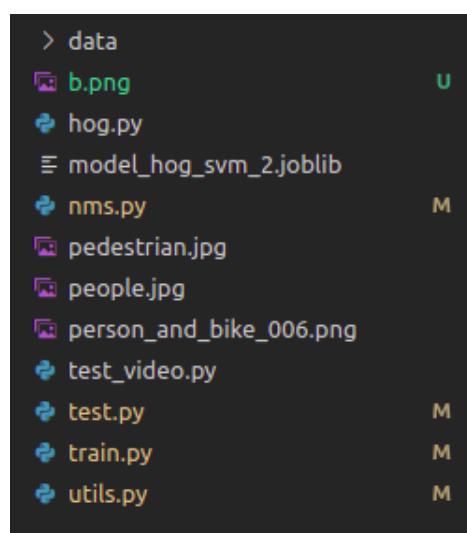
- Hình ảnh từ bộ dữ liệu GRAZ 01
- Hình ảnh từ bộ sưu tập hình ảnh kỹ thuật số cá nhân được chụp trong một khoảng thời gian dài.
- Rất ít hình ảnh được lấy từ web bằng hình ảnh google.

Chỉ những người thẳng đứng (chiều cao người > 100) mới được đánh dấu trong mỗi hình ảnh.

Nguồn: <http://pascal.inrialpes.fr/data/human/>

2.3 Kết quả thực nghiệm

2.3.1 Cấu trúc các file mã nguồn



Hình 2.7: Cấu trúc mã nguồn

Các file/thư mục gồm:

data: Chứa dữ liệu cho huấn luyện.

hog.py: File code thuật toán HOG để trích xuất đặc trưng.

nms.py: File code thuật toán Non-maximum suppression để loại bỏ các box overlap cao.

test.py: File code kiểm thử mô hình, dự đoán 1 ảnh từ mô hình đã huấn luyện.

train.py: File code huấn luyện mô hình.

utils.py: File chứa một số hàm hỗ trợ như đọc dữ liệu, predict của SVM, ...

model_hog_svm_2.joblib: Trọng số của mô hình SVM sau khi huấn luyện, file này được load lại cho mục đích dự đoán.

2.3.2 Huấn luyện

Sử dụng 2 bộ dữ liệu đã giới thiệu để tách các mẫu thành positive (người đi bộ) và negative (không phải người đi bộ). Sau đó trích rút đặc trưng HOG tương ứng với các mẫu từ dữ liệu, đầu ra với mỗi ảnh của HOG là một vector 1D có 3780 phần tử. Với các mẫu là người đi bộ, đặt là nhãn 1, và ngược lại đặt là 0. Sau đó vector đặc trưng và vector nhãn được đưa qua mô hình SVM để huấn luyện phân lớp, file weight của mô hình được lưu lại phục vụ cho việc kiểm thử.

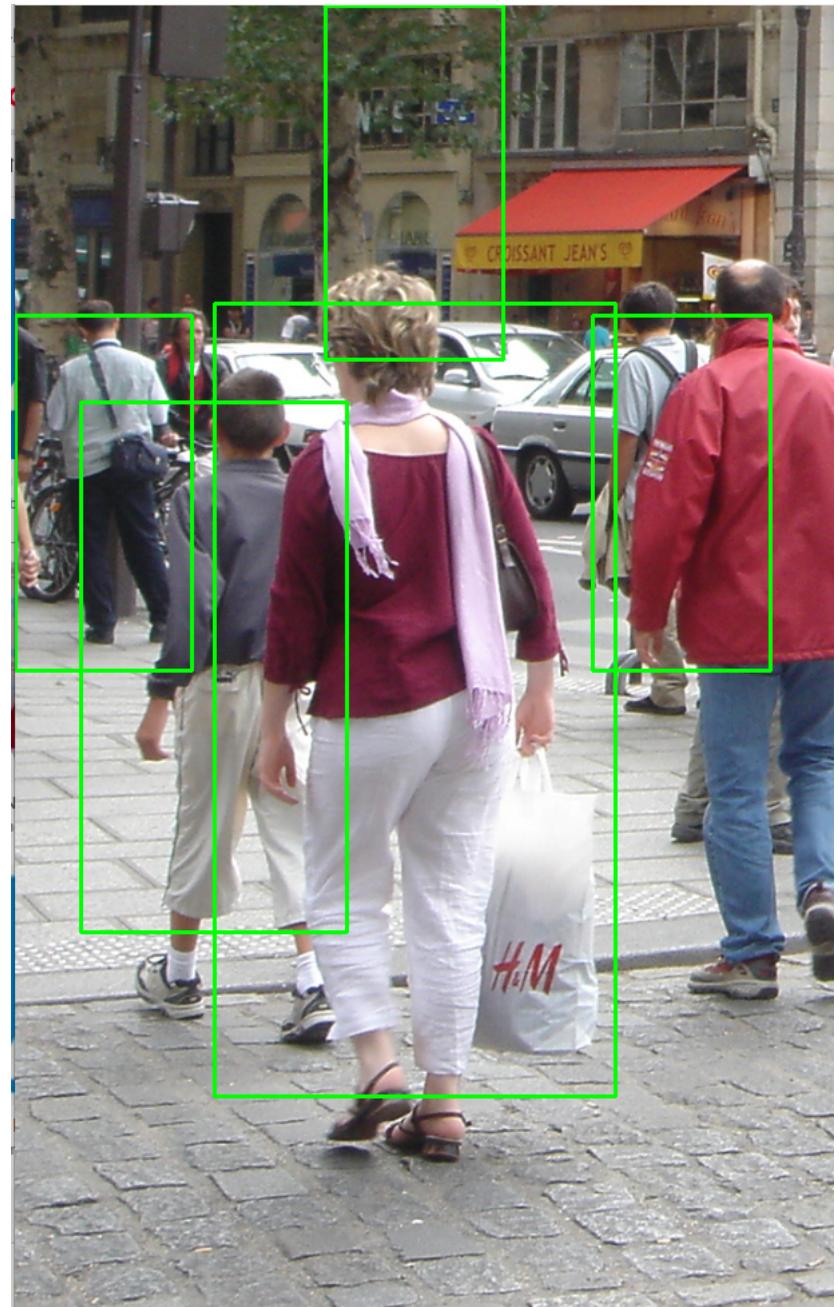
2.3.3 Kiểm thử

Với mỗi ảnh, quá trình dự đoán như sau:

1. Load model đã huấn luyện cho dự đoán
2. Dùng sliding window để lấy ra các box với kích cỡ khác nhau của ảnh.
3. Các box được đưa qua HOG để biến đổi thành vector đặc trưng. Các vector đặc trưng này sau đó được đưa qua mô hình đã huấn luyện cho phân lớp để dự đoán. Nếu xác suất dự đoán người đi bộ của vector đó lớn hơn ngưỡng đặt ra (Threshold) thì box đó được lưu giữ lại. Mỗi box lưu toạ độ góc trên trái nhất và góc dưới phải nhất.
4. Sử dụng Non-maximum suppression để loại bỏ các box mà chồng lấn nhau lớn hơn ngưỡng đặt ra (overlapThreshold).
5. Vẽ các box còn lại lên ảnh và hiển thị kết quả.



Hình 2.8: Kết quả dự đoán của mô hình cho 1 mẫu ảnh



Hình 2.9: Kết quả dự đoán của mô hình cho 1 mẫu ảnh

KẾT LUẬN

Trong quá trình làm báo cáo, nhóm em đã tìm hiểu được nhiều về các kỹ thuật cũng như các mô hình trong thị giác máy tính. Nhóm em đã triển khai được HOG cho trích xuất người đi bộ theo đúng yêu cầu của đề tài và paper gốc. Kết quả đạt tương đối. Về mô hình, nhóm em nhận xét như sau:

Ưu điểm:

- Thuật toán đơn giản, dễ triển khai.
- Nhận dạng được người đi bộ ở mức khá.

Nhược điểm:

- Vẫn sai sót ở nhiều mẫu ảnh,
- Chương trình chạy chậm, chưa thích hợp với các dự án thực tế đòi hỏi thời gian thực.

Chương trình đang sử dụng một trình phân lớp đơn giản là SVM, vì thời gian hạn hẹp nên nhóm chưa kịp thử nghiệm với các trình phân lớp khác cũng như những phương pháp cải tiến khác cho bài toán. Sở dĩ, nhiều mẫu ảnh sai sót là do dữ liệu huấn luyện chưa đủ lớn, các dữ liệu giới thiệu trong báo cáo đều rất ít và chưa đủ để làm cho mô hình thực sự tốt được. Ngoài ra, các tham số trong mô hình cũng quyết định lớn đến kết quả đầu ra, chúng ta phải dựa vào kinh nghiệm để chọn ngưỡng và các tham số tốt cho bài toán, như là các ngưỡng loại bỏ overlap của Non-maximum suppression hay ngưỡng box được chọn khi dự đoán, ... Việc chọn các tham số này đòi hỏi có kinh nghiệm và nhiều thời gian kiểm thử.

Chương trình chưa thích hợp với dự án thực tế vì chậm, lý do chậm chủ yếu là do bước kiểm thử sử dụng sliding window để tách ra các box cho mô hình. Việc dùng sliding window ngoài làm chương trình chậm thì nếu không code tốt và chọn kích cỡ, bước xайд window hợp lý còn có thể gây ra sự sai lệch cho mô hình. Vì không sử dụng thư viện hỗ trợ và tự xây dựng code nên nhóm đã mất khá nhiều thời gian để thử nghiệm tại bước này để chọn được tỷ lệ ảnh cũng như kích cỡ box hợp lý cho việc sliding window.

Hiện nay, các mô hình Deep learning hiện đại đều có khả năng trích xuất đặc trưng tốt hơn HOG. Tuy nhiên, không vì thế mà những mô hình cổ điển bị lãng quên, đó là những nền móng, kiến thức cơ sở cho thị giác máy tính. Với mục đích tìm hiểu, chúng em đã hoàn thiện bài toán. Tuy nhiên, vì thời gian hạn hẹp, nên nhóm còn nhiều ý tưởng chưa được triển khai để cải tiến mô hình tốt hơn. Trong tương lai, nhóm sẽ thử nghiệm thêm các tham số, bổ sung dữ liệu, thay thế sliding window bằng các mô hình hiện đại và nhanh hơn như cách mà các mô hình Fast-RCNN hoặc Faster-RCNN thực

hiện để chọn box, thử nghiệm và so sánh với các trình phân lớp khác như AdaBoost, Decision tree, Random Forest, hoặc các mạng phân lớp của Deep learning ...

Chân thành cảm ơn TS. Nguyễn Thị Oanh đã cung cấp các kiến thức về môn học thị giác máy tính cho nhóm để nhóm hoàn thiện được đề tài này. Đề tài còn nhiều sai sót, mong nhận được sự góp ý của cô và các bạn.

Tài liệu tham khảo

- [1] R. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation", In CVPR, 2014.
- [2] R. Girshick. "Fast R-CNN". arXiv:1504.08083, 2015.
- [3] S. Ren, K. He, R. Girshick, and J. Sun. "Faster R-CNN: Towards real-time object detection with region proposal networks", In NIPS, 2015.
- [4] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. "Selective search for object recognition", IJCV, 2013.
- [5] K. He, X. Zhang, S. Ren, and J. Sun. "Spatial pyramid pooling in deep convolutional networks for visual recognition", In ECCV. 2014.
- [6] Aston Zhang, Zack C. Lipton, Mu Li. Alex J. Smola, *Dive into Deep Learning*, d2l.ai, 2019.
- [7] Adrian Rosebrock, *Deep learning for Computer vision with Python*, pyimagesearch, 2017.
- [8] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In CVPR, 2005.
- [9] http://vision.stanford.edu/teaching/cs131_fall2021/index.html
- [10] <https://viblo.asia/p/tim-hieu-ve-phuong-phap-mo-ta-dac-trung-hog-histogram-o>
- [11] <https://www.learnopencv.com/histogram-of-oriented-gradients/>
- [12] <https://minhng.info/tutorials/histograms-of-oriented-gradients.html>
- [13] <https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/>
- [14] <https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-with-python-and-opencv/>