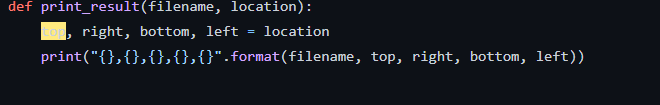
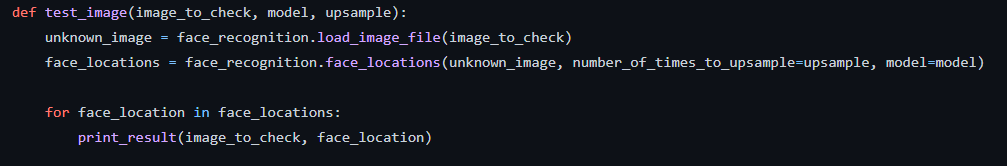
Lê Minh Trung – Face Detection

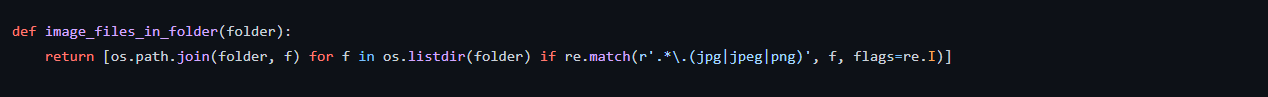
* Thư viện face\_recognition.api có thể sử dụng hệ thống Nhận dạng khuôn mặt đã được tạo làm API REST. Các api thư mục chứa một đơn giản Flask API cung cấp lối vào cho hệ thống nhận diện khuôn mặt.



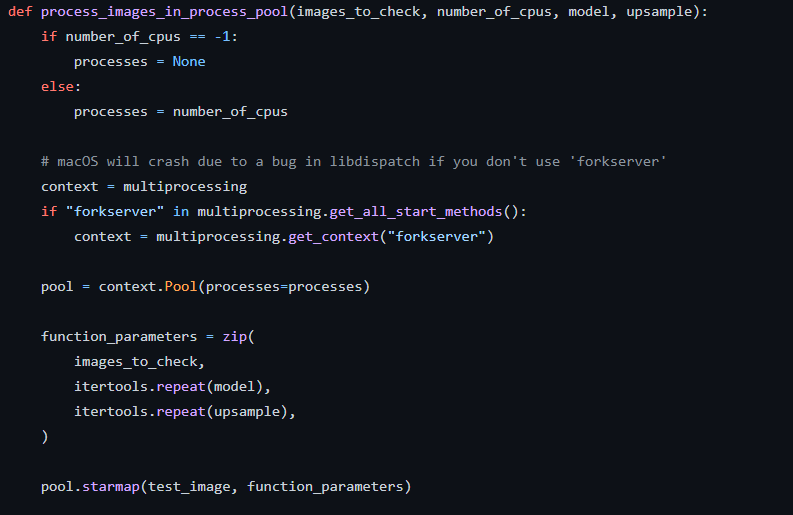
* Kết quả in ra tên của khuôn mặt nhận dạng và vị trí trên, dưới, trái phải của từng khuôn mặt trong bức hình.



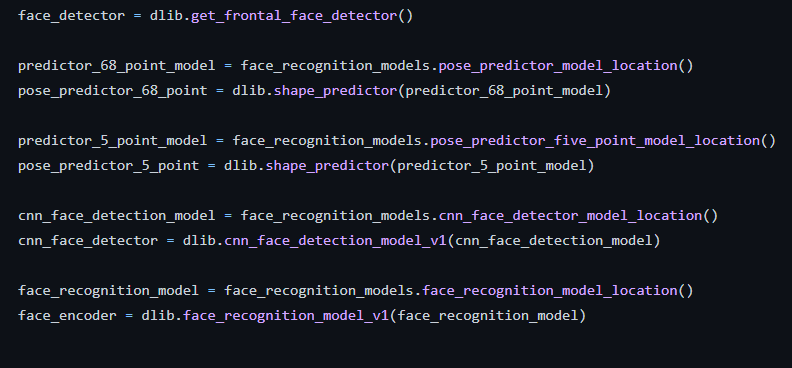
* API modul ‘face\_recognition’, tải hình ảnh vào một mảng numpy. Nếu đã có hình ảnh trong mảng numpy, ta có thể bỏ qua bước này. Sau đó, ta có thể thực hiện các thao tác trên hình ảnh, như tìm tất cả khuôn mặt trong hình ảnh, xác định các đặc điểm trên khuôn mặt hoặc có thể nhận mã hóa khuôn mặt cho từng khuôn mặt trong hình ảnh.



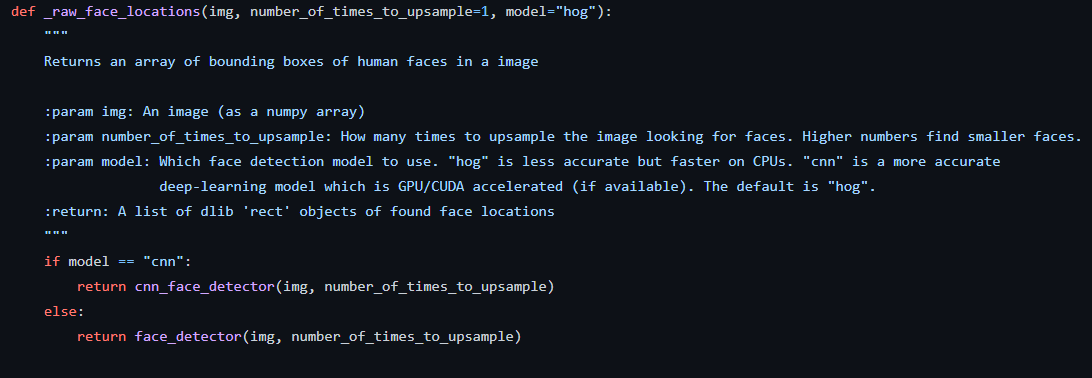
* Thư mục chứa các khuôn mặt sau khi nhận dạng được cắt bớt



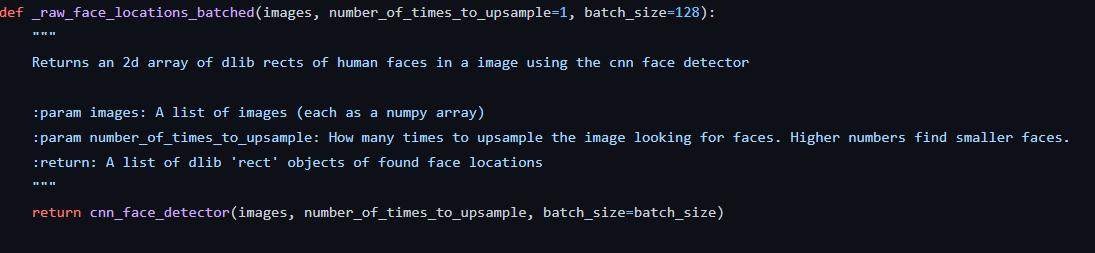
* Hàm xử lý ảnh đa tiến trình, đối với macOS sẽ gặp sự cố do lỗi trong libdispatch nếu không sử dụng 'forkserver'



* Sử dụng thư viện Dlib được gọi get\_frontal\_face\_detector() khá trực quan. Chức năng này chỉ hoạt động với hình ảnh thang độ xám, vì vậy sẽ phải thực hiện điều đó trước tiên với OpenCV.
* Hàm get\_frontal\_face\_detector() sẽ trả về một detector hàm mà chúng ta có thể sử dụng để truy xuất thông tin khuôn mặt. Mỗi khuôn mặt là một đối tượng chứa các điểm mà ảnh có thể được tìm thấy.
* DLib cung cấp một chức năng được gọi là shape\_predictor(). Đối tượng này là một công cụ lấy vùng ảnh có chứa đối tượng nào đó và xuất ra một tập hợp các vị trí điểm xác định tư thế của đối tượng. Ví dụ cổ điển về điều này là dự đoán tư thế khuôn mặt người, trong đó bạn lấy hình ảnh khuôn mặt người làm đầu vào và dự kiến ​​xác định vị trí của các điểm mốc quan trọng trên khuôn mặt như khóe miệng và mắt, đầu mũi, v.v. ra ngoài.



* Trả về một mảng các hộp giới hạn khuôn mặt người trong một hình ảnh
* param img: Một hình ảnh (dưới dạng một mảng numpy)
* param number\_of\_times\_to\_upsample: Số lần lấy mẫu hình ảnh tìm kiếm khuôn mặt. Số cao hơn tìm thấy khuôn mặt nhỏ hơn.
* param model: Sử dụng mô hình nhận diện khuôn mặt nào. "hog" kém chính xác hơn nhưng nhanh hơn trên CPU. "cnn" chính xác hơn mô hình học sâu được tăng tốc GPU / CUDA (nếu có). Giá trị mặc định là "hog".
* Trả về một danh sách các đối tượng dlib 'rect' của các vị trí khuôn mặt được tìm thấy



* Trả về mảng 2d các biểu hiện dlib của khuôn mặt người trong một hình ảnh bằng cách sử dụng công cụ dò tìm khuôn mặt cnn
* param images: Một danh sách các hình ảnh (mỗi hình ảnh là một mảng numpy)
* param number\_of\_times\_to\_upsample: Số lần lấy mẫu hình ảnh tìm kiếm khuôn mặt. Số cao hơn tìm thấy khuôn mặt nhỏ hơn.
* Trả về một danh sách các đối tượng dlib 'rect' của các vị trí khuôn mặt được tìm thấy

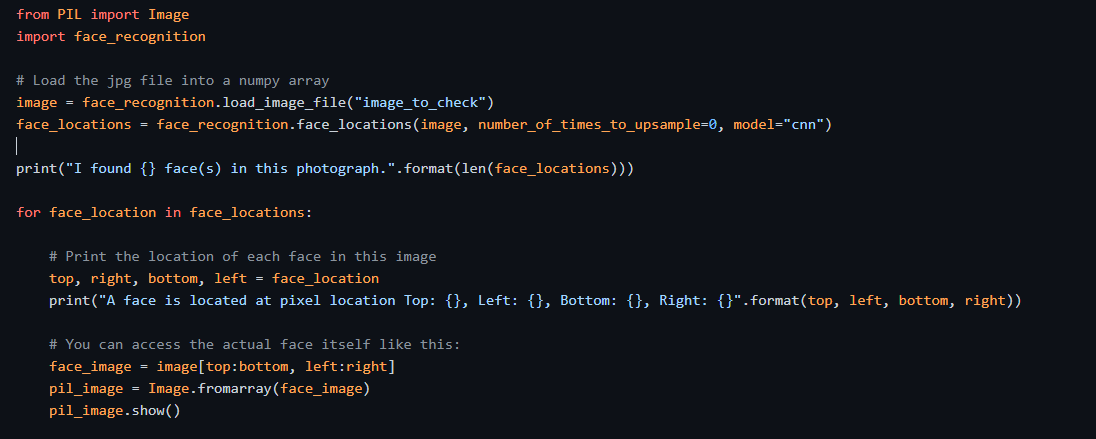
***Face detection***



* Tìm tất cả các khuôn mặt trong hình ảnh bằng cách sử dụng mô hình dựa trên HOG mặc định.
* Phương pháp này khá chính xác, nhưng không chính xác bằng mô hình CNN

và không được tăng tốc GPU.

***Face detection use CNN***



* Tìm tất cả các khuôn mặt trong hình ảnh bằng cách sử dụng mạng nơ-ron phức hợp được đào tạo trước.
* Phương pháp này chính xác hơn mô hình HOG mặc định, nhưng nó chậm hơn trừ khi bạn có GPU nvidia và dlib được biên dịch với các phần mở rộng CUDA. Nhưng nếu bạn làm vậy, điều này sẽ sử dụng tăng tốc GPU và hoạt động tốt.

**Face detection sử dụng thuật toán HOG (Histrogram of oriented gradient) trong Dlib**

1. **Thuật toán HOG**

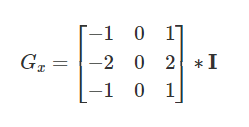
Hình dạng của một vật thể được mô tả thông qua hai ma trận đó là ma trận độ lớn gradient (gradient magnitude) và ma trận phương gradient (gradient direction). Đầu tiên hình ảnh được chia thành 1 lưới ô vuông và trên đó ta xác định rất nhiều các vùng cục bộ liền kề hoặc chồng lấn lên nhau. Các vùng này tương tự như những vùng hình ảnh cục bộ mà ta tính tích chập trong thuật toán CNN. Một vùng cục bộ bao gồm nhiều ô cục bộ (trong thuật toán HOG là 4) có kích thước là 8x8 pixels. Sau đó, một biểu đồ histogram thống kê độ lớn gradient được tính toán trên mỗi ô cục bộ mà chúng ta sẽ tìm hiểu ở phần 2.1 cách thức tính. Bộ mô tả HOG (HOG descriptor) được tạo thành bằng cách nối liền (concatenate) 4 véc tơ histogram ứng với mỗi ô thành một véc tơ tổng hợp. Để cải thiện độ chính xác, mỗi giá trị của véc tơ histogram trên vùng cục bộ sẽ được chuẩn hóa theo norm chuẩn bậc 2 hoặc bậc 1. Phép chuẩn hóa này nhằm tạo ra sự bất biến tốt hơn đối với những thay đổi trong chiếu sáng và đổ bóng.

1. **Tính toán gradient**

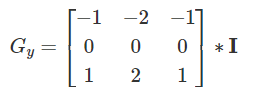
Bước đầu tiên là tiền xử lý dữ liệu ảnh (pre-processing image). Ta sẽ cần chuẩn hóa màu sắc và giá trị gamma. Bước này có thể được bỏ qua trong phần tính toán bộ mô tả HOG, vì việc chuẩn hóa bộ mô tả ở bước tiếp theo đã đạt được kết quả tương tự. Thay vào đó, tại bước đầu tiên của tính toán bộ mô tả chúng ta sẽ tính các giá trị gradient. Phương pháp phổ biến nhất là áp dụng một mặt nạ đạo hàm rời rạc (discrete derivative mask) theo một hoặc cả hai chiều ngang và dọc. Cụ thể, phương pháp sẽ lọc ma trận cường độ ảnh với các bộ lọc như [Sobel mask](https://en.wikipedia.org/wiki/Sobel_operator) hoặc scharr.

Để tính bộ lọc sobel, phép tích chập của ma trận kernel kích thước 3x3 được thực hiện với hình ảnh ban đầu. Nếu chúng ta kí hiệu I là ma trận ảnh gốc và Gx, Gy là 2 ma trận ảnh mà mỗi điểm trên nó lần lượt là đạo hàm theo trục x trục y. Chúng ta có thể tính toán được:

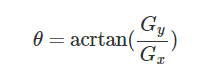
* Đạo hàm theo chiều dọc:

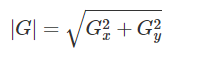


* Đạo hàm theo chiều ngang:



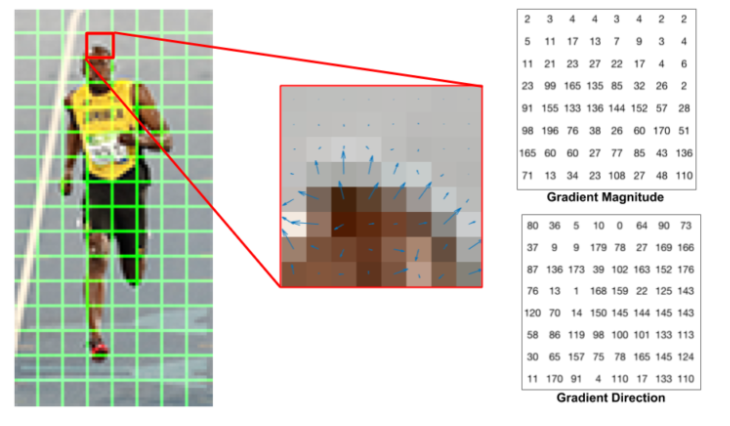
* **gradient direction**: Phương gradient. Là độ lớn góc giữa véc tơ gradient x và y giúp xác định phương thay đổi cường độ màu sắc hay chính là phương đổ bóng của hình ảnh. Giả sử Gx, Gy lần lượt là giá trị gradient theo lần lượt phương x và y của hình ảnh.



* **gradient magnitude**: Độ lớn gradient. Là chiều dài của véc tơ gradient theo phương x và phương y. Biểu diễn phân phối histogram của véc tơ này theo véc tơ phương gradient sẽ thu được véc tơ mô tả đặc trưng HOG.
* 

1. **Các bước tính HOG**

Hình ảnh được chia thành một lưới ô vuông mà mỗi một ô có kích thước 8x8 pixels. Có tổng cộng 64 ô pixels tương ứng với mỗi ô. Trên mỗi một ô trong 64 pixels ta cần tính 2 tham số đó là độ lớn gradient (gradient magnitute) và phương gradient (gradient direction). Vậy có tổng cộng 8x8x2 = 128 giá trị cần tính bao gồm 64 giá trị gradient magnitute và 64 giá trị gradient direction như ma trận hình bên dưới:



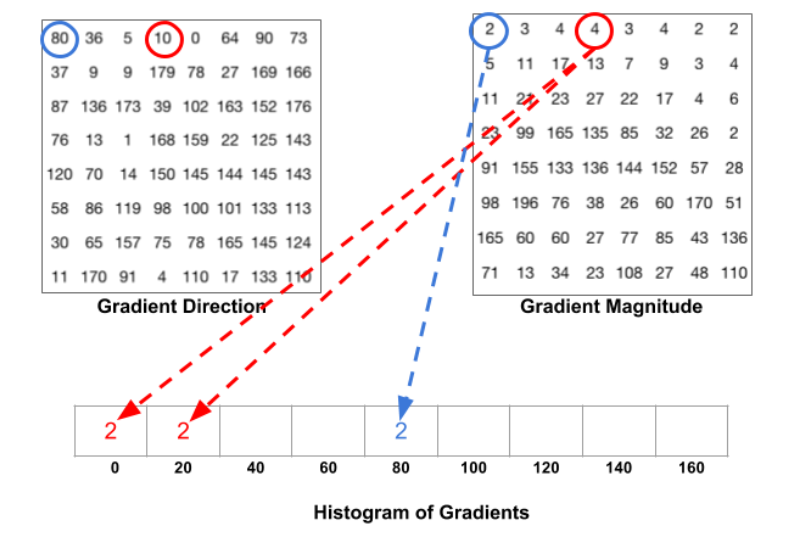
* **Véc tơ histogram sẽ được tạo ra:**

**Bước 1:** Mapping độ lớn gradient vào các bins tương ứng của phương gradient.

Sắp xếp các giá trị phương gradient theo thứ tự từ nhỏ đến lớn và chia chúng vào 9 bins. Độ lớn của phương gradient sẽ nằm trong khoảng [0, 180] nên mỗi bins sẽ có độ dài là 20 như hình bên dưới.

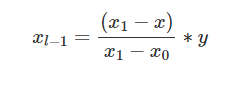
Mỗi một phương gradient sẽ ghép cặp với một độ lớn gradient ở cùng vị trí tọa độ. Khi biết được phương gradient thuộc bins nào trong véc tơ bins, ta sẽ điền vào giá trị giá trị của độ lớn gradient vào chính bin đó.

Hình bên dưới ô được khoanh trong hình tròn viền xanh tương ứng với phương gradient là 80 và độ lớn gradient là 2. Khi đó tại véc tơ bins của HOG, phương gradient bằng 80 sẽ rơi vào vị trí thứ 5 nên tại ô này chúng ta điền giá trị 2 ứng với độ lớn gradient.

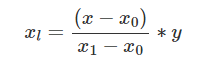


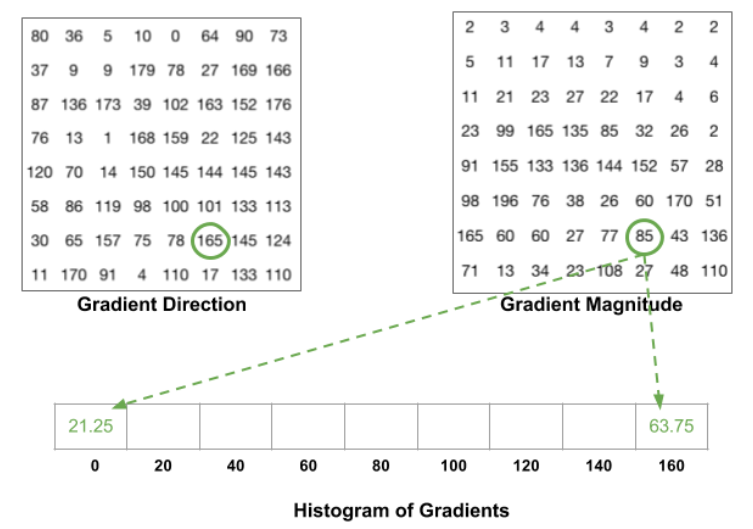
Đầu mút là các giá trị chia hết cho độ rộng của một bin (chẳng hạn 0, 20, 40,… là những đầu mút bin). Trong trường hợp độ lớn phương gradients không rơi vào các đầu mút, ta sẽ sử dụng linear interpolation để phân chia độ lớn gradient về 2 bins liền kề mà giá trị phương gradient rơi vào. Ví dụ: giá trị phương gradient bằng x ghép cặp với độ lớn gradient bằng y. x∈[x0,x1] tức là phương gradients rơi vào khoảng giữa bin thứ (l−1) và bin thứ l: . Khi đó tại 2 bins (l−1) và l được điền vào giá trị cường độ theo công thức interpolation:

* Gía trị tại bins l−1:



* Gía trị tại bins l:





* Với điểm được khoanh tròn bởi hình tròn màu xanh có phương gradient bằng 165 và độ lớn gradient bằng 85. Ta phân chia giá trị về các bins 0 (hoặc 180) và 160 các giá trị theo công thức interpolation bên trên. Kết quả cuối cùng chúng ta thu được là:

**Bước 2:** Chuẩn hóa véc tơ histogram theo block 16x16

Véc tơ histogram sẽ bị phụ thuộc vào cường độ các pixels của một bức ảnh. Với 2 bức ảnh có cùng nội dung nhưng bức ảnh biến thể tối hơn được tạo thành từ ma trận ảnh gốc nhân 1/2. Khi đó giá trị véc tơ histogram của ảnh gốc cũng sẽ gấp đôi véc tơ histogram của ảnh biến thể. Chính vì thế cần chuẩn hóa véc tơ histogram để cả 2 bức ảnh có cùng một véc tơ biểu diễn.

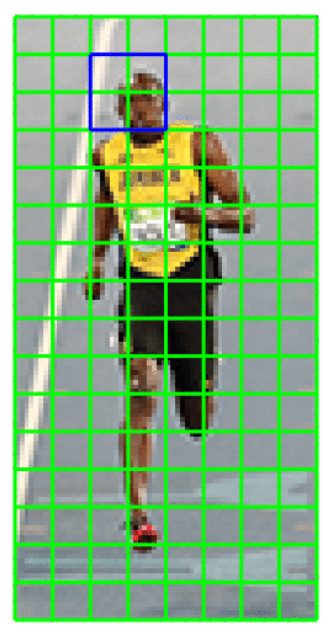
Chuẩn hóa norm chuẩn bậc 2:



Ngoài ra ta cũng có thể sử dụng norm chuẩn bậc 1.

Trong đó h là véc tơ histogram của các gradient

Qúa trình chuẩn hóa sẽ thực hiện trên một block kích thước 2x2 trên lưới ô vuông ban đầu (mỗi ô kích thước 8x8 pixel). Như vậy chúng ta sẽ có 4 véc tơ histogram kích thước 1x9, concatenate các véc tơ sẽ thu được véc tơ histogram tổng hợp kích thước là 1x36 và sau đó chuẩn hóa theo norm chuẩn bậc 2 trên véc tơ này. Việc di chuyển các window thực hiện tương tự như phép tích chập 2 chiều trong mạng CNN với step\_size = 8 pixels như hình ảnh bên dưới:



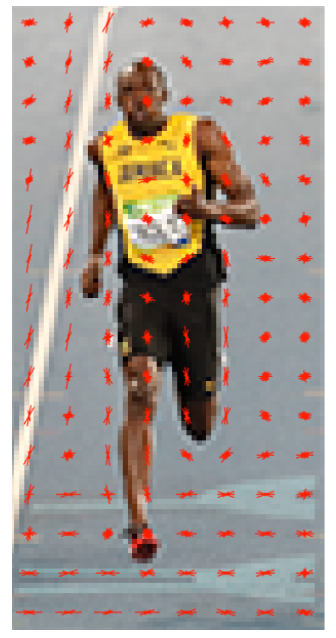
**Bước 3:** Tính toán HOG feature véc tơ.

Sau khi chuẩn hóa các véc tơ histogram, chúng ta sẽ concatenate các véc tơ 1x36 này thành một véc tơ lớn. Đây chính là véc tơ HOG đại diện cho toàn bộ hình ảnh.

Ví dụ: Hình ảnh được chia thành lưới ô vuông kích thước 16x8 (mỗi ô 8x8). Qúa trình tính toán HOG sẽ di chuyển 7 lượt theo chiều rộng và 15 lượt theo chiều cao. Như vậy sẽ có tổng cộng 7x15=105 patches, mỗi patch tương ứng với 1 véc tơ histograms 36 chiều. Do đó cuối cùng véc tơ HOG sẽ có kích thước là 105x36=3780 chiều. Đây là một véc tơ kích thước tương đối lớn nên có thể mô phỏng được đặc trưng của ảnh khá tốt.

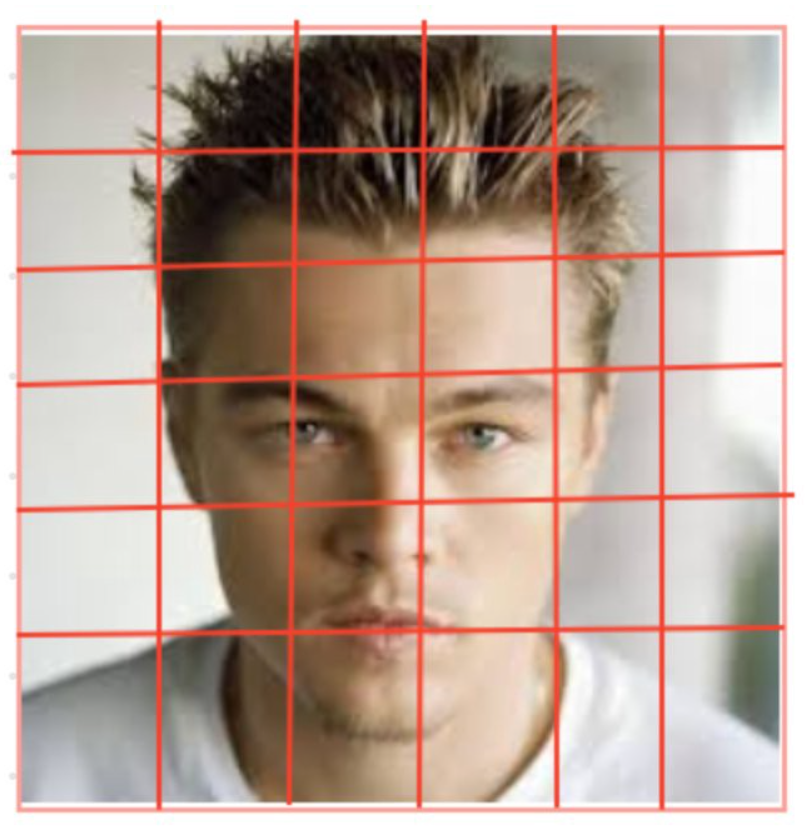
**Biểu diễn phân phối HOG trên ảnh**

Đối với mỗi một ô trên lưới ô vuông, chúng ta biểu diễn phân phối HOG bao gồm nhóm 9 véc tơ chung gốc chiều dài bằng độ lớn gradient và góc bằng phương gradient. Khi đó chiều của nhóm các véc tơ sẽ tương đối giống với dáng của vận động viên trong ảnh, đặc biệt là tại các vị trí chân và tay. Cụ thể hãy xem hình bên dưới:



1. **Face detection**

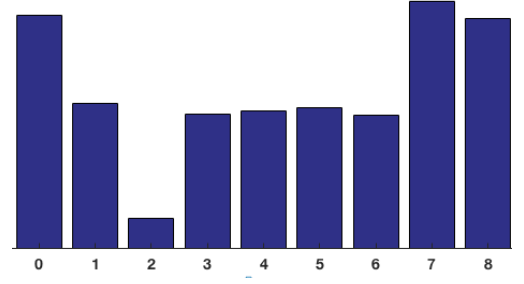
**Bước 1:** Ý tưởng cơ bản của HOG là chia hình ảnh thành các ô vuông nhỏ



**Bước 2:** Tính toán biểu đồ cho mỗi ô

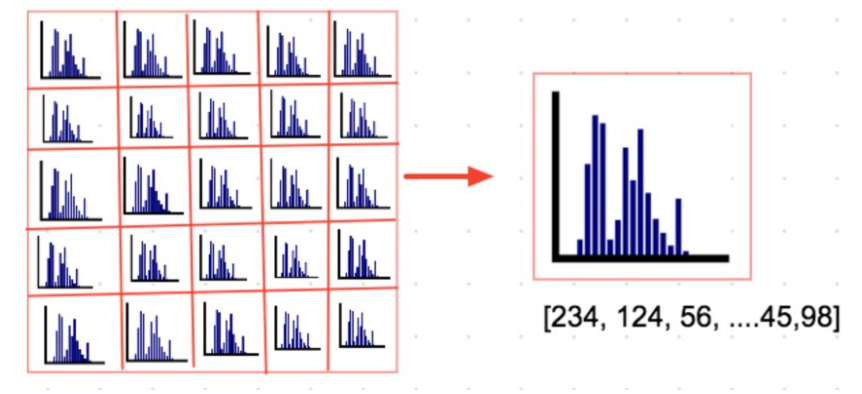


* Tính tổng tất cả các độ lớn gradient thuộc cùng 1 bins của véc tơ bins ta thu được biểu đồ Histogram of Gradients như bên dưới:



* Biểu đồ Histogram of Gradient gồm 9 bins tương ứng với một ô vuông trong lưới ô vuông.

**Bước 3:** Tập hợp tất cả các biểu đồ lại với nhau để tạo thành vectơ đặc trưng, ​​tức là nó tạo thành một biểu đồ từ tất cả các biểu đồ nhỏ duy nhất cho mỗi khuôn mặt



Kết hợp các biểu đồ nhỏ thành một biểu đồ là một vectơ đặc điểm cuối cùng

* Ngoài ra thư viện dlib có một phương pháp chuyển tiếp thẳng để trả về trình dò ​​tìm khuôn mặt HOG “dlib.get\_frontal\_face\_detector ()”



Tài liệu:

1. <https://livecodestream.dev/post/detecting-face-features-with-python/>
2. [http://dlib.net/python/index.html#dlib.shape\_predictor](http://dlib.net/python/index.html%23dlib.shape_predictor)
3. <https://livecodestream.dev/post/detecting-face-features-with-python/>
4. [https://phamdinhkhanh.github.io/2019/11/22/HOG.html#212-c%C3%A1c-b%C6%B0%E1%BB%9Bc-t%C3%ADnh-hog](https://phamdinhkhanh.github.io/2019/11/22/HOG.html%23212-c%C3%A1c-b%C6%B0%E1%BB%9Bc-t%C3%ADnh-hog)
5. <https://github.com/AyushExel/ml4face-detection/blob/master/Final-Face%20Detection%20Notebook.ipynb>
6. <https://www.eeweb.com/real-time-face-detection-and-recognition-with-svm-and-hog-features/>