Histograms of Oriented Gradients

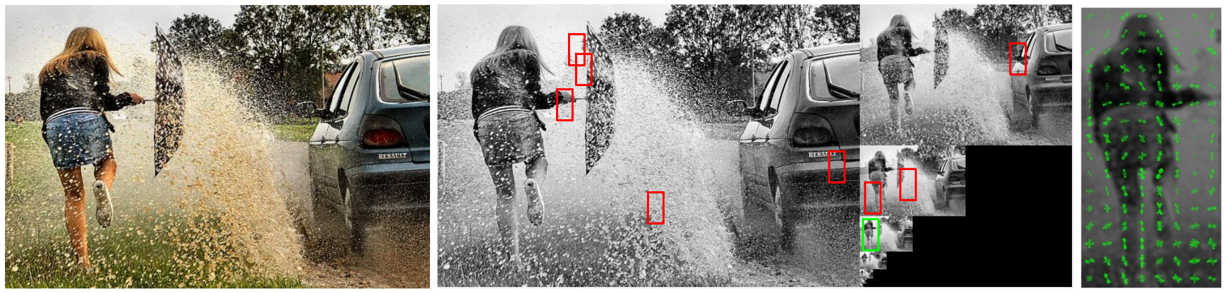
Carlo Tomasi

January 30, 2019

A useful question to ask of an image is whether it contains one or more instances of a certain object: a person, a face, a car, and so forth. Algorithms that answer this question are called *object detectors*. They often work by asking the same question in turn of every possible rectangle in the image that might possibly tightly bound one of the instances of interest.

For instance, when detecting a pedestrian walking by a camera, one could start with a *window* that is 128 pixels tall and 64 pixels wide: The 2 to 1 aspect ratio of such a rectangle is a rough compromise between the aspect ratio of a person viewed from the front and one viewed from the side with legs fully extended during a step. The actual size of the window—using multiples of 64 is a mere convenience—reflects the assumption that if the image of the person is significantly smaller than that then resolution might be insufficient to detect it reliably, so it is not even worth trying. The person could of course occupy a bigger part of the image. Because of this, one analyzes not only the original image, but also those in a pyramid of images: A 128 by 64 rectangle at a coarser level of the pyramid corresponds to a larger rectangle in the original image (see Figure 1).

The object detector then slides such a window over every image of the pyramid, perhaps in increments of a few pixels—the increment is called the *stride* of the detector—and computes a *feature* at every position, that is, a vector of numbers that describes the window's contents. The feature is fed to the classifier, which tells whether the window contains a pedestrian. The answer is likely to be positive for a set of windows that overlap the person's image, so the detector then chooses a single pixel to represent each connected set of positive classifier outputs. This scheme generalizes the correlation approach we saw in an earlier note: In that instance, the feature is a single number (the value of normalized cross-correlation between template and image at a particular pixel location) and the classifier compares that number against a fixed threshold.



[Input image from http://www.bewiser.co.uk/news/car-insurance/crackdown-rude-drivers-who-undertake-hog-lanes-and-splash-pedestrians]

Figure 1: A pedestrian detector slides a 128×64 window everywhere over a gray-scale image pyramid (middle; a few windows are shown in red) for the given input image (left) and runs a pedestrian/non-pedestrian classifier on every window. A good classifier will detect the pedestrian (green rectangle) if the pyramid is sampled finely enough in scale. The HOG feature (right) reports the distribution of the image gradient—suitably quantized and normalized—in each of 16 by 8 square cells in the target window.

This note describes one way to construct a feature vector from a fixed-size window [1]. The feature is specifically tuned to pedestrians. This is because humans are important subjects in imagery, and is therefore natural that the detection of humans in still images and video has drawn much attention in the literature.

A good feature makes the classifier's job as easy as possible: It removes what is irrelevant to classification, such as the colors of clothing, shadows, and whether the person wears a hat, but keeps all that distinguishes a person from something else. A perfect job at this task is of course wishful thinking. Instead, the design of the feature being described next starts from the following consideration by its authors:

> [L]ocal object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions, even without precise knowledge of the corresponding gradient or edge positions. In practice this is implemented by dividing the image window into small spatial regions ("cells"), for each cell accumulating a local 1-D histogram of gradient directions or edge orientations over the pixels of the cell. The combined histogram entries form the representation. For better invariance to illumination, shadowing, etc., it is also useful to contrast-normalize the local responses before using them. This can be done by accumulating a measure of local histogram "energy" over somewhat larger spatial regions ("blocks") and using the results to normalize all of the cells in the block. We will refer to the normalized descriptor blocks as *Histogram of Oriented Gradient (HOG)* descriptors [1].

This description is now fleshed out using actual parameter values from the paper quoted above, keeping in mind that different imaging situations or objects may call for different values. Also, the original paper gives only a high-level description, so some details may vary. The input is assumed to be a window I from a gray-level image, possibly from a pyramid.

Gradient. Approximate the two components I_x and I_y of the gradient of I by central differences[1]:

$$I_x(r,c) = I(r,c+1) - I(r,c-1) \quad \text{and} \quad I_y(r,c) = I(r-1,c) - I(r+1,c) \; .$$

While convolution with the derivative of a Gaussian yields more accurate derivatives in the presence of noise, the smoothing that this convolution entails would remove useful detail. In addition, some degree of noise averaging will occur when histograms are computed in later steps of the HOG computation, so Gaussian smoothing is both less beneficial and unnecessarily expensive.

The gradient is then transformed to polar coordinates, with the angle constrained to be between 0 and 180 degrees, so that gradients that point in opposite directions have the same angle:

$$\mu = \sqrt{I_x^2 + I_y^2} \quad \text{and} \quad \theta = \frac{180}{\pi}\left(\tan_2^{-1}(I_y, I_x) \mod \pi\right)$$

where $\tan_2^{-1}$ is the four-quadrant inverse tangent, which yields values between $-\pi$ and π.

Cell Orientation Histograms. Divide the window into adjacent, non-overlapping *cells* of size $C \times C$ pixels ($C = 8$). In each cell, compute a histogram of the gradient orientations binned into B bins ($B = 9$).

With so few bins, a pixel whose orientation is close to a bin boundary might end up contributing to a different bin, were the image to change slightly. To prevent these quantization artifacts, each pixel in a cell contributes to two adjacent bins (modulo B) a fraction of the pixel's gradient magnitude μ that decreases linearly with the distance of that pixel's gradient orientation from the two bin centers.

[1]Central difference requires division by 2, but this constant can be ignored because of subsequent normalizations.

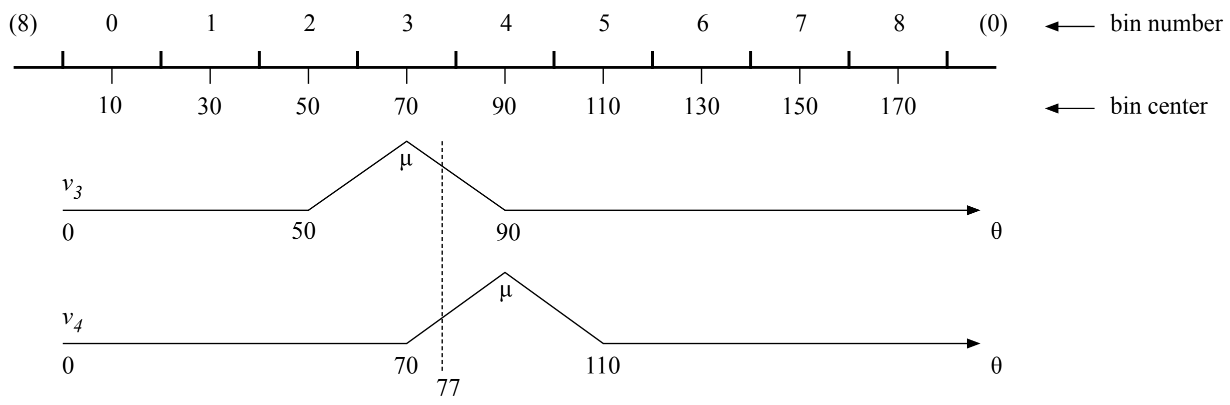


Figure 2: Voting by bilinear interpolation with $B = 9$ orientation bins. The two plots at the bottom show the votes v_3 and v_4 as the gradient orientation varies between 0 and 180 degrees. For instance, a gradient with orientation $\theta = 77$ degrees (dashed line) and magnitude μ contributes 0.65μ to bin 3 and 0.35μ to bin 4. The sum of the two contributions is always μ.

Specifically, the bins are numbered 0 through $B - 1$ and have width $w = \frac{180}{B}$. Bin i has boundaries $[wi, w(i+1))$ and center $c_i = w(i + \frac{1}{2})$. A pixel with magnitude μ and orientation θ contributes a vote

$$v_j = \mu \frac{c_{j+1} - \theta}{w} \quad \text{to bin number} \quad j = \left\lfloor \frac{\theta}{w} - \frac{1}{2} \right\rfloor \mod B$$

and a vote

$$v_{j+1} = \mu \frac{\theta - c_j}{w} \quad \text{to bin number} \quad (j+1) \mod B \; .$$

This scheme is called (for dubious reasons) *voting by bilinear interpolation* and is illustrated in Figure 2. The resulting *cell histogram* is a vector with B nonnegative entries.

Block Normalization. Group the cells into overlapping *blocks* of 2×2 cells each, so that each block has size $2C \times 2C$ pixels. Two horizontally or vertically consecutive blocks overlap by two cells, that is, the *block stride* is C pixels. As a consequence, each internal cell is covered by four blocks. Concatenate the four cell histograms in each block into a single *block feature* $\mathbf{b}$ and normalize the block feature by its Euclidean norm:

$$\mathbf{b} \leftarrow \frac{\mathbf{b}}{\sqrt{\|\mathbf{b}\|^2 + \epsilon}} \; .$$

In this expression, ϵ is a small positive constant that prevents division by zero in gradient-less blocks. The evidence for preferring this normalization scheme over others is entirely empirical.

Block normalization is a compromise: On one hand, cell histograms need to be normalized to reduce the effect of changes in contrast between images of the same object. On the other hand, overall gradient magnitude does carry some information, and normalization over a block—a region greater than a single cell—preserves some of this information, namely, the relative magnitudes of gradients in cells within the same block. Since each cell is covered by up to four blocks, each histogram is represented up to four times with up to four different normalizations.

HOG Feature. The normalized block features are concatenated into a single HOG feature vector $\mathbf{h}$, which is normalized as follows:

$$\begin{aligned} \mathbf{h} &\leftarrow \frac{\mathbf{h}}{\sqrt{\|\mathbf{h}\|^2 + \epsilon}} \\ h_n &\leftarrow \min(h_n, \tau) \\ \mathbf{h} &\leftarrow \frac{\mathbf{h}}{\sqrt{\|\mathbf{h}\|^2 + \epsilon}} \,. \end{aligned}$$

Here, h_n is the n-th entry of $\mathbf{h}$ and τ is a positive threshold ($\tau = 0.2$). Clipping the entries of $\mathbf{h}$ to be no greater than τ (after the first normalization) ensures that very large gradients do not have too much influence—they would end up washing out all other image detail. The final normalization makes the HOG feature independent of overall image contrast.

The resulting HOG feature is the concatenation of four times as many cell histograms as there are blocks. For instance, with a 128×64 window and cells with 8×8 pixels there are 16 cells vertically and 8 horizontally. With an 8-pixel block stride there are then 15 blocks vertically and 7 horizontally, and with 4 cells per block and 9 orientation bins per histogram the length of $\mathbf{h}$ is

$$15 \times 7 \times 4 \times 9 = 3780 \text{ entries.}$$

This is a rather rich descriptor for a window of $128 \times 64 = 8192$ pixels. Since the feature is the concatenation of $15 \times 7 \times 4 = 420$ histograms, it is sometimes useful to think of it as a 420×9 matrix.

As an example, the right panel in Figure 1 gives a relatively intuitive visualization of the HOG feature for the green window in the middle panel of the figure: Each cell in the visualization contains a tiny "pie chart" of sorts, with 18 slices, grouped into 9 pairs of opposite slices. The orientation of each pair corresponds to the center of a histogram bin and both the radius and the opacity of each pair are proportional to the average vote for that bin over the histograms associated with that cell. An orientation of zero degrees (horizontal gradient) corresponds to a vertical edge, so the slice pairs for 10 and 170 degrees are close to vertical.

The HOG feature conveys information that is somewhat like that of an edge map, except that some of the gradient magnitude information is retained and the location of the edges is only recorded to cell resolution. This location coarseness and the normalization scheme are the key properties of HOG features, as they give the representation some degree of invariance to small and local geometric and photometric changes.

In addition to adjusting for object size, the use of a pyramid during detection has also a beneficial effect on the level of detail. For instance, the blurring that results from the pyramid computation removes some of the grain caused by the splash of mud in Figure 1. That grain would confuse the HOG detector at finer scales.

Practical Aspects: Variants of HOG. The description above is one interpretation of the original paper [1] on HOG. Other papers have proposed different variants [3], and each variant may be implemented with different parameter values. A major practical difference between the Dalal and Triggs version [1] and the Felzenszwalb *et al.* version [3] is that while the former concatenates the four histograms in a block, the latter averages them together and stores the four histogram norms as additional entries in the feature vector. This results into a shorter feature vector. In addition, the Felzenszwalb *et al.* version uses oriented rather than unoriented gradients (that is, they consider a gradient $\mathbf{g}$ to be different from $-\mathbf{g}$). Which choice makes more sense depends on the application domain.

A small implementation difference is whether to output four histograms per block or four histograms per cell. In the description above, there are 15×7 blocks but 16×8 cells. If one outputs four histograms per

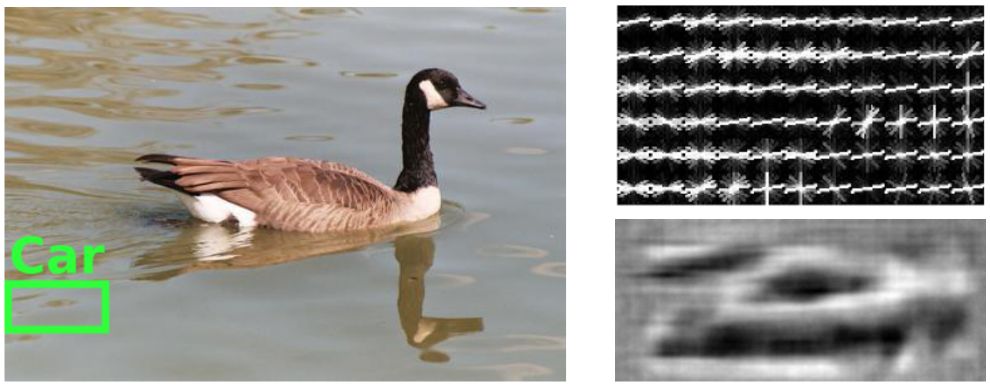


Figure 3: An obvious false-positive detection of a car (left), the usual visualization of the HOG feature (top right), and the gradient-magnitude image computed by the HOGgles system (bottom right).

block, then cells on the boundary of the window are represented by fewer than four histograms. If one outputs four histograms per cell, then histograms for boundary cells may be repeated. These alternatives result in slightly different feature vector lengths.

The description of HOG given above is for gray-level images. Color images can be processed by first converting them to gray. Alternatively, the gradient can be computed in each color band, and the gradient with greatest magnitude at each pixel can be used to compute the HOG feature.

The VLFEAT software library (`http://www.vlfeat.org`) developed by Antonio Vedaldi implements both variants of HOG mentioned above, and provides means for modifying their parameters. Other implementations exist.

Since HOG features are typically computed over a very large number of windows per image, efficiency is important, and several techniques have been employed to design very fast HOG feature computation algorithms [4, 2].

Visualizing HOG Features

A more interesting visualization method for HOG features than the "pie chart" was proposed recently [5]: Given a set of images and a set of HOG features from them, use sparse coding techniques to learn an approximate inverse for the function that computes the HOG feature for a given image. With this approximation, one can find the image that the HOG detector "sees." For instance, as shown on the web page for the "HOGgles" project (`http://web.mit.edu/vondrick/ihog/`), a car detector found a car in an unlikely image detail, as shown in Figure 3.

Visual inspection of the HOGgle output (bottom right of Figure 3) shows that the gradient magnitude distribution encoded by the HOG feature can be reasonably be interpreted as that of a car. This suggests that the culprit for the false positive is neither the training set nor the classifier, but rather the design of the feature. This type of visualization may suggest ways to improve on HOG.

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, June 2005.

[2] P. Dollar, S. Belongie, and P. Perona. The fastest pedestrian detector in the West. In *Proceedings of the British Machine Vision Conference*, pages 68.1–68.11. BMVA Press, 2010.

[3] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, September 2010.

[4] F. Porikli. Integral histogram: A fast way to extract histograms in Cartesian spaces. *IEEE Conference on Computer Vision and Pattern Recognition*, 1:829–836, 2005.

[5] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba. HOGgles: Visualizing Object Detection Features. *International Conference on Computer Vision*, pages 1–8, 2013.