An Implementation on Histogram of Oriented Gradients for Human Detection

Cansın Yıldız

Dept. of Computer Engineering
Bilkent University
Ankara,Turkey
cansin@cs.bilkent.edu.tr

***Abstract*— I implemented a Histogram of Oriented Gradients (HOG) detector for pedestrians using [1]. Although this new implementation gives good result on performance tests, it has some drawbacks when it comes to real application. Causes for it will be discussed.**

I. Introduction

THIS report is a detailed presentation of the conducted research on *Histogram of Oriented Gradients (HOG) for Human Detection*. **Fig. 1** shows some results of implemented HOG-based detector. After presenting the details of the method and dataset used for human detection, the obtained results will be explained.

II. Overview of the Method

I have a simple HOG detector and a sliding window pair for the application. A window slides through a test image, and for each shot HOG detector decides whether it is a pedestrian or not.

As suggested, for simplicity some steps of the original HOG algorithm is skipped at implementation, which ended up in four main steps: gradient computation, orientation binning, descriptor blocks and block normalization. Before starting their explanation separately, it will be wise to mention about default of the implementation.

The detector implemented at this experiment is based on *the default detector* suggested [1]. The detector has the following attributes: grayscale with no gamma correction; [-1 0 1] gradient filter with no smoothing; linear gradient voting into 9 orientation bins in 0°-180°; 16x16 pixel blocks of four 8x8 pixel cells; L2-Norm block normalization; block overlap (stride) of half of the block size; 64x128 detection window; 5-nearest neighbor algorithm.

III. Datasets and Methodology

At the beginning of the tests, I have used [1]'s whole dataset containing 2416 positive images and 1218 negative images. But since the classifier test's takes too long as I suspected, I had to drop the size a little bit.

So, different from [1], I discarded the left-right reflections of positive training examples because of performance issues. I also sampled 7308 patches from 1218 negatives (six patches each), but because of the same performance issues, I reduced

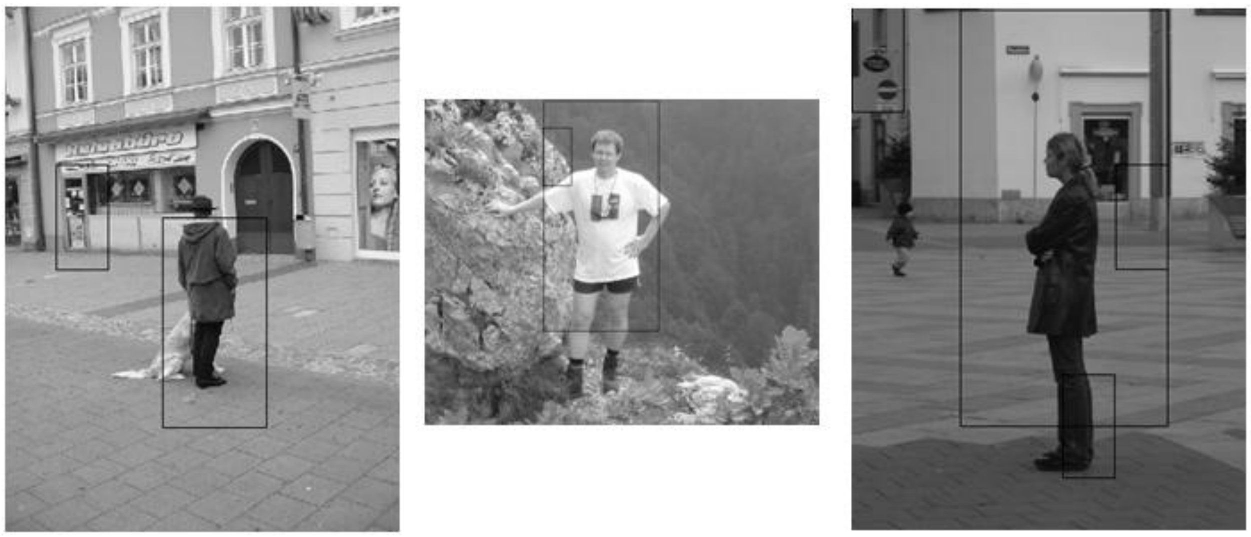

Fig. 1. Some Results of HOG detector. Boxes indicates detection of a pedesterian.

the negative images by only taking 1 patch from each image for 800 negatives (See **Fig. 2**). So, I ended up using 160 positive images and 800 negatives for my default classifier.

Positive training images are from *96X160H96/Train/pos* directory. Negative training images are from *Train/neg* directory. Positive performance test images are from 70*X134H96/Test/pos* directory, and negative performance test images are from *Test/neg* directory.

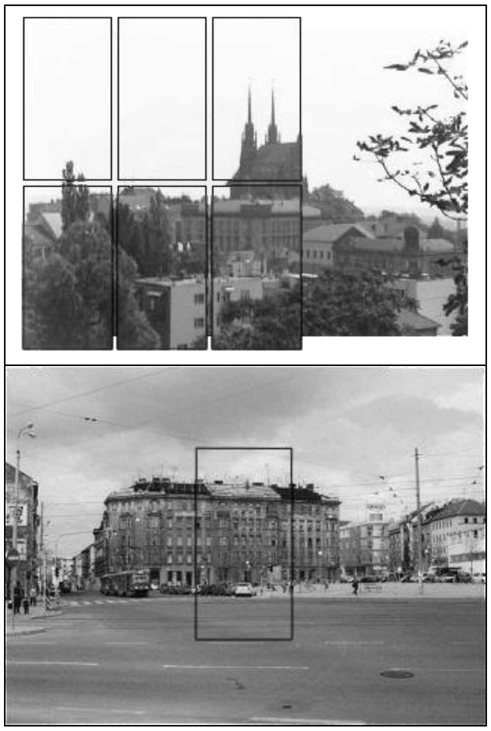
Fig. 2. Six Patch Extraction vs Single Patch Extraction for Neg. Train Dataset

IV. Implementation Details

A. Gradient Computation

For gradient computation, first the grayscale image is filtered to obtain x and y derivatives of pixels using ***conv2(image,filter,'same')*** method with those kernels:

$$D_x = \begin{bmatrix}-1 & 0 & 1\end{bmatrix} \text{ and } D_y = \begin{bmatrix}-1 & 0 & 1\end{bmatrix}^T \quad (1)$$

After calculating x,y derivatives (***I_X*** and ***I_Y***), the magnitude and orientation of the gradient is also computed:

$$|G| = \sqrt{I_X{}^2 + I_Y{}^2} \text{ and } \theta = \arctan\frac{I_Y}{I_X} \quad (2)$$

One thing to note is that, at orientation calculation ***rad2deg(atan2(val))*** method is used, which returns values between [-180°,180°]. Since unsigned orientations are desired for this implementation, the values which are less than 0° is summed up with 180°.

Some computed gradient for positive and negative train data can be seen at **Fig. 3** and **Fig. 4**, respectively.

B. Orientation Binning

The next step is to compute cell histograms for later use at descriptor blocks. As previously mentioned at Overview of the Method, 8x8 pixel size cells are computed with 9 orientation bins for [0°,180°] interval. For each pixel's orientation, the corresponding orientation bin is found and the orientation's magnitude |***G***| is voted to this bin.

Two sample orientations before binning can be seen at **Fig. 5** and **Fig. 6**.

C. Descriptor Blocks

To normalize the cells' orientation histograms, they should be grouped into blocks. From the two main block geometries, the implementation uses R-HOG geometry. Each R-HOG block has 2x2 cells and adjacent R-HOGs are overlapping each other for a magnitude of half-size of a block.

D. Block Normalization

Although there are three different methods for block normalization, L2-Norm normalization is implemented using ***norm(vec)*** method:

$$f = \frac{v}{\sqrt{\|v\|_2{}^2 + e^2}} \quad (3)$$

E. Detector Window

As previously mentioned, the detector window size is 64x128 pixels. This result in 8x16 cells and 7x15 R-HOG blocks, since blocks are overlapping. Each R-HOG block has 2x2 cells, which also has 1x9 histogram vector each. So the overall size of R-HOG descriptor of a window is 7x15x2x2x9. Since there is no need for multi-dimensionality in **R-HOG descriptor**, a vector of **size 3780** is used for each window.

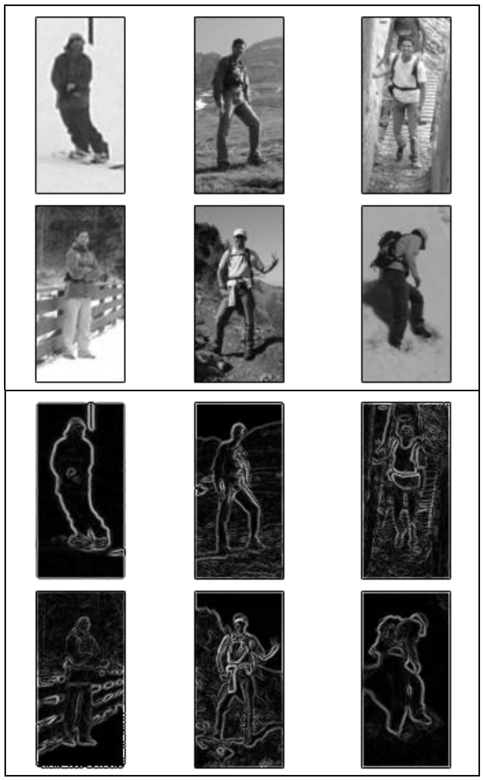
Fig. 3. Some samples from Pos. Train Dataset and their calculated magnitude vector visualizations

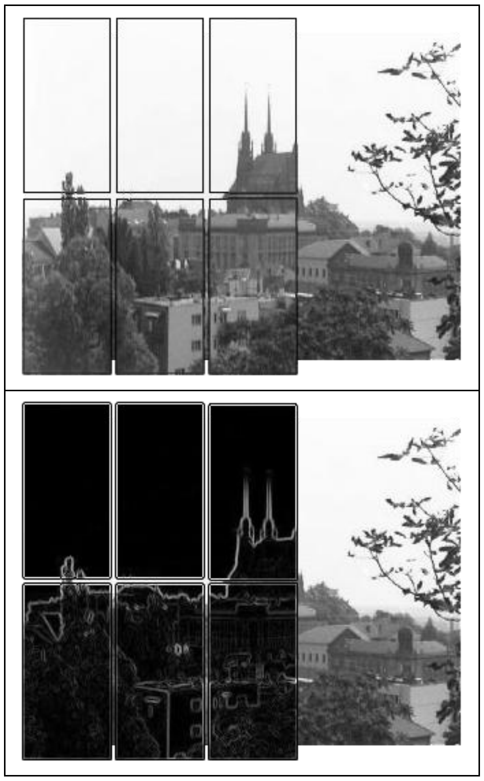
Fig. 4. Some samples from Neg. Train Dataset and their calculated magnitude vector visualizations

F. *Classifier*

When a test image is given to the system, two half-sizes are generated, resulting in three images with scales *1*, *1/2* and *1/4* of the original image. For each of these generated images, a window of size 64x128 is scrolled across the entire image with 32 pixel horizontal and 64 pixel vertical step sizes. And for each window that is cropped, a classifier is run as described below.

The implemented classifier is a simple *k-NN Classifier* with *Euclidian distance* measure. It has three modes. **NN classifier** mode simply computes the nearest neighbor of a test window and sets this neighbor's label as result. **Strict 5NN classifier** mode computes five nearest neighbors for a test window. Then if the closest neighbor's label ***and*** the dominant label in five neighbors is ***both*** *pedestrian*, the resulting label will also be *pedestrian*. Similarly, **Loose 5NN classifier** mode also computes five nearest neighbors. But this time, if the closest neighbor's label ***or*** the dominant label in five neighbors is *pedestrian,* the resulting label will also be *pedestrian.*

To sum up, I can say that the hardest acceptor for pedestrian label is **Strict 5NN classifier**, where **Loose 5NN classifier** is the easiest.

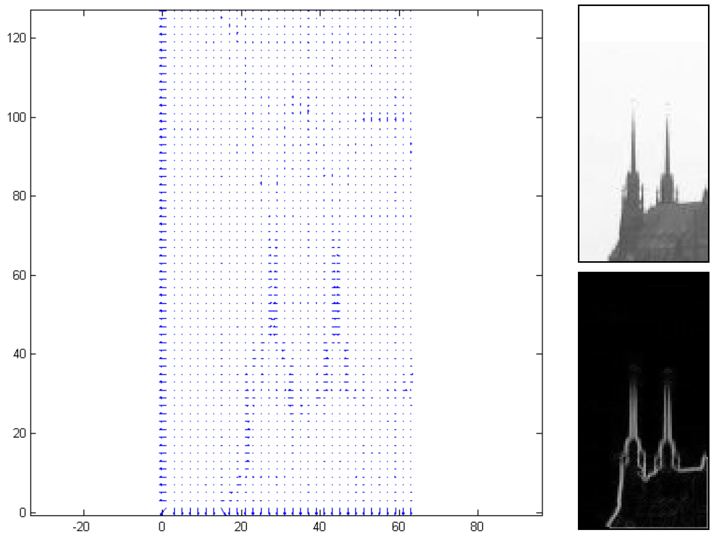

Fig. 5. A sample from Neg. Train Dataset, its magnitude and orientation visualization

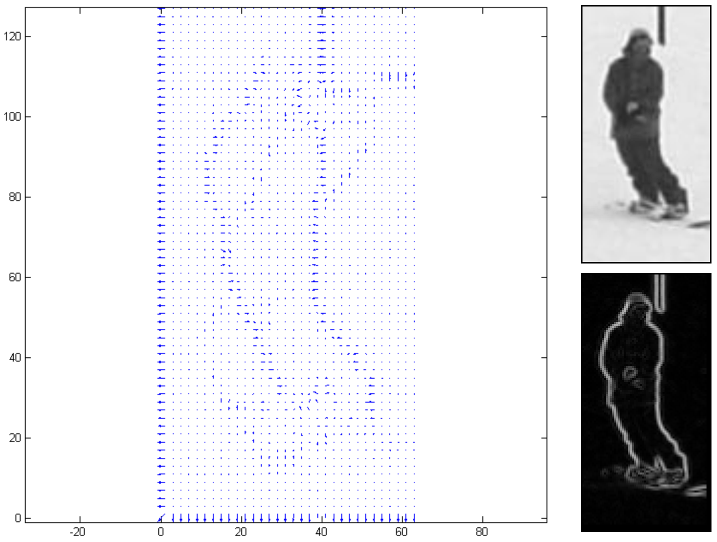

Fig. 6. A sample from Pos. Train Dataset, its magnitude and orientation visualization

V. Performance Study

I have evaluated overall performance for different environments. The parameters were *positive training dataset size*, *negative training dataset size* and *classifier method*. Two different kind of performance are measured during tests.

Positive Performance test is done using 1000 images (70x134) of true pedestrians. The reported percentage is the ratio of successfully *recognized* pedestrians and total number of pedestrians (i.e. 1000). In other words, if the percentage is low, probability of getting false negatives is high.

Negative Performance test is done using 250 images having no pedestrians. For each image 4 patches are cropped and tested. The reported percentage is the ratio of successfully *not-recognized* pedestrians and total number of patches (i.e. 1000). In other words, if the percentage is low, probability of getting false positives is high.

The performance comparison of different configurations can be seen at Table 2, Table 1 and Table 3. As seen, overall performance is quite high for every scheme chosen. But there are some expected differences between each of them.

TABLE I-II-III

PERFORMANCE RESULTS

TABLE 1. FULLPOS DATASET HAVING 2416 POS & 1218 NEG TRAINING

Classifier	Positive Performance	Negative Performance
Nearest Neighbor	80.6%	96.6%
Strict 5NN	77.9%	98.1%
Loose 5NN	90.9%	94.3%

TABLE 2. FULL DATASET HAVING 2416 POS & 7308 NEG TRAINING

Classifier	Positive Performance	Negative Performance
Nearest Neighbor	84.2%	~100%*
Strict 5NN	81.3%	~100%*
Loose 5NN	84.7%	~100%*

* *These are assumptions*

TABLE 3. SMALL DATASET HAVING 160 POS & 800 NEG TRAINING

Classifier	Positive Performance	Negative Performance
Nearest Neighbor	44.0%	99.3%
Strict 5NN	33.8%	99.6%
Loose 5NN	59.2%	97.8%

Dataset Selection

The first test configuration has full range of positive train images but having somewhat less negative train images. It has 2416 positive train images and 1218 negative train images that are single-patched from actual neg. images. It can be seen that, *Positive Performance* of first configuration outperforms both second and third configurations. This is an expected case; since negative training set is not large enough; the closest neighbor of any test image is more happen to be pedestrian images. But the biggest failure of this configuration is that it is much worse at catching false positives, because of the same characteristic. Since false positives are the most unwanted behaviors by most detection algorithms, this configuration will not be favored.

The second test case has the full dataset provided by [1]. It has 2416 positive train images and 7308 negative train images. Those 7308 negative train images are six patched from actual negative train images as shown at **Fig. 2**. Since this configuration has much more negative train data, the false positive failure is almost zero. 1000 *Negative Performance* images cannot be tested at this configuration, since the dataset was huge. But for a smaller test of 100 images, system does not detect any false positives.

Although the second configuration is the most reliable one, it has a huge drawback of performance. So, the third configuration is derived from this need. It has 160 positive images and 800 negative images each single-patched from actual negatives. Unfortunately, the overall performance is dropped radically as I expected. It has nearly half of *Positive Performance* of second configuration, but *Negative Performance* is still good. Although it has a good performance gain, I can still say that *Full Dataset Having 2416 Pos. and 7308 Neg. Training* is the best configuration among three of them.

Classifier Selection

Another important factor on performance is classifier method. There is actually not much to say about it. *Strict 5NN* had obviously the highest *Negative Performance* and the lowest *Positive Performance*, since it looks for both closest neighbor and dominant neighborhood. Oppositely, *Loose 5NN* had the lowest *Negative Performance* and the highest *Positive Performance*. *Nearest Neighbor* is at somewhere in-between.

The best choice among those approaches would be ***Strict 5NN.*** Although it looks like it lacks the *Positive Performance* of others; at application tests it occurred that *Negative Performance* is even more important, since most of the windows that are tested for a scene will not contain any pedestrian, we don't want to have false positives out of everywhere.

VI. OVERVIEW OF THE RESULTS

As explained at Classifier, for a given image three different scales of 1,1/2 and ¼ are first generated. Than a window of size 64x128 is slide at all of those scales. For every window position, a R-HOG descriptor is computed and looked for the nearest neighbor. If it's pedestrian, we have detection. Although the performance study promises a high applicability of the implementation to this real-life example; the tested full-size images hardly say so.

Upon further investigation, I see that when the detection window is a position so that the pedestrian is not exactly at center of the image with respect to left-right line, the performance drops drastically. To better explain the behavior; I had an additional test where all 1000 positive test images from performance tests are shifted left by 3 pixels. The detection performance of *Small Dataset with Strict 5NN* is dropped from *33.8%* to *17.3%*. This concludes that, the real problem of application is not just about calculating HOGs, but finding correct windows is also an important issue. The default step size between each window was 64x32. But when I further decrease it to a value like 8x8, both the computation time increase and false positive increase was at inacceptable degrees.

At a first glance, it could seem like that with a negative performance of 98.8%, the false positives cannot increase that much. But consider the case, for an image of 640x480 pixels, with a step size 8x8 and window size 64x128, there will be around 3200 window tests. Let's say 3000 window will not cover the pedestrian. So with a false positive rate of 0.4%, we will end up having 12 false detections! For an image with only 1 pedestrian, it is not tolerable.

This situation should be handled by using some intelligent window sliding technique like having a voting system of the detection results of each window with respect to their position, size etc.

VII. Conclusion

Although my implementation's performance results were good, to have a reasonable computation time, I had to use *Small Dataset* configuration for my detection application. This really lessens the product's quality. But with a more compact dataset of pedestrians and with a better coding than my all for-loop involved code, I believe a much successful application could be achieved.

I had started the paper with **Fig. 1**, a demonstration of end-user application, and I want to conclude it with some sample results from *Positive Performance Tests.* You can see them at **Fig. 7** and **Fig. 8**.

References

[1] Navneet Dalal , Bill Triggs, "Histograms of Oriented Gradients for Human Detection," in Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1, p.886-893, June 20-26, 2005

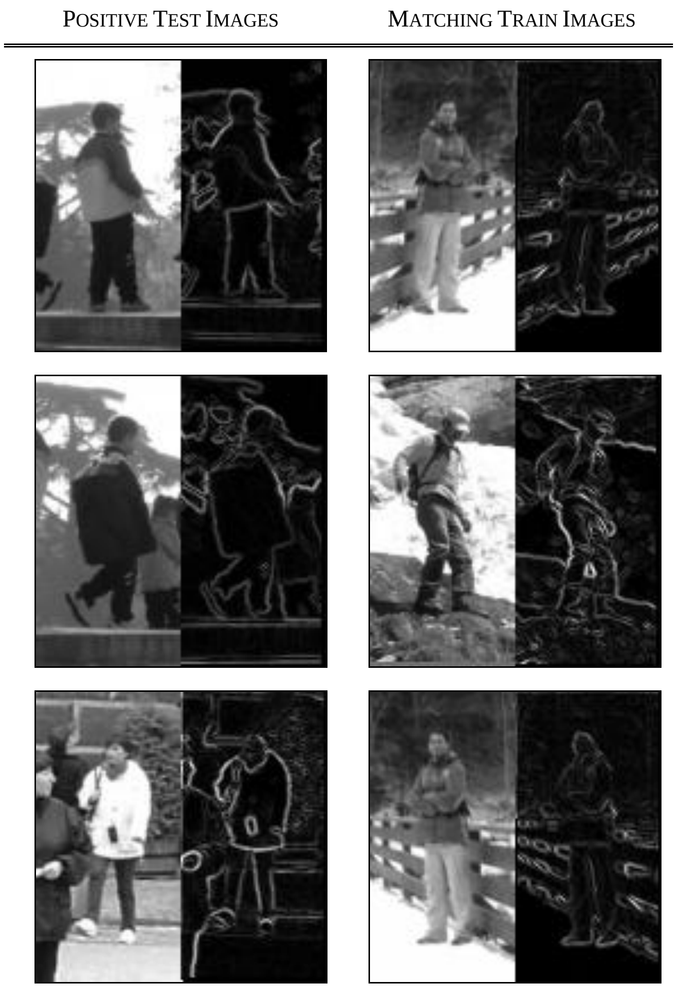

Fig. 7. Some sample positive test images and their corresponding positive training matches

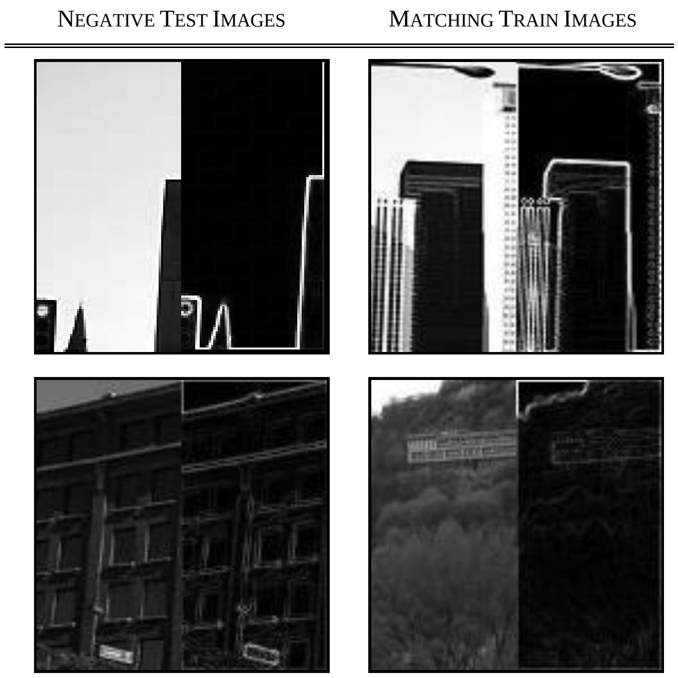

Fig. 8. Some sample negative test images and their corresponding negative training matches