ENGG5104 Image Processing and Computer Vision

Assignment4 – Pedestrian Detection

Due Date: 11:59 PM, April 10th, 2016

1. Background

Pedestrian detection (see Fig. 1) is a challenging problem in computer vision. One of the typical and effective frameworks applies histogram of gradient (HOG) as descriptor and linear SVM to train the pedestrian detector.

Figure 1: Pedestrian detection

Using HOG descriptors to represent training samples, we can train a linear detector by SVM. In detection phase, we convolve the linear detector with HOG descriptors extracted from the dense detection windows within an image, assigning detection (prediction) score to each window. A threshold is enforced to decide whether a window contains a human.

2. Histogram of Gradient

HOG descriptor [1][2][3] captures edge or gradient structure that is very characteristic of local shape. In addition, it is a local representation with controllably invariance to local geometric and photometric transformations. Before illustrating the construction of HOG descriptor, we firstly describe the Laplacian Gradient.

2. 1 Laplacian Gradient

Laplacian Gradient (LG) is a fundamental concept in computer vision. Fig.2 shows how to compute the gradient of a single pixel. Take a patch from an image and mark a certain pixel within this patch to be red. This pixel's value is 58. We can also access its four neighbors (left, right, top and bottom pixels) and their pixel values are 42, 31, 36 and 17. LG of the selected pixel (value of 58) consists of the gradients of horizontal (31-42) and vertical (36-17) directions, which is LG=[-11,19]. Its magnitude and angle can be figured out as 21.95 and 2.1 (rads) base on the LG.

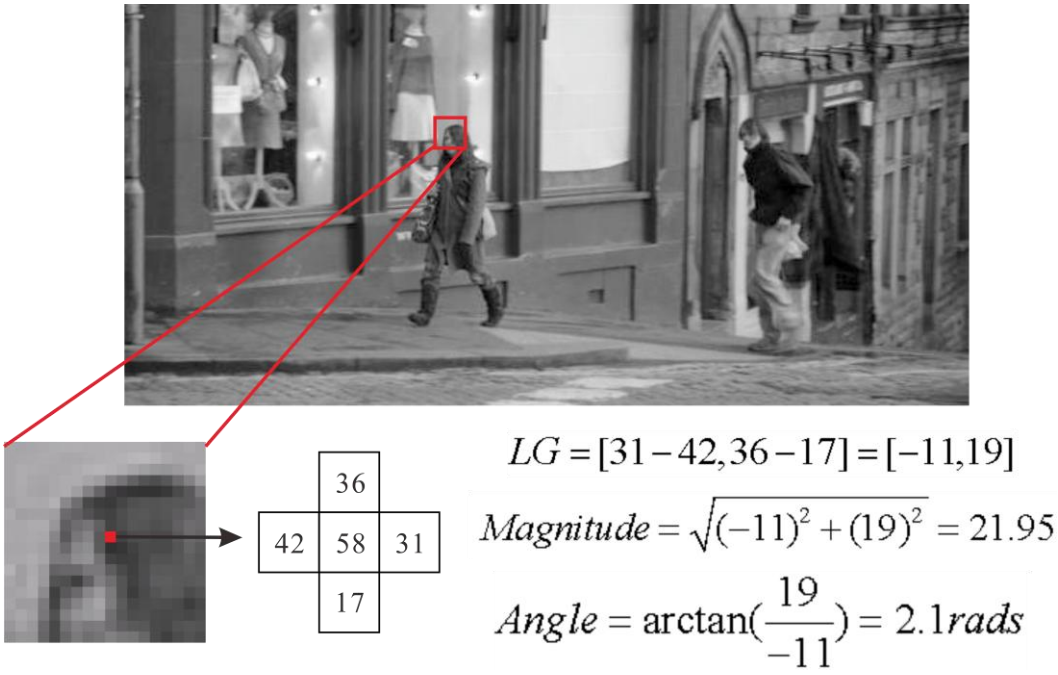

Figure 2: Gradient vector, angle and magnitude

2.2 Constructing HOG Descriptor

Given an image patch, which we call it a "cell" in the rest of this specification, we can obtain angles of all the pixels within the cell by following the above-mentioned computation. For example, we set a cell cover 8 x 8 pixels, then we can obtain 64 angles and magnitudes. **We accumulate the 64 magnitudes into a 9 bin histogram according to the corresponding angles which ranges from $-\pi$ to π, i.e. the interval is 40 degrees.** The histogram is the HOG of a cell.

In practice, we often group several cells into a block, e.g. the block 1 and block 2 which contain 2 x 2 cells shown in Fig. 3.**The neighborhood blocks always overlap with each other by 50%.** Take block 1 for example, we respectively obtain 4 HOGs of the cells within this block. Then the 4 HOGs are concatenated

into a vector. Denote the concatenated vector as v, we normalize v by its L2-norm, i.e. $v = v / \sqrt{\| v \|_2^2 + \varepsilon^2}$, where ε is a small residual for avoiding "division by zero" in practice. The normalized vector v is the HOG descriptor of a block. Assume we have a 256x128 gray scale image and set the cell to cover 8x8 pixels. Then this image can be divided into 256/8=32 cells vertically and 128/8=16 cells horizontally. We also enforce each block contain 2x2 cells. Following the strategy of overlapping the blocks by 50%, we can group the cells into 32-1=31 blocks vertically and 16-1=15 blocks horizontally. Totally there are 31x15=465 blocks in this image. Since the dimensionality of the HOG descriptor for a block is 9 bins x (2x2) cells = 36, then dimensionality of final descriptor of this 256x128 image is 36 x 465 blocks = 16,740. **In the rest of this specification and the implementation, we fix the number of bin to 9. We also fix the size of a cell to cover 8 x 8 pixels and block to contain 2 x 2 cells**.

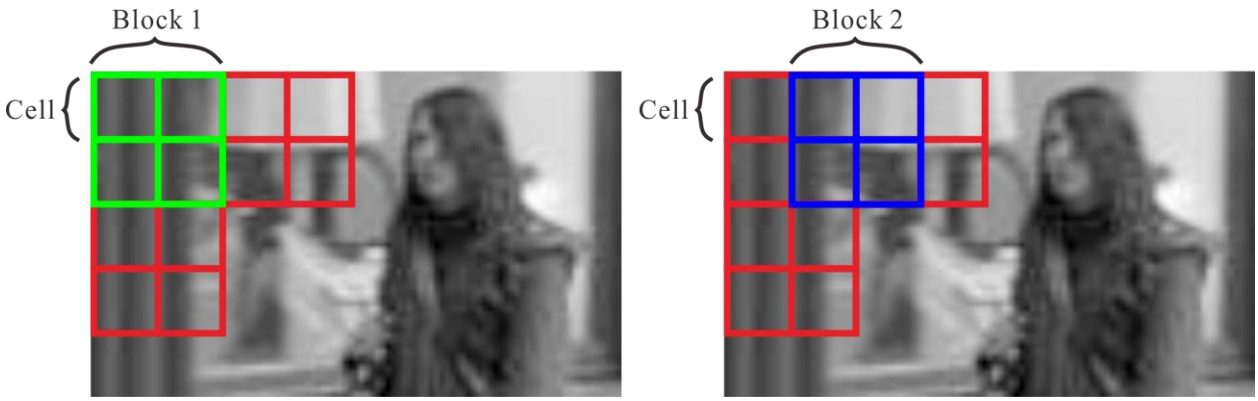

Figure 3: Cell and block

3. Pedestrian Detection Using HOG

In this section, we describe how to train a linear detector and use the detector to detect pedestrian within a given image.

3.1 Train Linear Detector

Given a set of training images, we divide it into positive samples and negative samples. The positive samples are the images containing only one pedestrian, while the negative ones contain no human. Fig. 4 shows the positive and negative samples.

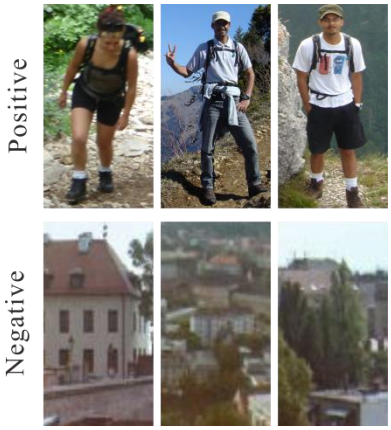

Figure 4: positive and negative samples

We resize each sample to a fixed size, i.e. 128 x 64. This size is applied to the training samples in the implementation later. Then we extract HOG descriptor for each training sample, which is a 15 x 7 x 36=3,780D vector. All the HOG descriptors are fed into two-class linear SVM to train the human detector. The obtained detector is also a 3,780D vector. In practice, we reshape the detector to a 3D matrix, i.e. in 15 x 7 x 36, for the convenience of detection in the following step.

3.2 Sliding Window Strategy

Given a novel image, our task is to detect the pedestrian(s) from it. In this step, we firstly follow Section 2 to obtain the cells and blocks within the novel image. Let us take a 912 x 912 gray scale image for example. It is divided into 113 x 113 blocks. We can extract the HOG descriptor for each block and then form a 3D matrix in 113 x 113 x 36. In the above steps, we train a linear pedestrian detector in 15 x 7 x 36. Then we convolve the 15 x 7 x 36 detector on the 113 x 113 x 36 matrix, which yields central part of the convolution in 113 x 113. We enforce a threshold to select the position where the score of convolution is above the threshold as hypothesis. In this project, we will use the OpenCV function "filter2D" to implement the sliding window strategy.

3.2 Multiple Scale Manner

In practice, the scales of human are various. However, the scale of the detector is fixed (15 x 7 x 36), which makes it difficult to detect the human on other scale, e.g. 30 x 14. For better performance, we perform detection on multiple scales of a given image. To achieve detection on multiple scales of image, we simple resize the image by several fixed factors, e.g. 0.21, 0.27, 0.3. The thresholds for these scales are set to 3.0, 3.5 and 4.5 respectively. You are allowed to tune/remove/add these factors to achieve better results.

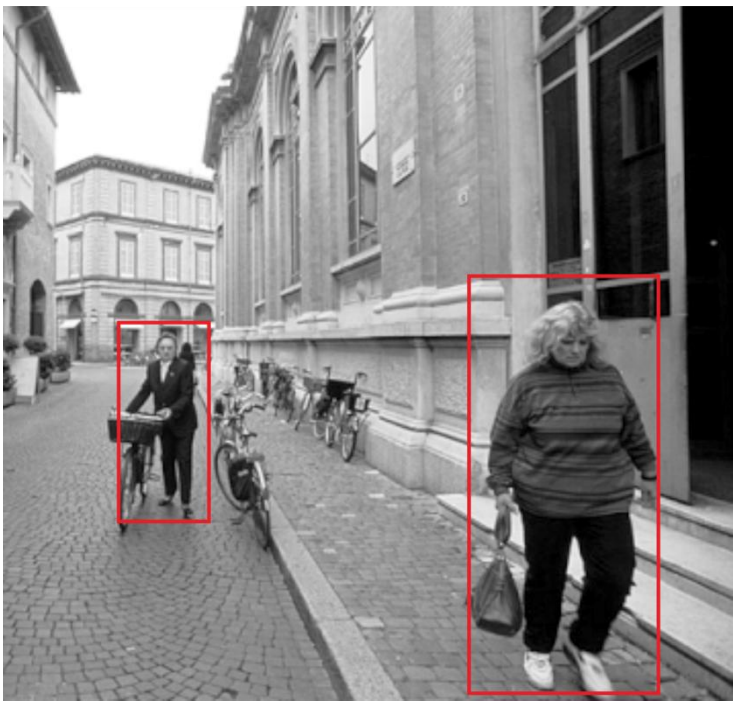

Figure 5: The sample result of detection

4. Implementation Details

Given the skeleton code for object detection using HOG, you need to write your own code in it. The requirement is as follows.

TODO #1: Compute angle and magnitude of gradient for each pixel (file:*HOGExtractor.cpp*)

For each pixel in the image, you should firstly compute the gradient vector, angle and magnitude as described in Section 2.1. Note that the range of angle should be from $-\pi$ to π.

Hints: The vertical and horizontal gradient can be computed by convoluting the kernel [1;0;-1] and [-1 0 1] on the input image, using the OpenCV function "filter2D". The angle can be computed by the math function "atan2", which returns the angle exactly ranges from $-\pi$ to π.

TODO#2: Construct HOG for each cell(file: *HOGExtractor.cpp*)

We fix the size of cell to make it cover 8 x 8 pixels and the bin of HOG to be 9. Before we segment the cells, we have padded the image in order to make its height and width to be the multiplication of 8. Your task is to construct the HOG for each cell within the input image. To achieve this goal, you should first quantize the range of angle ($-\pi$ to π) to 9 bins. And then, accumulate the pixel's magnitude to the 9 bin histogram according to its angle. For each cell, you can construct a 9 bin histogram.

TODO#3: Construct the normalized HOG for each block (file: *HOGExtractor.cpp*)

We fix the size of block to make it cover 2 x 2 cells. Note that the neighboring blocks overlap with each other by 50%. To construct the HOG of a block, you are to concatenate the 4 HOGs of the 4 cells within the given block, and then normalized the concatenated vector by its L2-norm. You are also free to try other strategies of normalization [3], e.g. L1 normalization. The OpenCV function "normalize" could be used to normalize a vector.

TODO#4: Convolute the learned detector on the HOG descriptor of the novel image(file: *Detector.cpp*)

After all the implementations required in file "HOGExtractor.cpp", we can extract the HOG descriptor from the novel image. Then your task is to convolute the learned detector on the extracted HOG descriptor.

Hints: Since the detector and HOG descriptor are all 3D matrices, you may need a loop to traverse the third dimension to obtain the 2D convolution on each dimension using OpenCV function "filter2D". Then the final convolution score can be obtained by summing up the 2D convolution along the third dimension. Take a 256 x 256 x 36 HOG descriptor and a 15 x 7 x 36 detector as examples. The process of convoluting exactly results in a 256 x 256 score map. The OpenCV function "split" can be used to separate the channels along the third dimension of the descriptor/detector.

TODO#5: Select the convolution above the threshold and compute its bounding box(file: *Detector.cpp*)

After TODO #4, we have a score map. To select the hypothesis, we use a threshold to make the decision. If a value in score map exceeds the threshold, we suppose it to be a hypothesis. Then we place the center of bounding box on this location. Your task in this step includes select out the hypothesis and place bounding box in the same height and width of the detector. Note that you should also compute the coordinate of the bounding box in image space. Fig. 3 shows the relationship among the pixel, cell and block. With this relationship, it is easy to recover this coordinate.

TODO#6: Build image pyramid(file: *Detector.cpp*)

In this step, you are required to build an image pyramid. The pyramid contains the same image in different scales.

Hints: The OpenCV function "resize" is well-designed for this purpose.

TODO#7: Perform detection on multiple scales of image(file: *Detector.cpp*)

After obtaining the image pyramid, your task then is to perform the detection on the images in the pyramid, by calling the implemented function in TODO #6. Please remember to recover the coordinate of hypothesis

on different scales to the original one, by multiplying the inverse of corresponding scales. Fig 5 shows examples of the detection.

5. Marking

BasicPart (100%)

We provide two set of samples for training and validating respectively. The training set consists of 2,416 pedestrian images (positive samples) and 6,090 non-pedestrian images (negative samples). For fair comparison, we restrict all of you to use this fixed training set to learn the linear detector. But you are welcome to pre-process the training data, including flipping, rotation and so on, which may enhance the performance of the detector.

Besides filling in the skeleton code, you are free to adjust the parameter of linear SVM, the thresholds and scales for detection. Please DO NOT modify the size of bin, cell and block. Your task is to tune the linear detector and all the controllable parameters in order to achieve ideal performance on the validating set.

We set up measurement of detector's performance in file "main.cpp". Our benchmark for measurement is described in [4][5], which is called "average precision" (AP) and implemented in file "DetEvaluator.cpp". Please DO NOT edit the codes in this file. Your implementation will be evaluated on the private testing set preserved by us. The cpp program outputs the AP of your implementation and save the result in a text file. You are encouraged to analyze the performance of detection by plotting the curve of precision-recall using the MATLAB script "plot_pr.m" in the folder "Report". The curve of precision-recall should be like Fig.6.

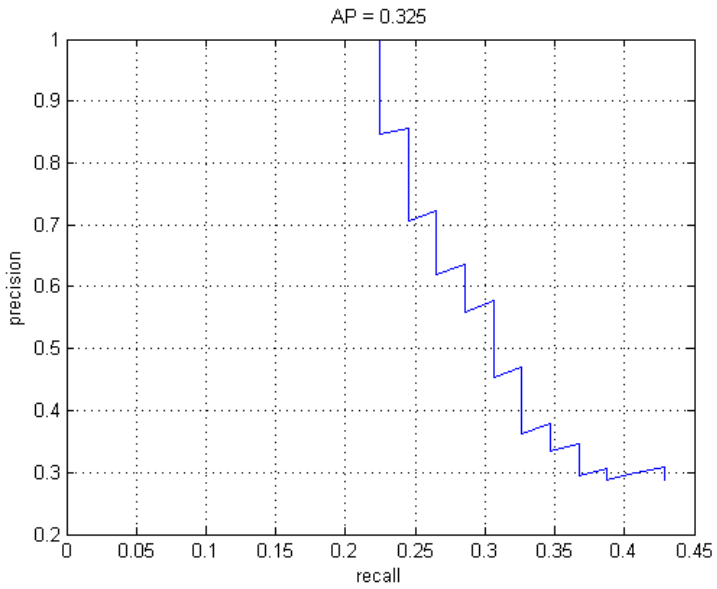

Figure 6: The curve of precision-recall

Extra Credit

You may consider the following bonus:

Bonus 1(10%): Implement C-HOG described in [3].In the basic part of this project, the implemented HOG is called R-HOG (rectangular HOG). This is because the block partitions cells into square or rectangular grids, as shown in Fig. 3. C-HOG (circular HOG) conducts log-polar grid to divide the cells, as shown in Fig.7.

Figure 7: Log-polar grid conducted by c-hog

The C-HOG layout has four parameters: 1) the numbers of angular and radial bins; 2) the radius of the central bin in pixels; and 3) the expansion factor for subsequent radii. As reported in [3], the layout (which include 1 central cell and 4 angular cells) shown in Fig.7 achieves ideal performance. To implement C-HOG, you should extract the HOGs from the central and angular cells within a block, concatenate and normalize them.

Bonus 2(10%): Try the following block normalization schemes described in [3] other than L2-norm:

1) L1-norm: $v = v / (\| v \|_1 + \varepsilon)$

2) L1-Sqrt: $v = \sqrt{v / (\| v \|_1 + \varepsilon)}$

3) L2-Hys: $v = v / \sqrt{\| v \|_2^2 + \varepsilon^2}$, limiting the maximum value of to a constant, e.g. 0.2.

Bonus 3(10%): Implement your own strategy of non-maximum suppression. Non-maximum suppression is a post processing which is used to eliminate or combine the overlapped bounding boxes, as shown in Fig. 8.

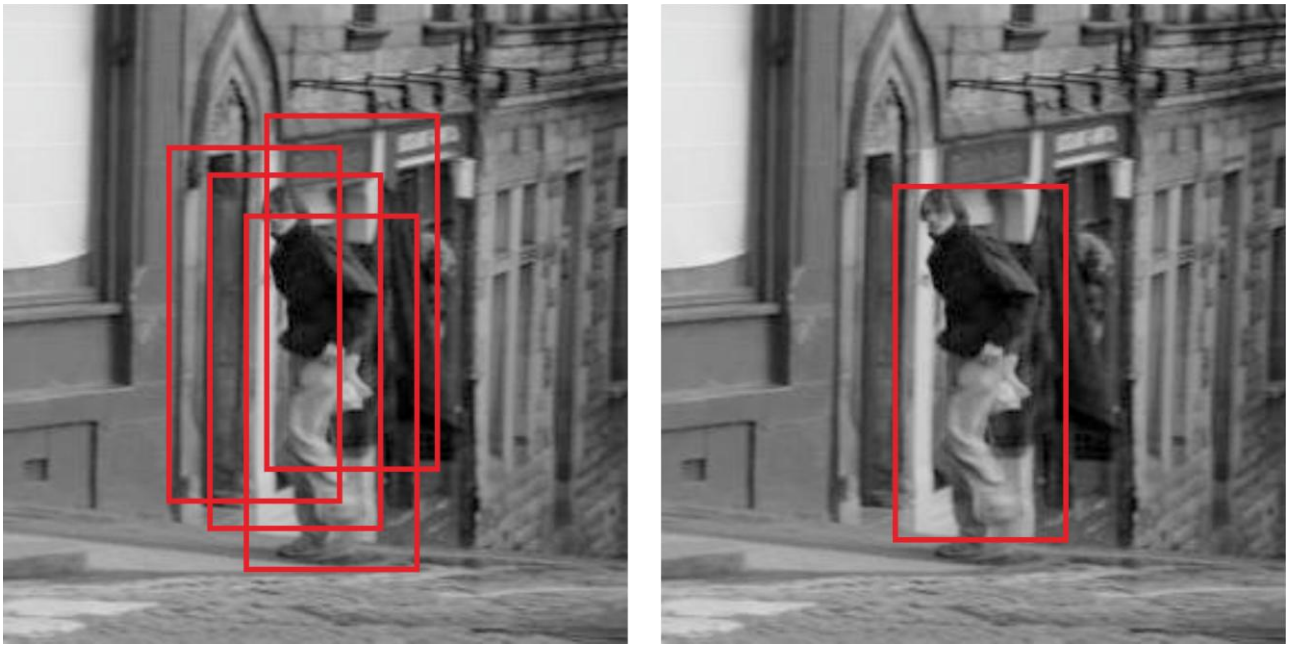

Figure 8:Non-maximum suppression

We provide a simple yet efficient version of non-maximum suppression(see function "NonMaxSup"in file "PostProcessor.cpp". As shown in Fig. 9, this function greedily selects the bounding boxes with high scores, while eliminating those significantly covered by the previously selected detection. However, this function may suffer from a problem. As the provided version greedily select the bounding boxes in descent order of detection scores, it is likely to eliminate the following ones which may also have the same or approximate scores. As shown in Fig. 10, the strategy of elimination select the box (score 1.6) and skip the other one (score 1.5), though the latter one seems to be a better hypothesis regarding to the ground truth. Thus, based on the original strategy, an improved version should consider combining those bounding boxes which heavily overlap and have approximate detection scores. For example, the two bounding in Fig.10 are combined into a tight and bigger bounding. There are also other flexible strategies to eliminate or combine the bounding box according to their overlapping and detection scores. To win this bonus, your goal is to develop a more reasonable scheme of non-maximum suppression which profits to enhance the performance of detection.

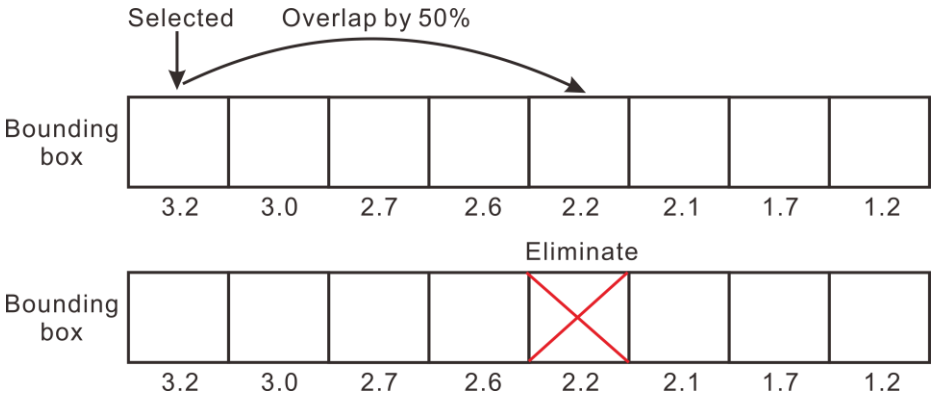

Figure 9: Greedy non-maximum suppression

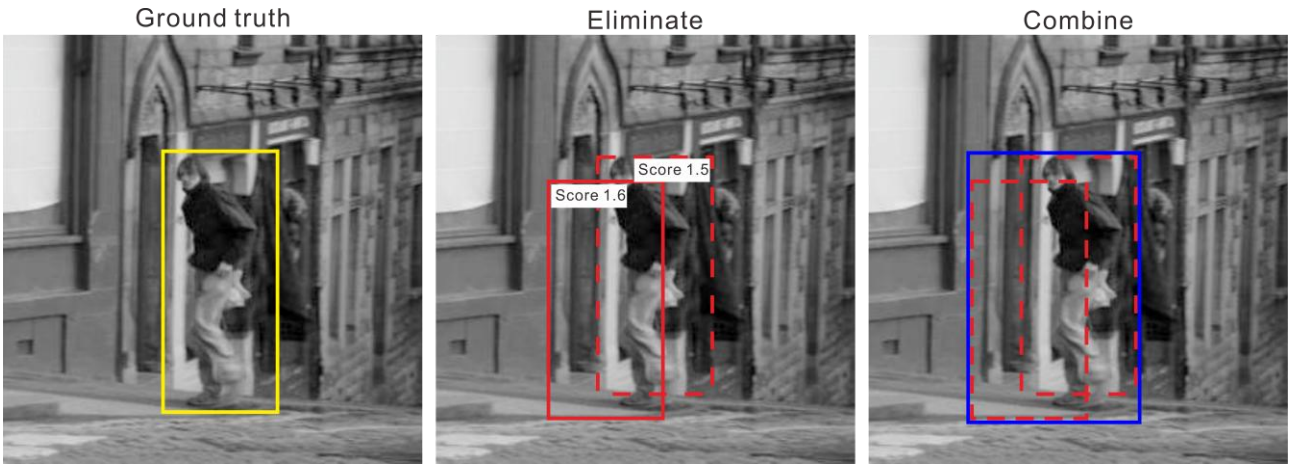

Figure 10: Eliminate and combine

Bonus 4(10%): 5 students producing the best results regarding the measurement will automatically win this bonus.

6. Code Installation

We provide the skeleton code organized as a project of Visual Studio 2010. The only prerequisite is OpenCV 249. You should download OpenCV from its official homepage:

https://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.4.9/

Put this OpenCV to Library subfolder of your project. The structure is like

<your project>\Library\opencv_2_49\include - include folder for OpenCV

<your project>\Library\opencv_2_49\build - library folder for OpenCV

We have already set up all the necessary environmental parameters for you to compile and run the code. You may also use other version of Visual Studio and OpenCV. But you need to set up the environment by yourself.

7. Submission

Your submission should contain two items: **code files (in cpp)** and **a report (in HTML)**

Code

The completed code files, including your implementation and the skeleton code we provide.

Report Requirements

1. The report must be written in **HTML**.
2. In the report, you should describe your algorithm and any decisions you made to write your algorithm a particular way.
3. You should also show and discuss the results of your algorithm (including pre-processing of training data, results, parameter settings, and analysis etc.).
4. Discussion on algorithms' efficiency (based on time elapsed).
5. Ideas and algorithms from others **MUST** be clearly claimed. List papers or links in the **Reference** section if needed.
6. Feel free to add any other information you feel is relevant.

Submission Format

The folder you hand in must contain the following:

1. **README.txt** - containing anything about the project that you want to tell the TAs, including brief introduction of the usage of the codes
2. **code/** - the project of Visual Studio 2010directory containing all code for this assignment. To smooth the process of marking, we remind you to DELETE 1) all the training and validation images in folder "Dataset"; 2) the extracted HOG descriptor in folder "Descriptor" and 3) the OpenCV in folder "Library". Please REMAIN the learned detector in folder "Detector" since we will use it to evaluate your implementation.
3. **html/** - directory containing all your html report for this assignment (including images). Your web page should only display compressed images (e.g. jpg or png).
4. **html/index.html** - home page for your results

Please compressed the folder into <*your student ID*>-asgn4**.zip** or <*your student ID*>-asgn4**.rar**, and upload it to e-Learning system.

8. Remarks

1. Five late days total, to be spent wisely.
2. 20% off per day for late submission.
3. Your mark will be deducted if you do not follow the instructions for the submission format.

4. The assignmentis to be completed **INDIVIDUALLY** on your own.
5. You are encouraged to use methods in related academic papers.
6. Absolutely **NO sharing or copying** of code! **NO sharing or copying** of reports! Offender will be given a failure grade and the case will be reported to the faculty.

Reference

[1]. HOG tutorial. http://chrisjmccormick.wordpress.com/2013/05/09/hog-person-detector-tutorial/

[2]. HOG introduction. http://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

[3]. N.Dalal and B.Triggs. <<Histograms of Oriented Gradients for Human Detection>>, CVPR 2013.

[4]. Precision and recall. http://en.wikipedia.org/wiki/Precision_and_recall

[5]. Average precision. http://en.wikipedia.org/wiki/Information_retrieval#Average_precision