Final Project Report
Introduction to Computer Vision
Prof. Lee Kyoung Mu
TA: Gyeongsik Moo, Baik Sung Yong David

Nguyen Tuan Nghia
Student ID: 2018-21525
Email: nghianguyentuan@snu.ac.kr
Seoul National University

1 Human Detection

In this section, I discuss briefly how to detect human in single image using HOG features and Linear SVM classifier.

1.1 Setting Up Environment

NOTE: During Testing MODE, only the best models of each task are tested. Please uncomment corresponding line in main_script.m to test the others!
At first, we need to make sure that the working environment is clean by executing 'clear' command. Since MODE is set by user, this command is not included in the code. After MODE is set, the main script can be safely executed.
Let's set up the environment, including training and test image directories, and ground-truth files. For human detection, window size is set to 128 by 64 and a stride of 8 by 8 is applied for dense scan of sliding window. Cell size is set to 8 by 8 pixel and cell layout in each block is 2 by 2 for extracting HOG features, while for LBP, cell size is 16 by 16 pixel. I use the MATLAB functions 'extractHOGFeatures(*)' and 'extractLBPFeatures(*)' in my implementation. 'vl_hog(*)' was also tried, however, this command provides poor options to handle input and output, so it is not included in this work.
All the pre-trained model is saved in the file 'model.mat' and will be loaded to the workspace if MODE is set to 2 (testing mode).

```
images_pos_dir = '../../datasets/INRIAPerson/train_64x128_H96/pos/';
images_neg_dir = '../../datasets/INRIAPerson/Train/neg/';

% Test image directory
images_test_dir = '../../datasets/INRIAPerson/Test/pos/';

% Ground-truth labels
human_label_path = 'human_gt.txt';

% Detection window size
human_window_size = [128 64];
human_window_stride = [8 8];

% HOG Settings
human_hog_cell_size = [8 8];
human_hog_block_cell = [2 2];

% LBP Settings
human_lbp_cell_size = [16 16];

if (MODE == 2)
   % Load the pre-trained model
   load('model.mat');
end
```

1.2 Simple HOG and Linear SVM

Before training the first simple model, we need to prepare the training data. I implement the helper function 'prepare_training_input(*)' to create positive and negative samples from INRIA dataset. For the positive samples, I simply crop the detection window from the center of the normalized positive images. Negative samples are randomly selected from the whole image set which contains no human. Each negative images give 10 samples. The final training set includes 2,416 positive and 12,180 negative images
Next, I extract HOG features from each sample, and create the training data as well as its associated label data. I use 'parfor' for parallel computing, which makes it 2 times faster to perform features extraction on the input. Training linear SVM is done with the help of MATLAB 'fitcsvm(*)'. The result of this function is a model, which can be used to predict labels on test data through 'predict(*)'. However, I notice that the result model contains a lot of redundant things that make its size large. Therefore, only 'Beta' and 'Bias' data of the model is kept while the others are discarded. Also, I do not use 'predict(*)' due to its slow performance. Instead, scores and labels can be directly computed from 'Beta' and 'Bias' parameters:

$$score = X \times \beta + b \tag{1}$$

$$label = \begin{cases} 1 \text{ if } score \geq 0 \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

where X is feature vector, β is 'Beta' (weights) data and b is 'Bias' data.

```
% Create list of positive and negative images for common use
[imp_list, n_imp] = prepare_training_input(images_pos_dir, human_window_size, 'center',
    []);
[imn_list, n_imn] = prepare_training_input(images_neg_dir, human_window_size, 'random',
    10);

imx_list = [imp_list; imn_list];

% Ground-truth for training
% 1 ~ positive, 0 ~ negative
y = [ones(n_imp,1); zeros(n_imn,1)];

% Try to measure feature length on a toy sample
unrolled_hog_feature_length = length(extractHOGFeatures(ones(human_window_size),
    'CellSize', human_hog_cell_size, 'BlockSize', human_hog_block_cell));

% Initialize training data
X = zeros(n_imp + n_imn, unrolled_hog_feature_length, 'single');

% Extract HOG features
% Parallel computing: 2x faster
parfor i = 1:n_imp + n_imn
   X(i,:) = extractHOGFeatures(imx_list{i}, 'CellSize', human_hog_cell_size, 'BlockSize',
       human_hog_block_cell);
end

% Train Linear SVM classifier
fprintf('Training SVM Classifier with HOG ... ');
svm_model = fitcsvm(X,y);
fprintf('done\n');

% Trained model contains a lot of redundant things
% We only keep the main components: Beta and Bias
% Score = X * Beta + Bias and Threshold = 0
svm_human_hog = [svm_model.Beta; svm_model.Bias];
```

After the model is ready, it can be tested through the helper function 'test_svm_classifier_hog(*)'.

```
% HOG + LINEAR SVM MODEL
test_svm_classifier_hog(svm_human_hog, human_window_size, human_hog_cell_size,
    human_hog_block_cell, images_test_dir, human_label_path)
```

1.3 Mining Hard Negatives

The simple HOG and Linear SVM model presented in the previous section uses random data sampled from the whole image set. It raises a problem that whether the those samples are good enough or not. To answer this question, as well as to improve the performance of the current model, I apply Hard Negatives Mining technique as described by Dalal et. al. [1].
The hard negatives are actually the samples that are misclassified by the simple model. The number of hard negatives can be very large and may cause out of memory error, thus, I limit the total number of negative samples by at most 30,000. A new model, called 'expanded' model, can trained on the initial set, plus a number of hard negatives. Most of the time, I find that the limit is reached, thus, if the hardware resource is enough, the performance can be improved a little. Otherwise, we can address this problem by using cascade architecture, of which the details will be shown in the next section.

```
expanded_svm_human_hog = hard_negative_mining_hog(svm_human_hog, human_window_size,
    human_hog_cell_size, human_hog_block_cell, X, y, images_neg_dir, 1);
```

The new expanded model is basically the same as the previous one. Testing the new model can be done through the same help function.

```
% EXPANDED HOG + LINEAR SVM MODEL
test_svm_classifier_hog(expanded_svm_human_hog, human_window_size, human_hog_cell_size,
    human_hog_block_cell, images_test_dir, human_label_path)
```

1.4 HOG and Cascade Classifier

The cascade architecture implemented in this work is based on the one proposed by Dalal et. al. [1]. At first, a list of more than 6,000 variable-sized blocks is created, as well as some hyper-parameters for training. The result model is also fine-tuned a little bit to make it work better.
My implementation supports two things: fine-tuning/growing from pre-trained model and hard negatives mining. The AdaBoost training algorithm in this work is implemented based on the version proposed by Viola et. al. [2] for their cascade face detector.

```
% Get list of all blocks
all_block_pos_human = get_all_var_block(12, human_window_size, human_hog_block_cell);

F_TARGET_human = 0.001; % Target false positive rate of final cascade classifier
f_MAX_human = 0.7;     % False positive rate per weak classifier
d_min_human = 0.9975; % Minimum acceptable detection rate per cascade node
n_block_human = 250;
max_node_human = 40;

% Train cascade classifier for human detection
[cascade_classifier_human_hog, cascade_threshold_human_hog] =
    train_cascade_classifier_hog(imp_list, imn_list, all_block_pos_human, ...
human_hog_block_cell, {}, [], images_neg_dir, n_block_human, max_node_human,
    F_TARGET_human, f_MAX_human, d_min_human);
```

Testing the cascade is done through the helper function 'test_cascade_classifier_hog(*)'.

```
% HOG CASCADE CLASSIFIER
```

```
test_cascade_classifier_hog(cascade_classifier_human_hog, cascade_threshold_human_hog,
    human_window_size, human_hog_block_cell, images_test_dir, human_label_path);
```

1.5 Simple LBP and Linear SVM

The LBP feature is also investigated in this work. I use MATLAB function 'extractLBPFeatures(*)' to extract LBP features from images, then use them to train Linear SVM classifier. The same set of positive and negative images used to train HOG and Linear SVM is used to train LBP and Linear SVM.

```
unrolled_lbp_feature_length = length(extractLBPFeatures(ones(human_window_size),
    'CellSize', human_lbp_cell_size));

% Initialize training data
% Use the same image list for training HOG + SVM model to train LBP + SVM one
X = zeros(n_imp + n_imn, unrolled_lbp_feature_length);
parfor i = 1:n_imp + n_imn % PARALLEL: 2x faster
   X(i,:) = extractLBPFeatures(imx_list{i},'CellSize',human_lbp_cell_size);
end

fprintf('Training SVM Classifier with LBP ... ');
svm_model = fitcsvm(X,y);
fprintf('done\n');
svm_human_lbp = [svm_model.Beta; svm_model.Bias];
```

Testing is done in almost the same manner as with HOG features.

```
% LBP + LINEAR SVM MODEL
test_svm_classifier_lbp(svm_human_lbp, human_window_size, human_lbp_cell_size,
    images_test_dir, human_label_path)
```

2 Face Detection

2.1 Setting Up Environment

Face detection flow is similar to human detection. There are some changes to the hyper-parameters for this task, which are declared at the beginning. For face detection, FDDB and COCO dataset are used. Since the formats of these datasets are different, I write corresponding helper functions to import data to the workspace. These functions are 'prepare_data_coco(*)', 'prepare_data_fddb_train_pos(*)', 'prepare_data_fddb(*)'. Also, I find that the provided annotations is unusual. The author of FDDB dataset uses ellipse bounding curves instead of rectangular boxes. So at first, I have to manipulate the ellipse annotations through the helper function 'ellipse_to_rect_box(*)'. This function is implemented based on the work of xlljoy [3].

```
% FDDB Dataset
images_fddb_dir = '../../datasets/FDDB/';
folds_dir = '../../datasets/FDDB/FDDB-folds/';

% Ground-truth labels
face_label_path = 'face_gt.txt';

% Detection window settings
face_window_size = [64 64];
face_window_stride = [4 4];

% HOG Settings
face_hog_cell_size = [8 8];
face_hog_block_cell = [2 2];
```

```
% LBP Settings
face_lbp_cell_size = [16 16];

if (MODE == 2)
   % Load the pre-trained model
   load('model.mat');
end
```

2.2 Training Face Detectors

The same scheme used to train human detectors is used to train face detector. The training set includes 8,272 positive and 20,000 negative images of size 64 by 64. I train a simple HOG and Linear SVM model at first, then apply Hard Negatives Mining technique, with a maximum of 64,000 negative images, on the result model to improve the performance. A cascade classifier using HOG is also tried out. Finally, LBP features are used in place of HOG ones to evaluate the effect.
The Hard Negative Mining, as described above however, did not work as expected. The second trial, I decreased the number of maximum negative images to 30,000 images and find the new model works much better. For the cascade one, it performs poorly because the number of nodes is still too low, only 7. I did not have enough time to train more.
One thing to note on the negative set of this task, COCO, is that the images in this set are quite large. I select randomly a part of it, then crop a number of regions in each of them instead to speed up creating training data.

```
% Prepare positive samples
train_set = prepare_data_fddb(folds_dir, 1:8, 1);
[imp_list, n_imp] = prepare_data_fddb_train_pos(images_fddb_dir, train_set,
    face_window_size);

% COCO Dataset
% Since the images in COCO Dataset are pretty large,
% it is better that we sample only a part of the set,
% and get multiple samples from each of them
images_coco_dir = '../../datasets/COCO/train2014/';
[imn_list, n_imn] = prepare_data_coco(images_coco_dir, face_window_size, 4000, 5);

% Training data
imx_list = [imp_list; imn_list];
y = [ones(n_imp,1); zeros(n_imn,1)];
unrolled_hog_feature_length = length(extractHOGFeatures(ones(face_window_size),
    'CellSize', face_hog_cell_size, 'BlockSize', face_hog_block_cell));
X = zeros(n_imp + n_imn, unrolled_hog_feature_length, 'single');

parfor i = 1:n_imp + n_imn
   X(i,:) = extractHOGFeatures(imx_list{i}, 'CellSize', face_hog_cell_size, 'BlockSize',
       face_hog_block_cell);
end

fprintf('Training SVM Classifier with HOG ... ');
svm_model = fitcsvm(X,y);
fprintf('done\n');
svm_face_hog = [svm_model.Beta; svm_model.Bias];

expanded_svm_face_hog = hard_negative_mining_hog(svm_face_hog, face_window_size,
    face_hog_cell_size, face_hog_block_cell, X, y, images_coco_dir, 1);
```

```
all_block_pos_face = get_all_var_block(2, face_window_size, face_hog_block_cell);

F_TARGET_face = 0.001; % Target false positive rate of final cascade classifier
f_MAX_face = 0.5;     % False positive rate per weak classifier
d_min_face = 0.9975; % Minimum acceptable detection rate per cascade node
n_block_face = size(all_block_pos_face,1);
max_node_face = 40;

[cascade_classifier_face_hog, cascade_threshold_face_hog] =
    train_cascade_classifier_hog(imp_list, imn_list, all_block_pos_face, ...
face_hog_block_cell, {}, [], images_coco_dir, n_block_face, max_node_face, F_TARGET_face,
    f_MAX_face, d_min_face);

unrolled_lbp_feature_length = length(extractLBPFeatures(ones(face_window_size),
    'CellSize', face_lbp_cell_size));
X = zeros(n_imp + n_imn, unrolled_lbp_feature_length);

parfor i = 1:n_imp + n_imn
   X(i,:) = extractLBPFeatures(imx_list{i},'CellSize',face_lbp_cell_size);
end

fprintf('Training SVM Classifier with LBP ... ');
svm_model = fitcsvm(X,y);
fprintf('done\n');
svm_face_lbp = [svm_model.Beta; svm_model.Bias];
```

Testing also shares the same flow as of human detection.

```
% HOG + LINEAR SVM MODEL
test_svm_classifier_hog(svm_face_hog, face_window_size, face_hog_cell_size,
    face_hog_block_cell, images_fddb_dir, face_label_path)

% EXPANDED HOG + LINEAR SVM MODEL
test_svm_classifier_hog(expanded_svm_face_hog, face_window_size, face_hog_cell_size,
    face_hog_block_cell, images_fddb_dir, face_label_path)

% HOG CASCADE CLASSIFIER
test_cascade_classifier_hog(cascade_classifier_face_hog, cascade_threshold_face_hog,
    face_window_size, face_hog_block_cell, images_fddb_dir, face_label_path);

% LBP + LINEAR SVM MODEL
test_svm_classifier_lbp(svm_face_lbp, face_window_size, face_lbp_cell_size,
    images_fddb_dir, face_label_path)
```

3 Results

The specification of all detectors is shown in Table 1 and below figures.

4 Discussion and Proposal

4.1 Processing Speed

Theoretically, the cascade architecture should reduce the the processing time in comparison with other methods. However, I find that it depends on the implementation of feature extraction algorithm. As observed, assuming a block of 16 by 16 is used and overlapping between them is 8 by 8, 'extractHOGFeatures(*)' automatically scan all 105 blocks of the human detection window in around 0.0015 seconds while calling

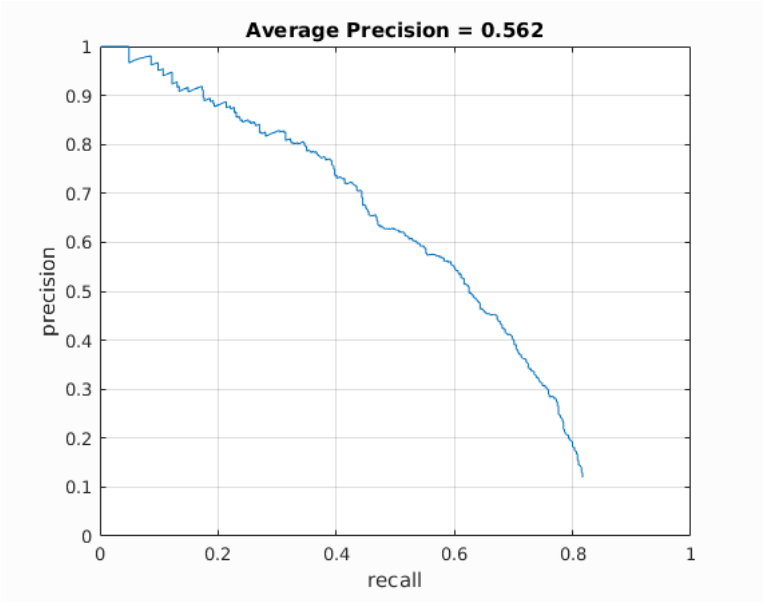

Figure 1: HOG + SVM Human: Average Precision

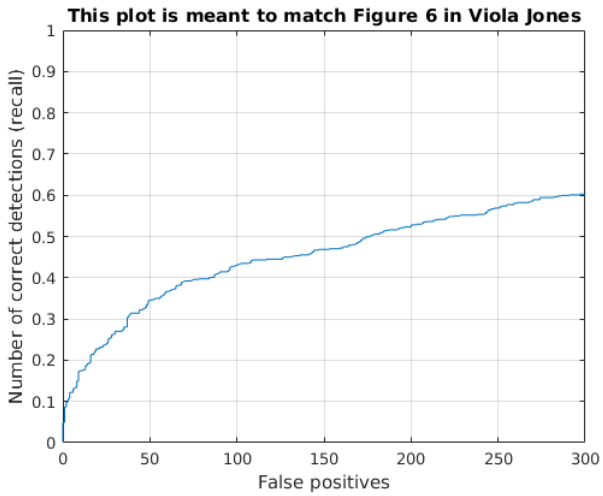

Figure 2: HOG + SVM Human: Confidence

Table 1: Model Specification

Model	Hard Negative	Cascade	Average Precision	Note
svm_human_hog			56.2	
expanded_svm_human_hog	×		61.9	
cascade_classifier_human_hog	×	×	63.9	17 nodes
svm_human_lbp			40.1	
svm_face_hog			41.0	
expanded_svm_face_hog	×		56.6	
cascade_classifier_face_hog	×	×	72.1	7 nodes
svm_face_lbp			-	

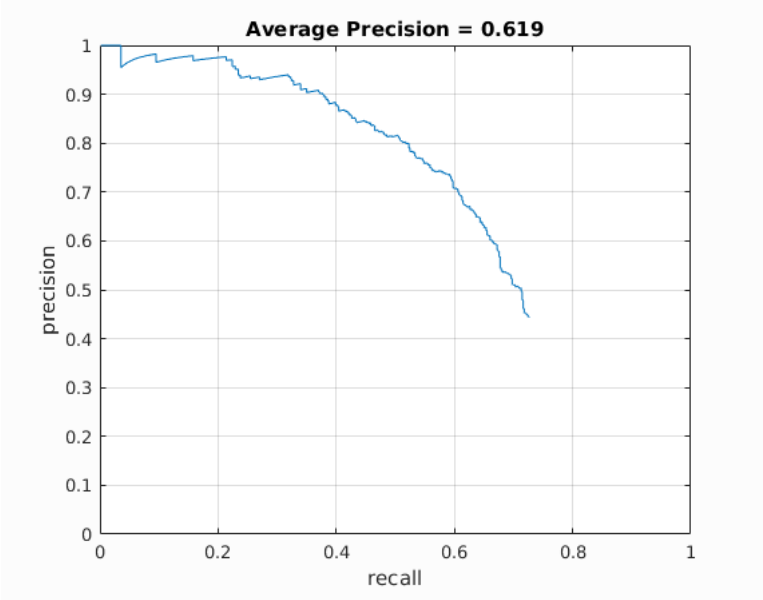

Figure 3: Expanded HOG + SVM Human: Average Precision

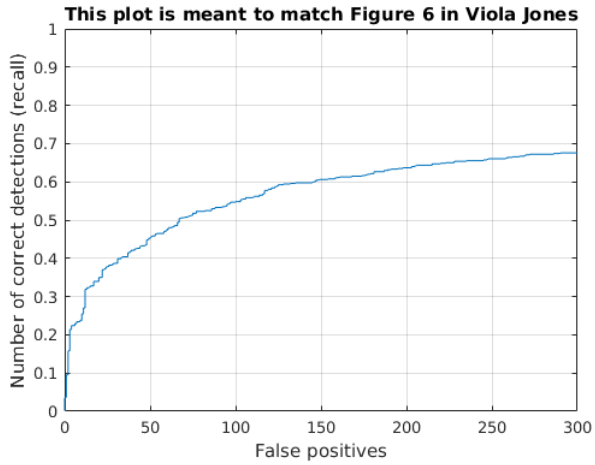

Figure 4: Expanded HOG + SVM Human: Confidence

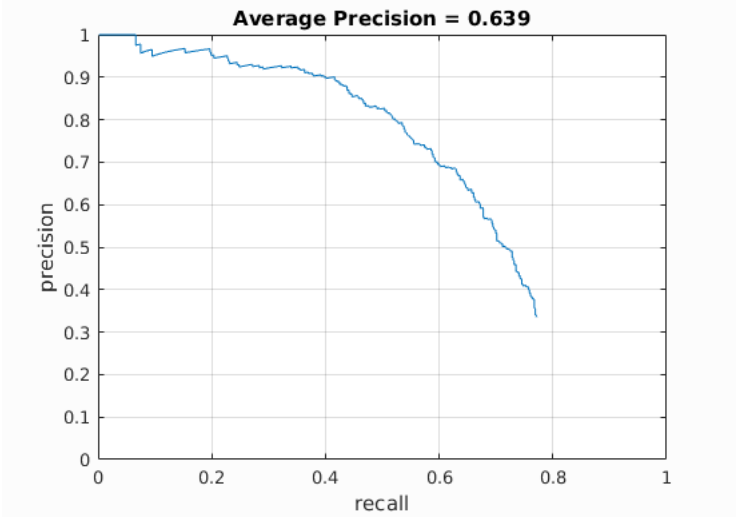

Figure 5: Cascade HOG Human: Average Precision

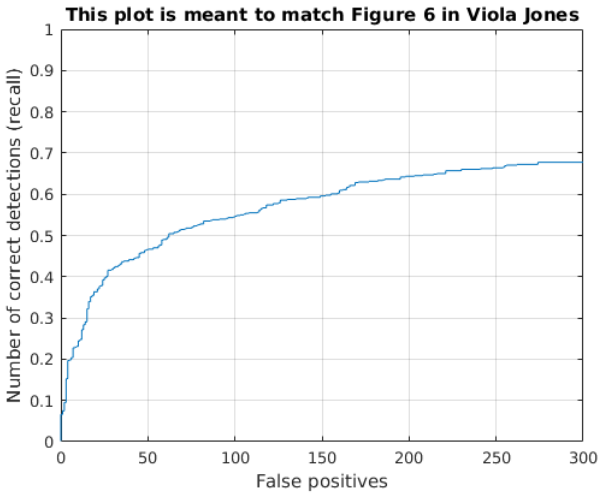

Figure 6: Cascade HOG Human: Confidence

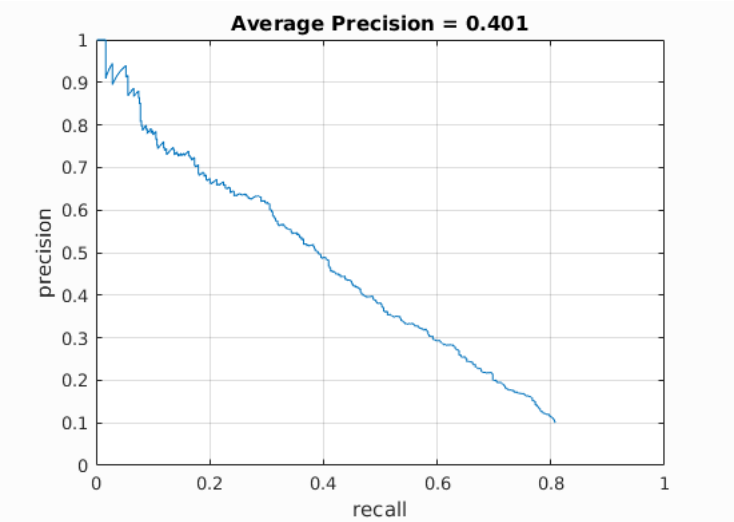

Figure 7: LBP + SVM Human: Average Precision

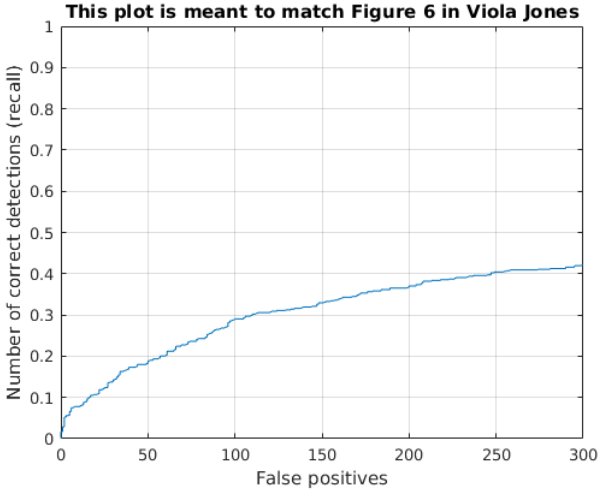

Figure 8: LBP + SVM Human: Confidence

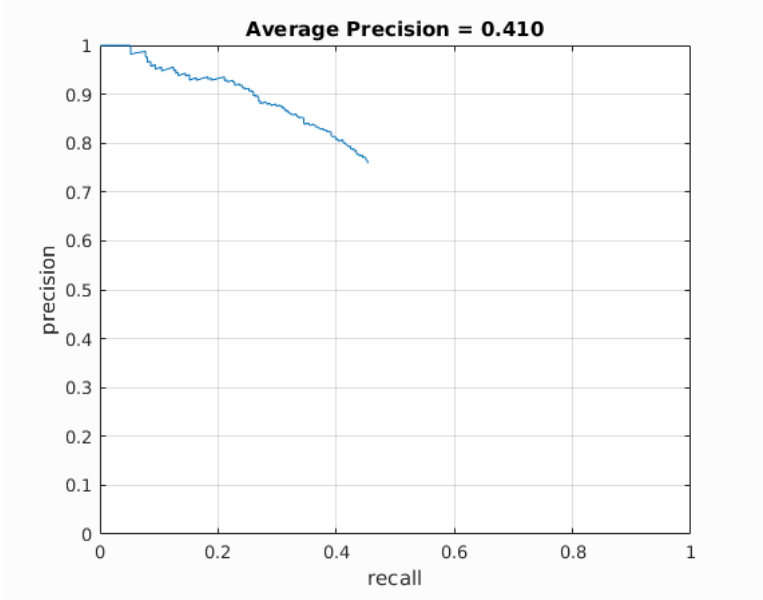

Figure 9: HOG + SVM Face: Average Precision

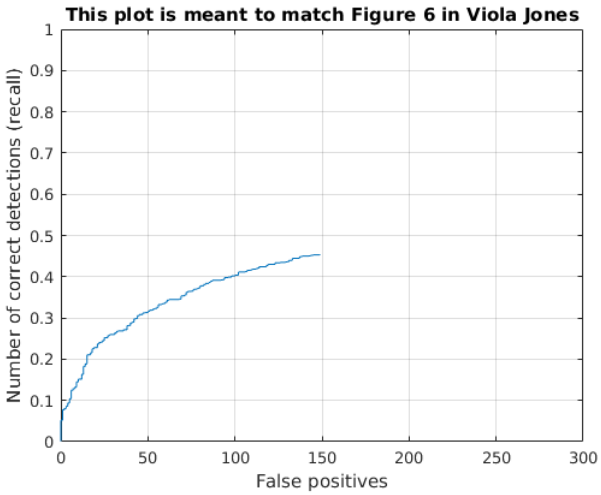

Figure 10: HOG + SVM Face: Confidence

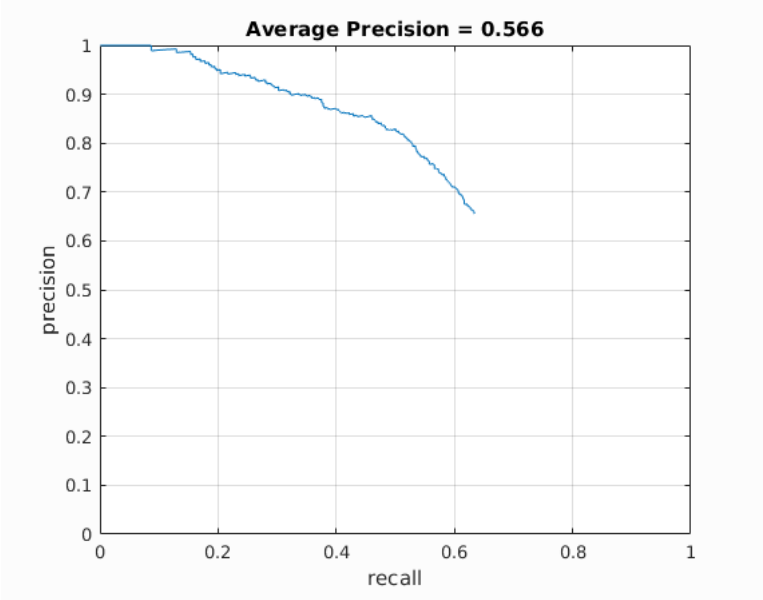

Figure 11: Expanded HOG + SVM Face: Average Precision

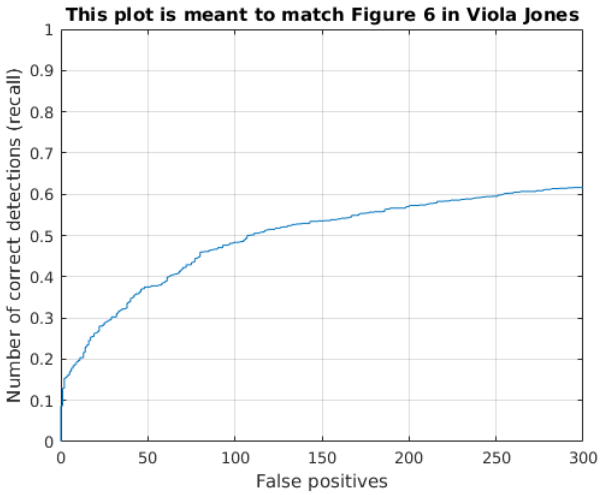

Figure 12: Expanded HOG + SVM Face: Confidence

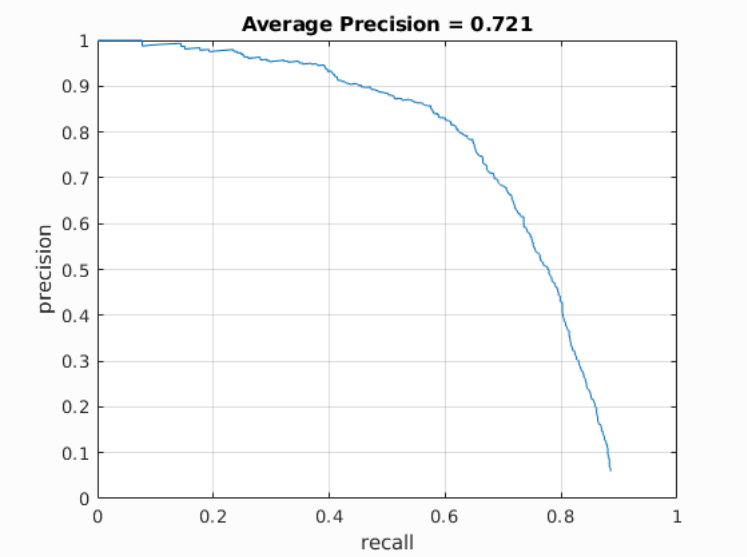

Figure 13: Cascade HOG Face: Average Precision

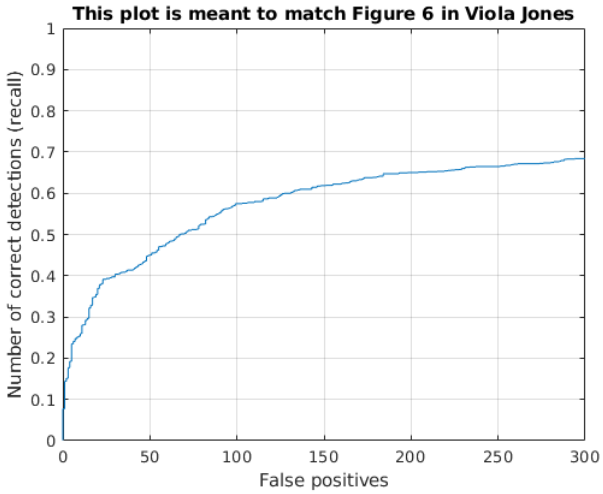

Figure 14: Cascade HOG Face: Confidence

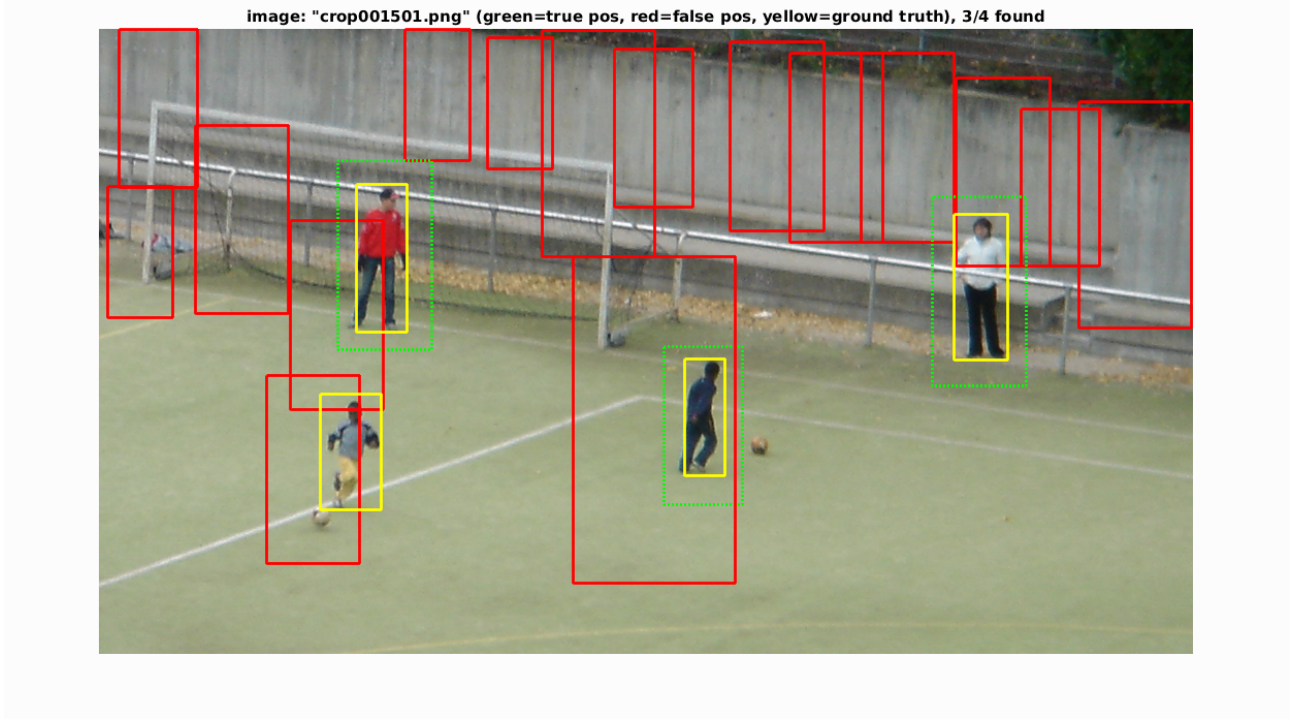

Figure 15: Simple HOG and Linear SVM for human detection

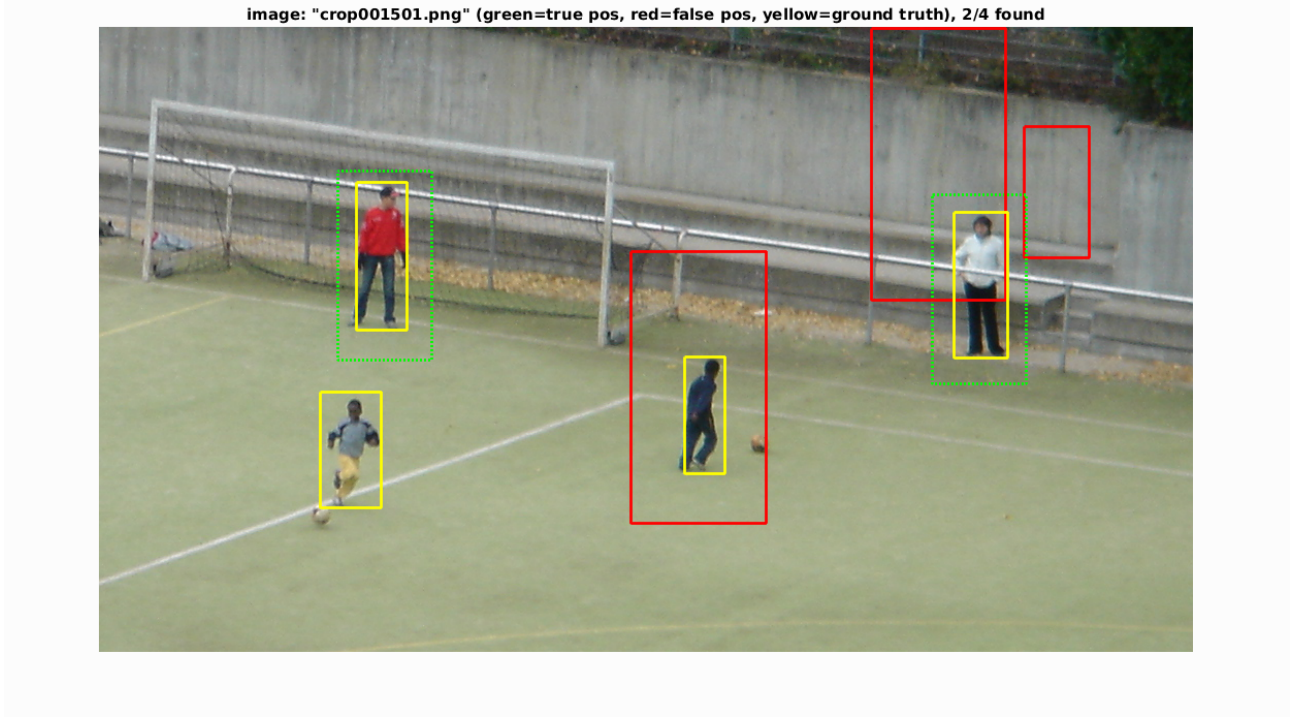

Figure 16: Expanded Simple HOG and Linear SVM for human detection

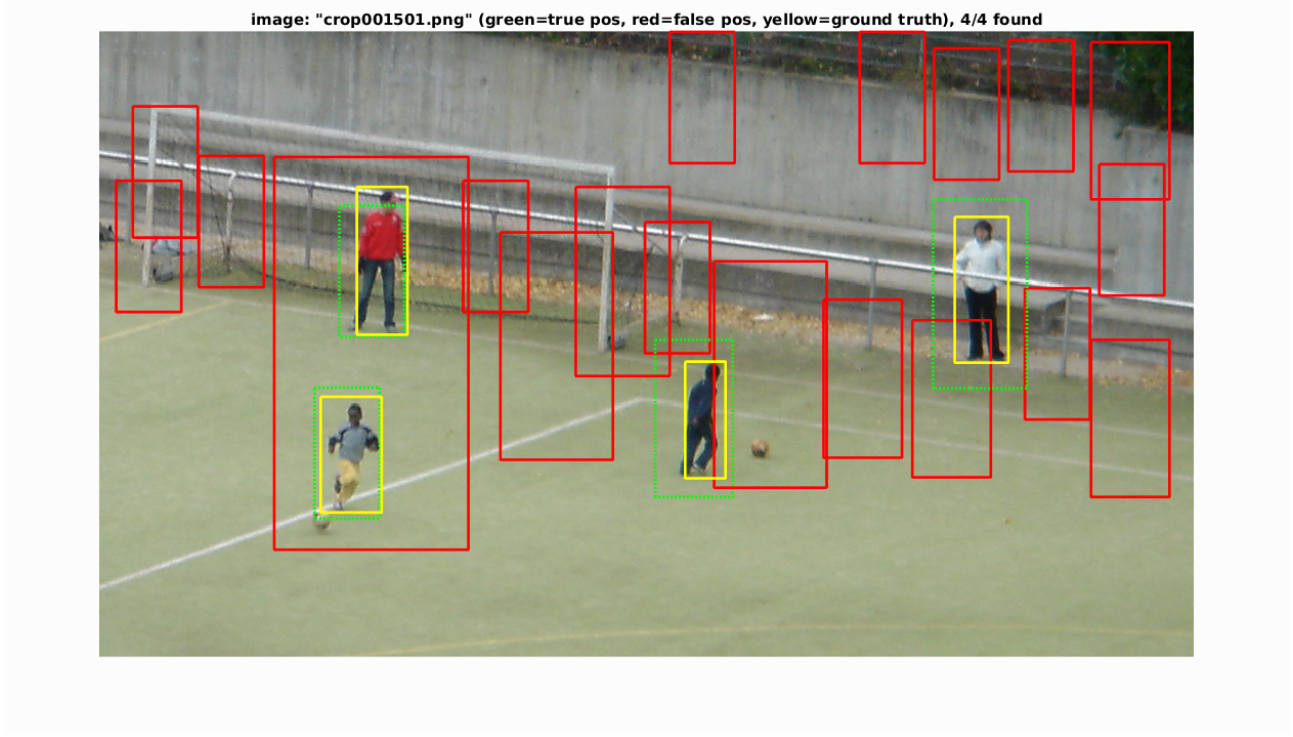

Figure 17: Cascade HOG for human detection

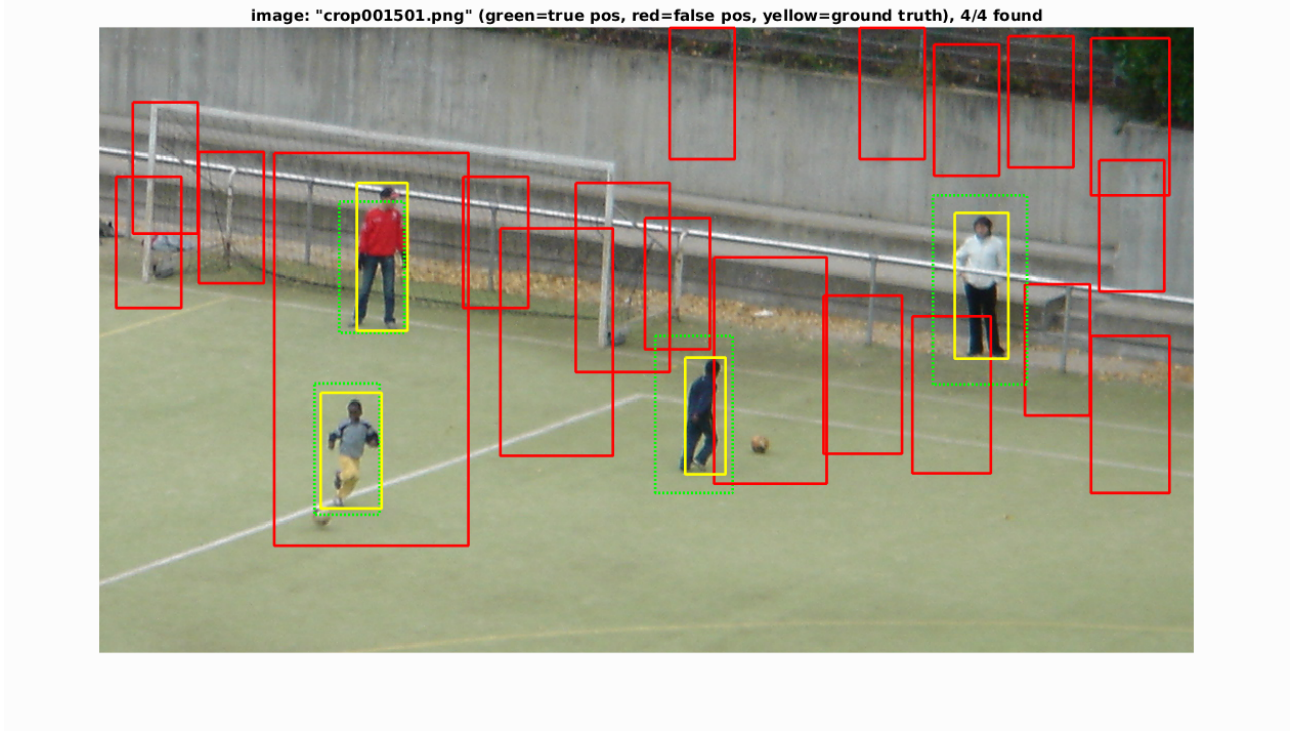

Figure 18: Simple LBP and Linear SVM for human detection

this function 105 times on a single block can take upto 0.11 seconds. This problem becomes clear in the implementation of cascade architecture, as in every stage, we need to extract (likely) multiple random regions. In detail, calling the function for every region as described spend approximately 20 times more time.
To address this problem (for this specific MATLAB implementation), I use 'concatenated image'. Concretely, given N detection windows being tested, for each block, I crop the corresponding region in each sample and concatenate them into one 'image'. When calling 'extractHOGFeature(*)', I apply an overlap of 0 pixel (or non-overlapping), then reshape the result to the correct size. That is to say, the number of calls to extraction function is drastically reduced and the processing speed is increased by approximately 6 times.
Another way to speed up feature extraction is to apply parallel computing. In this work, I use 'parfor', which speed the process by 2 times (4 workers). Using 'parfor' requires in-advance allocation, thus, not every for loop can be replaced by this.

```
region = block(:,nb);
unrolled_sample = [];
for ns = 1:n_samples
unrolled_sample = [unrolled_sample samples{ns}(region(1):region(2),region(3):region(4))];
end
X = extractHOGFeatures(unrolled_sample,'CellSize',[region(5) region(6)], 'BlockSize',
    block_cell, 'BlockOverlap', [0 0]);
X = reshape(X, feature_length, n_samples);
X = X';
```

4.2 Average Precision

It can be noticed from Table 1 that the simple HOG + SVM performs quite well compared to others. However, it still gives many false positives. Applying Hard Negative Mining technique is recommended to eliminate this problem. Assuming the same number of training negative samples are used, I find it better to start with a relatively small set, and add more samples via Hard Negative Mining. The limitation of this method is the hardware resource. We need to make the whole training set fit to the workspace, otherwise, it will raise the out of memory error.
Applying Hard Negative Mining to cascade architecture effectively makes full use of big training set. Memory problem can be avoided by constraining the number of samples for each node. After training one node, a part of the current negative set is filtered, so we can safely add more 'harder' samples to it until we reach the limit again.
One thing that may improve the performance of detector is to use multiple feature types, for example: HOG + LBP. However, I have not tried this approach yet.

5 Conclusion

In this work, I present my implementation of using HOG feature for human and face detection tasks. Various technique such as Hard Negative Mining and Cascade Architecture are investigated to improve the performance. LBP feature is also tested. The best average precision for human detection task is 63.9% by using cascade HOG detector with 17 nodes. For face detection task, I achieve the highest of 56.6% by using the expanded HOG and SVM model. In order to improve the processing speed, I use parallel computing method for feature extraction and 'concatenated image' trick to reduce the number of function calls. The 'concatenated image' trick is specifically applied for the MATLAB function 'extractHOGFeatures(*)' only, however.

References

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, pp. 886–

893, IEEE, 2005.

[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, vol. 1, pp. I–I, IEEE, 2001.

[3] "https://github.com/xlljoy/fddb_ellipse2rect."