ICE3014 (41) – Multimedia Engineering

Final Assignment Report

Face Detection

Professor Yi Juneho

2010311484 – 권영대 –반도체공학과

2013313934 – 박성욱 –전자전기공학부

2014318416 – 배수민 – 컴퓨터공학과

Table of contents

1 Abstract

With an ever growing number on different fields of applications of face detection, be it a high grade security system in public environments, or simply a face recognition to tag people on Instagram, in this day and age, multimedia engineering is becoming more and more important. Many algorithms and approaches for image analysis exist, and none of them are yet perfect, as standardization is quite hard to achieve, for example misrecognition, detection speed and reliability are, even in the most modern approaches still quite a challenge.

However the concurrent technology used in the research community combines different approaches in several steps, leading to quite robust results. By moving a “recognition window” over an image and dividing it into segments (Sliding Window Face Detection), allows us to recognize patterns and sub-patterns of faces, to combine them in a texture histogram for each block, thus enables to distinguish between non-face objects (negative features) and actual face objects (positive features). Summarizing the results in a learning curve using a database, where successful face-detections from training images are stored, which then can be used on a testing image, is what we utilized for this project, implementing Histogram of Oriented Gradients (HOG) and Support Vector Machine (SVM) to process images for face detection.

2 Method of Multi-Scale Processing

In this project, we implemented Multi-Scale Face Detection System with HOG features and SVM classifier. Our method of Multi-Scaled Face Detection System consists of the following steps:

1. Extracting HOG features from positive and negative training examples.
2. Train linear SVM classifier with the extracted HOG features.
3. Compute HOG features in the test dataset.
4. While sliding the detection window from left to right and from top to bottom over the target image, detect faces by generating bounding boxes around faces which produce certain matches greater than the threshold value.
5. Scale an image until it becomes smaller than the sliding window size.
6. Apply non maximum suppression on bounding boxes in the image, in order to increase the performance by removing less confidentially overlapping boxes.

The image data came from Caltech Web Faces, the SUN scene database and CMU-MIT database. For positive training examples, we utilized Caltech Web Faces which contains 6,713 cropped 36x36 faces. We used the SUN scene database for negative training examples which were random patches sampled from images of SUN scene database.

3 Codes

Since most of the code base for the face detection project is provided, especially for project.m, we will only introduce the code which we implemented in this project. Our code is as follows.

3.1 Project.m

In project.m, we changed parameters for HOG and SVM in order to boost the performance of face detection. For the HOG parameter, we changed ‘hog_cell_size’ from 2 to 9 such as 2, 3, 6, and 9. We were able to get high performance as we decreased the ‘hog_cell_size’ from 9 to 2.

For the SVM parameter, we tested 2 lambda values such as 0.0001 and 0.00001, and could get a slightly higher performance when we used 0.00001 instead of 0.0001.

Code

```

%%%%% Set Hog Parameter
feature_params = struct('template_size', 36, 'hog_cell_size', 6);

%%%%% Train Classifier.
[w, b] = vl_svmtrain(X', Y', 0.00001);

%%%%% Run detector on test set.
%%%%% @params test_scn_path Indicate Test file path
%%%%% @params w trained weights by using svm
%%%%% @params b trained bias by using svm
%%%%% @params feature_params HOG feature parameters ('template_size', 36, 'hog_cell_size', 6)
%%%%% rescale each step of your multiscale detector
%%%%% threshold for a detection
[bboxes, confidences, image_ids] = run_detector(test_scn_path, w, b, feature_params);

```

get_random_negative_features.m

If we want to change the 'hog_cell_size' parameter in "project.m", we need to change some parts of codes in "get_random_negative_features.m", to automatically adapt to the change of 'hog_cell_size' in "project.m". By changing the code in "get_random_negative_features.m" as shown below, when changing the parameter in "project.m" it becomes automatically applied in "get_random_negative_features.m".

Code

```

%%%%% the number of hog cells in 1 hog template
window_size_cell = (feature_params.template_size / feature_params.hog_cell_size);
for y=1:(temp_size(1) - (window_size_cell-1) )/freq
  for x=1:(temp_size(2) - (window_size_cell-1) )/freq
    xStart = x*freq;
    yStart = y*freq;
    xEnd = xStart + (window_size_cell - 1);
    yEnd = yStart + (window_size_cell - 1);

```

3.2 run_detector.m

Code

```

%%%%% @params test_scn_path Indicate Test file path
%%%%% @params w trained weights by using svm
%%%%% @params b trained bias by using svm
%%%%% @params feature_params HOG feature parameters ('template_size', 36, 'hog_cell_size', 6)
%%%%% rescale each step of your multiscale detector
%%%%% threshold for a detection
function [bboxes, confidences, image_ids] = ....
    run_detector(test_scn_path, w, b, feature_params)

test_scenes = dir( fullfile( test_scn_path, '*.jpg' ));

%%%%% Last containing variables for whole images
bboxes = zeros(0,4);
confidences = zeros(0,1);
image_ids = cell(0,1);

%%%%% parameters
template_size = feature_params.template_size;
hog_cell_size = feature_params.hog_cell_size;
window_size_cell = template_size / hog_cell_size;
x_step = 1;
y_step = 1;
threshold = 0.5;
D = (feature_params.template_size / feature_params.hog_cell_size)^2 * 31
input_feature = zeros([1 D]);

for i = 1:length(test_scenes)

    img = imread( fullfile( test_scn_path, test_scenes(i).name ));
    img = single(img)/255;
    if(size(img,3) > 1)
        img = rgb2gray(img);
    end

    %%%%% variables for 1 image
    scale_factor = 1.4;
    cur_bboxes = zeros(0, 4);
    cur_confidences = zeros(0,1);
    cur_image_ids = cell(0,1);
```

```
    %%%%% Loop : scale of image
    while 1
        temp_img = imresize(img, scale_factor);
        %%%%% if size of x, y axis of image is larger than template size,
        %%%%% then break
        if ( size(temp_img, 2) < template_size ) || ( size(temp_img, 1) < template_size )
            break;
        end
        %%%%% get hog feature
        hog = vl_hog(temp_img, feature_params.hog_cell_size);
        %%%%% parameters for 1 image
        x_size_cell = size(hog, 2);
        y_size_cell = size(hog, 1);
        x_pos_cell = 1;
        y_pos_cell = 1;
        %%%%% Loop : y axis
        while ( (y_pos_cell+window_size_cell-1) <= y_size_cell )
            %%%%% Loop : x axis
            while ( (x_pos_cell+window_size_cell-1) <= x_size_cell )

                %%%%% get window cell sized hog feature from original hog feature
                temp_hog = hog(y_pos_cell:y_pos_cell+window_size_cell-1, x_pos_cell:x_pos_cell+window_size_cell-1, :);
                %%%%% compute confidence of this window
                temp_confidence = sum(temp_hog(:).*w) + b;

                %%%%% if confidence is bigger than the threshold,
                %%%%% then get this window as a face
                if temp_confidence > threshold
                    temp_confidence;
                    %%%%% set bounding box
                    temp_bboxes = [x_pos_cell, y_pos_cell, x_pos_cell+window_size_cell-1, y_pos_cell+window_size_cell-1];
                    temp_bboxes = temp_bboxes * hog_cell_size / scale_factor;
                    %%%%% append bounding box to cur_bboxes
                    cur_bboxes = [cur_bboxes; temp_bboxes];
                    %%%%% append confidence to cur_confidences
                    cur_confidences = [cur_confidences; temp_confidence];
                    %%%%% append image name to cur_iamge_ids
                    cur_image_ids = [cur_image_ids; {test_scenes(i).name}];
                end

                x_pos_cell = x_pos_cell + x_step;
            end
            x_pos_cell = 1;
            y_pos_cell = y_pos_cell + y_step;
        end
        % scale_factor = scale_factor * scale_step
        scale_factor = scale_factor - 0.1;
    end

    %%%%%%
    [is_maximum] = non_max_supr_bbox(cur_bboxes, cur_confidences, size(img));

    cur_confidences = cur_confidences(is_maximum,:);
    cur_bboxes      = cur_bboxes(     is_maximum,:);
    cur_image_ids   = cur_image_ids(  is_maximum,:);

    bboxes      = [bboxes;      cur_bboxes];
    confidences = [confidences; cur_confidences];
    image_ids   = [image_ids;   cur_image_ids];

End
```

3.3 Test Cases

1. Cell size = 3 and we could get good results

image: "original1.jpg" (green=true pos, red=false pos, yellow=ground truth), 8/8 found

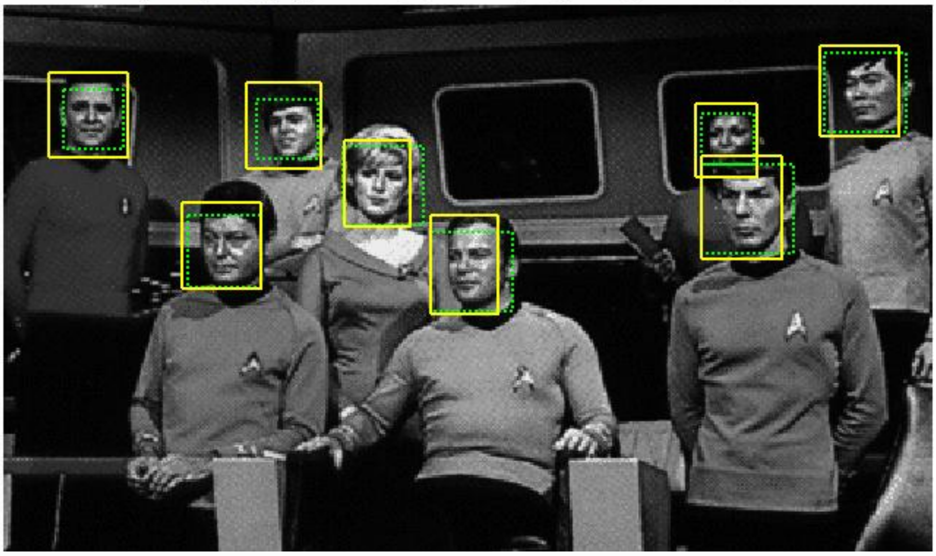


image: "trekcolr.jpg" (green=true pos, red=false pos, yellow=ground truth), 3/3 found

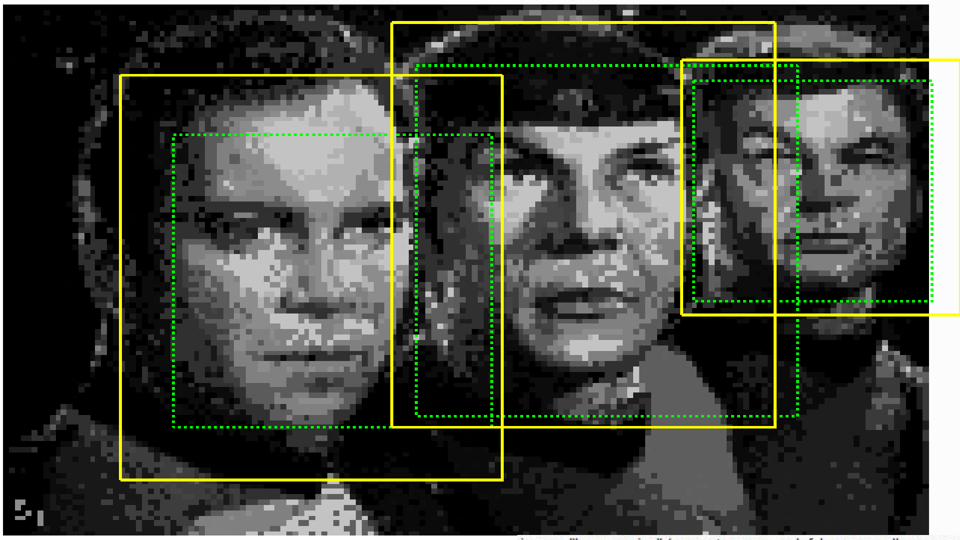


image: "audrey2.jpg" (green=true pos, red=false pos, yellow=ground truth), 1/1 found

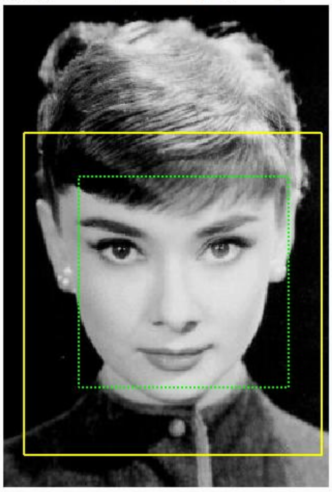
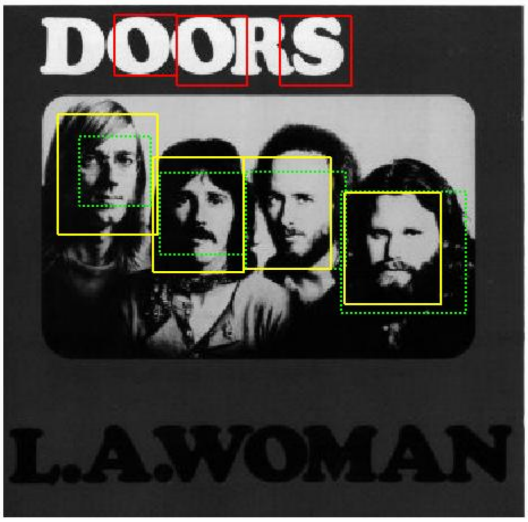
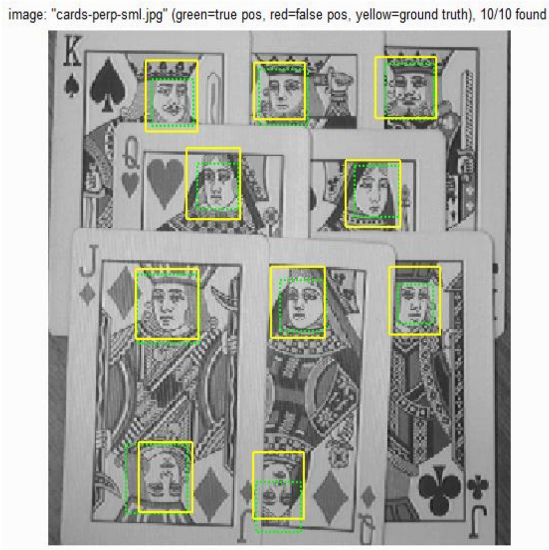
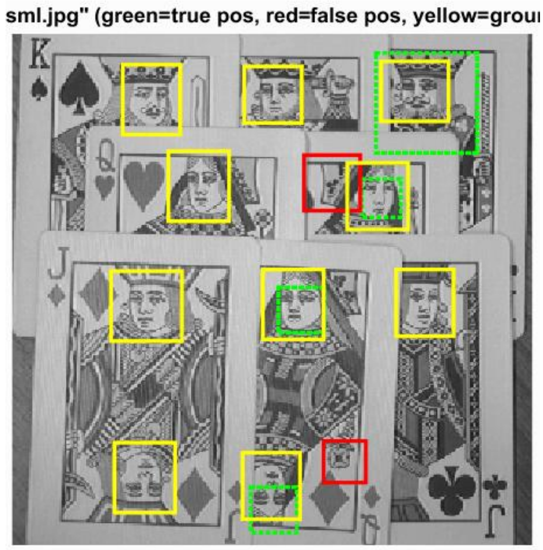
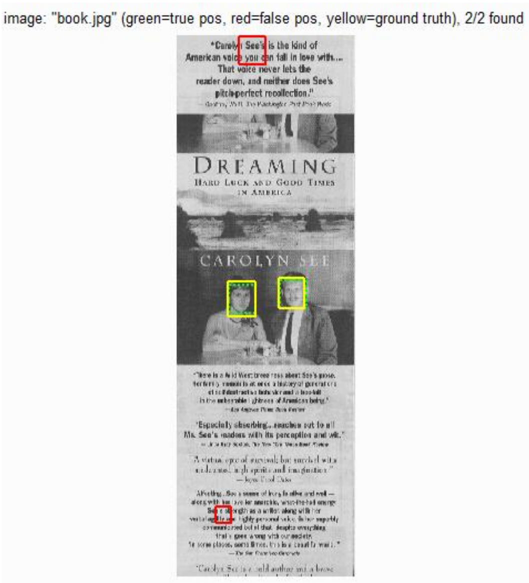
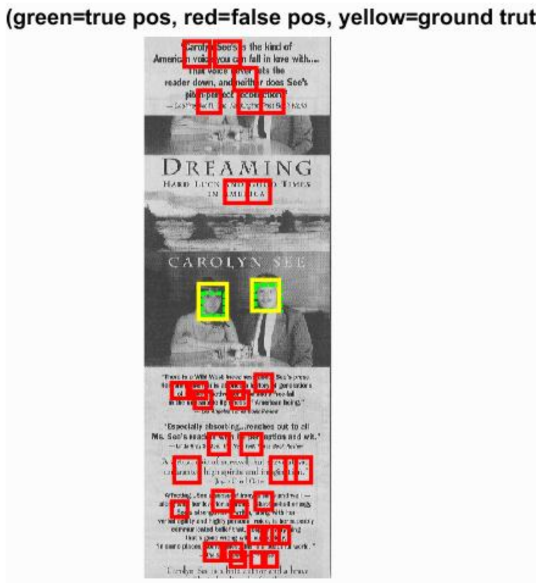
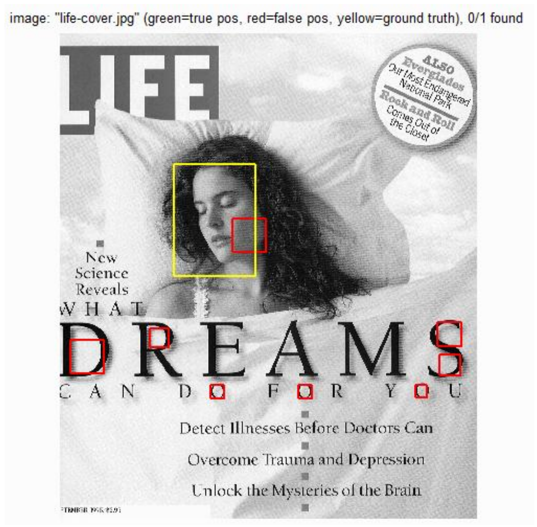


image: "lawoman.jpg" (green=true pos, red=false pos, yellow=ground truth), 4/4 found

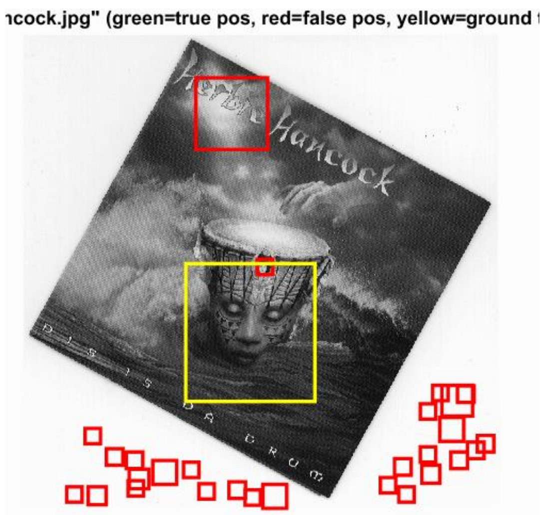


2. Special cases that results showed many differences depending on cell size

<table>
<tr><td>Cell size = 3</td><td>Cell size = 6</td></tr>
<tr><td colspan="2">Cell size and Detection Results</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td colspan="2">For 4 pictures below, you can see that the number of false positives that are related to Words is decreased at Cell size = 3</td></tr>
<tr><td>
</td><td>
</td></tr>
</table>



In this picture, we could not get good result in any ways.



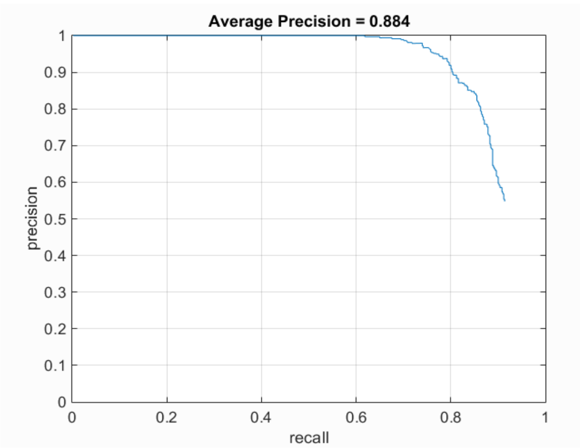
Usually, spelling O and S or something alike, is often recognized as a face. For the last picture, we couldn't detect it a face in any condition, which is a pretty special case. The most suggested explanation is because the face is tilted.

4 Results

A precision-recall curve for the CMU+MIT test set. You can learn more about precision-recall curves on the web here.

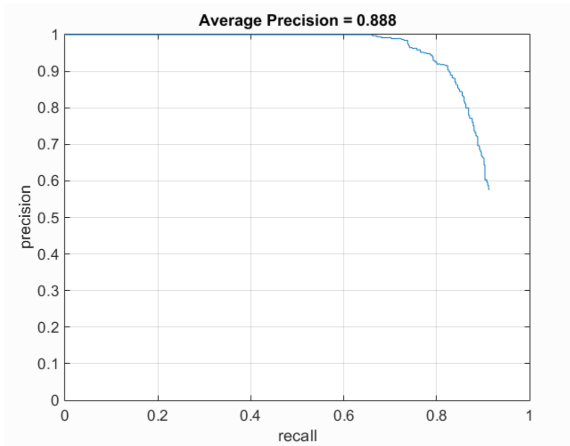
1) Lambda = 0.00001

Cell size = 2



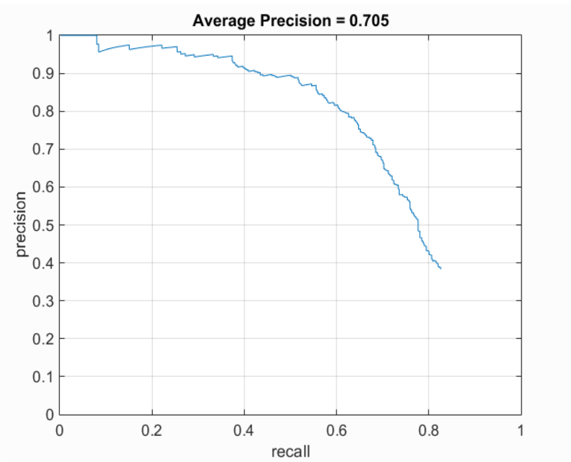
AP=0. 884

Cell size = 3



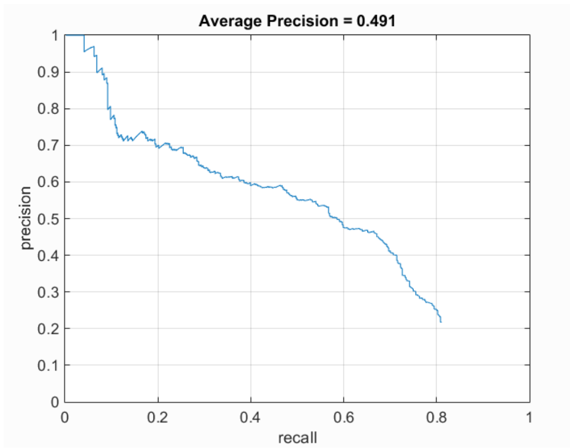
AP = 0.888

Cell size = 6



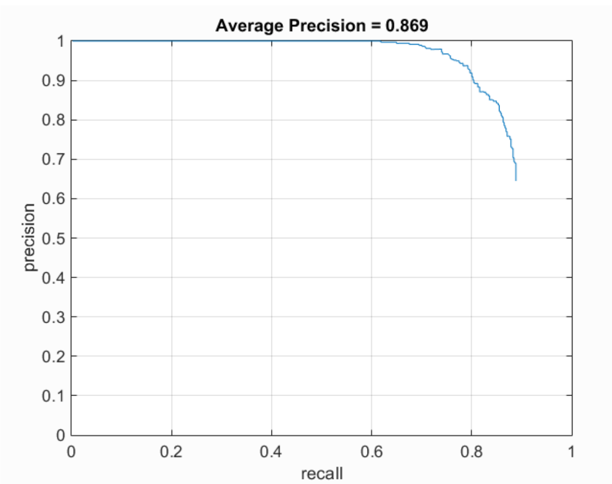
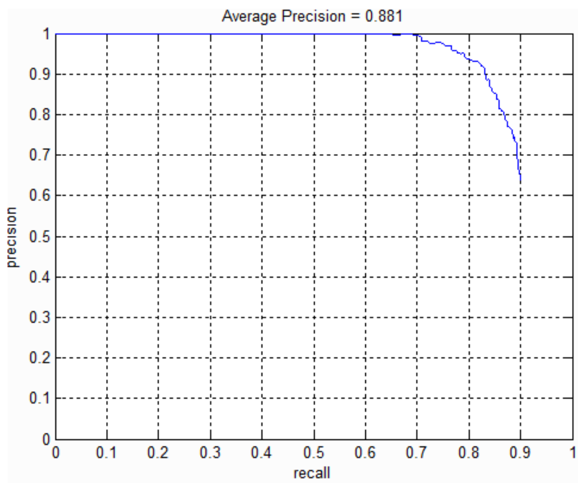
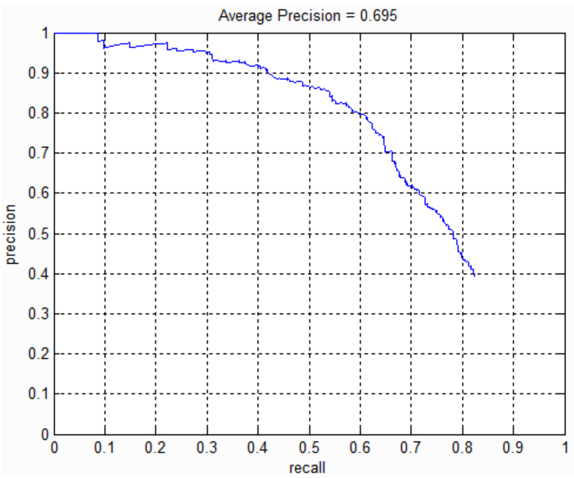
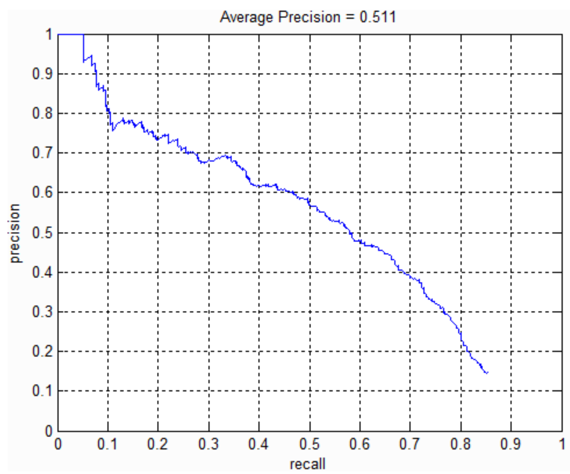
AP = 0.705

Cell size = 9



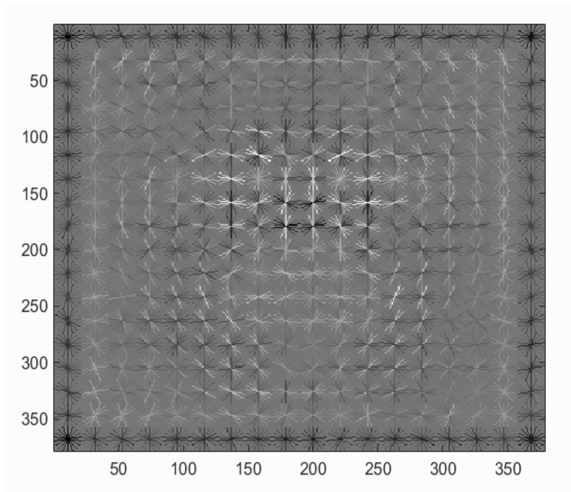
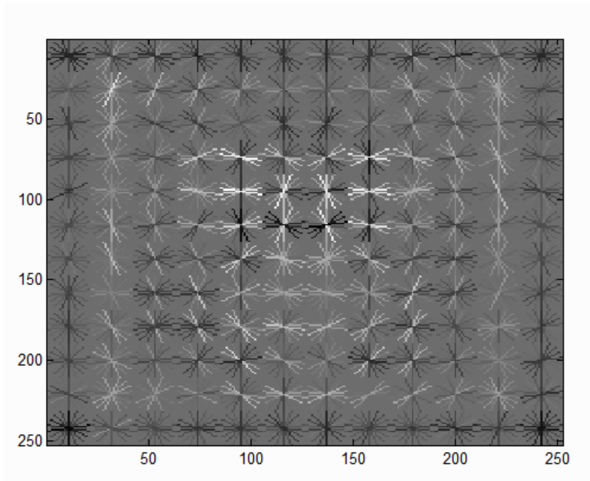
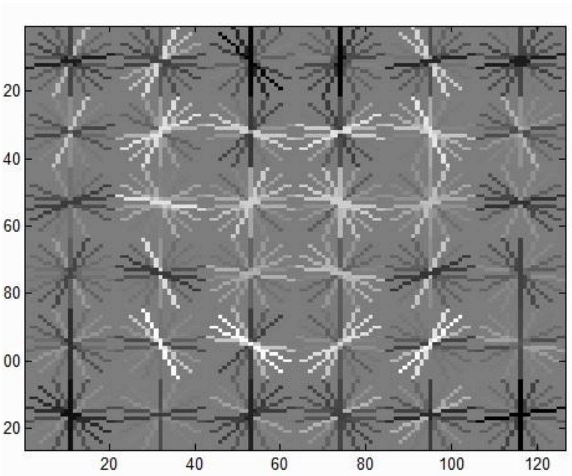
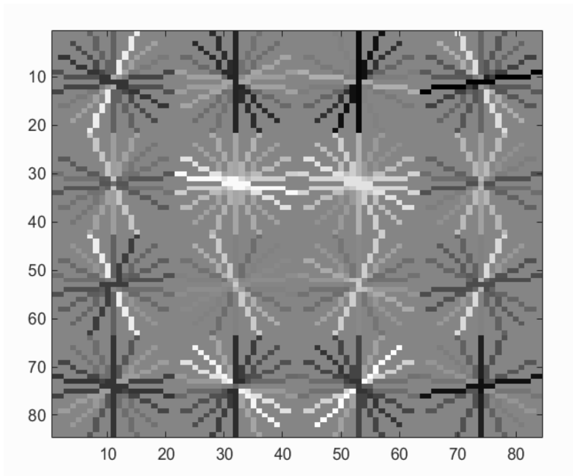
AP = 0.491

2) Lambda = 0.0001

Cell size = 2	Cell size = 3
 AP =0.869	 AP = 0.881
Cell size = 6	Cell size = 9
 AP = 0.695	 AP = 0.511

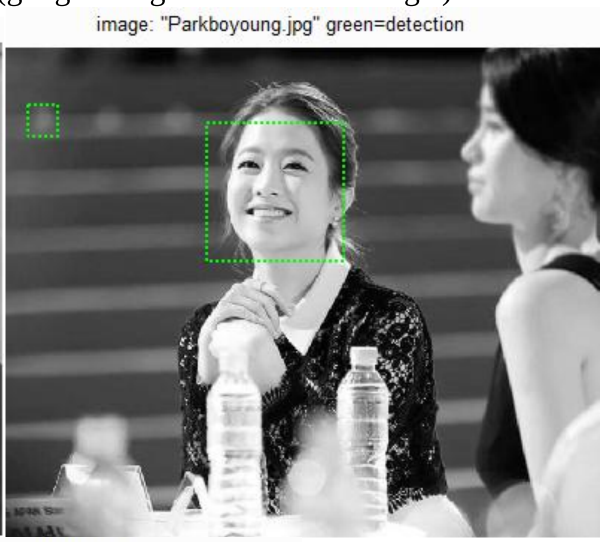
3) HOG

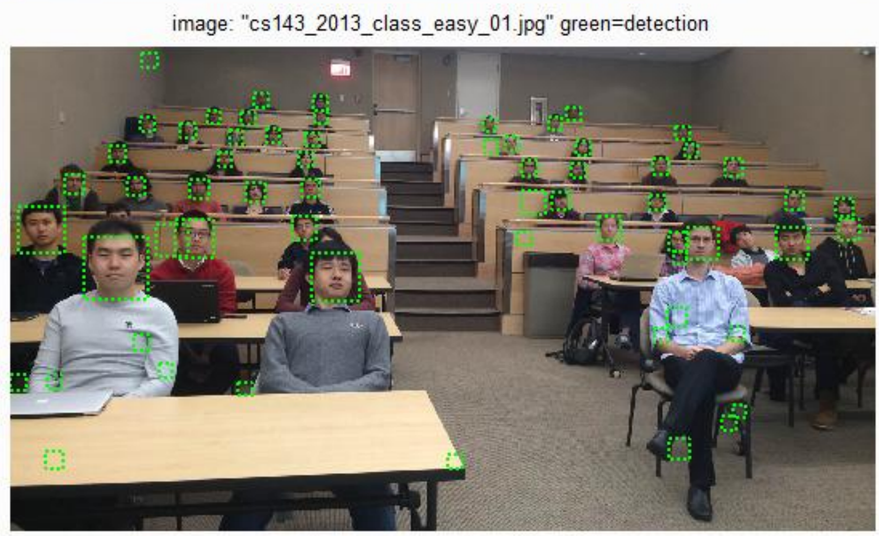
The HOG features and the detection results visualized.

<table>
<tr><td>Cell size = 2</td><td>Cell size = 3</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>Cell size = 6</td><td>Cell size = 9</td></tr>
<tr><td>
</td><td>
</td></tr>
</table>

Using smaller cells, we can intuitively find that it looks much more like a face of a person.

· 4) Detection results on our own test images (google image and our own image)





We can see that shaded faces or occluded faces are harder to detect than normal faces, but the hardest one are tilted faces, because for computers, there is almost no correspondence to normal faces.

5 Discussion

We experimented with Multi-Scale Face Detection and different threshold values, lambda and hog cell sizes. Depending on these values, we could get various results. With the given cell size 6, we could get an AP of about 0.7. With the threshold value of 0.4 and a lambda value of 0.00001, produced the highest performance with cell size 6. It was not a satisfying result, so we tried different cell sizes. For using a different cell size, we had to fix some codes in the "get_random_negative_features.m" file, because we had to train based on the new cell size to get the right w and b values.

We have tested our face detection algorithm several times to make differences on cell size, threshold and lambda value. The first improved results that we found was with the cell size case. The cell size is the dominant factor that makes the PR curve better. With cell size 3, we could get the highest AP for our code. We found that there exists a trend that decreasing cell size enhances the performance of the Face Detection system. However, we could not test many times with cell size 2, because making the cell size 2 required too much of testing time and did not have much difference to cell size 3.

The threshold value was also a big factor. With high a threshold value, we could get rid of many false-positives. But when the threshold value is too high, the effect of increased true-negative detection gets bigger, which produces a lower AP. When we used our own pictures as an additional test data, we needed a higher threshold value. We assume that it is, because usually pictures from our camera are more clear and distinct. With clear and distinct pictures, it can more distinctly detect, whether it is face or non-face, thus a higher threshold value produces less false positives features with the true positives remaining.