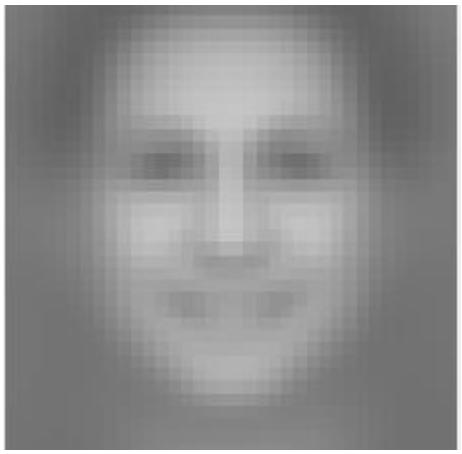
COMP408 Assignment 3, Prepared by Arda Kırkağaç, 49799

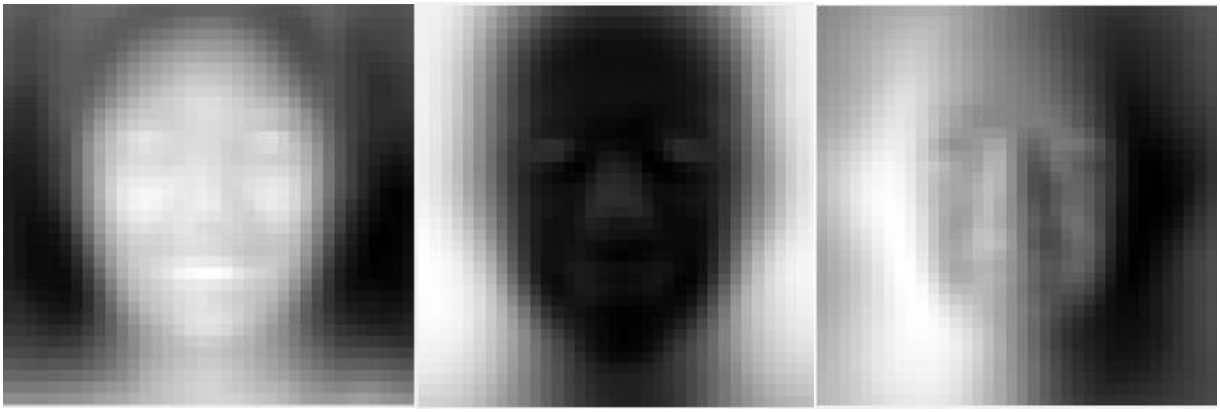
In this assignment, our task was to implement eigenface technique over a pipeline, which already generates good results using Histogram of Oriented Gradients features on face detection. We have positive and negative training data, which we feed to a classifier later, to learn about the feature differences between a face and a non-face.

We have 6,713 cropped faces with 36x36 resolution. I processed them one by one, expanding them as a vector, and generated a mean face to normalize our training and test data for the rest of our pipeline. Below, you can see the mean face, which is calculated by summing over all pixel values for each cropped face, and then dividing them by the total number of faces.



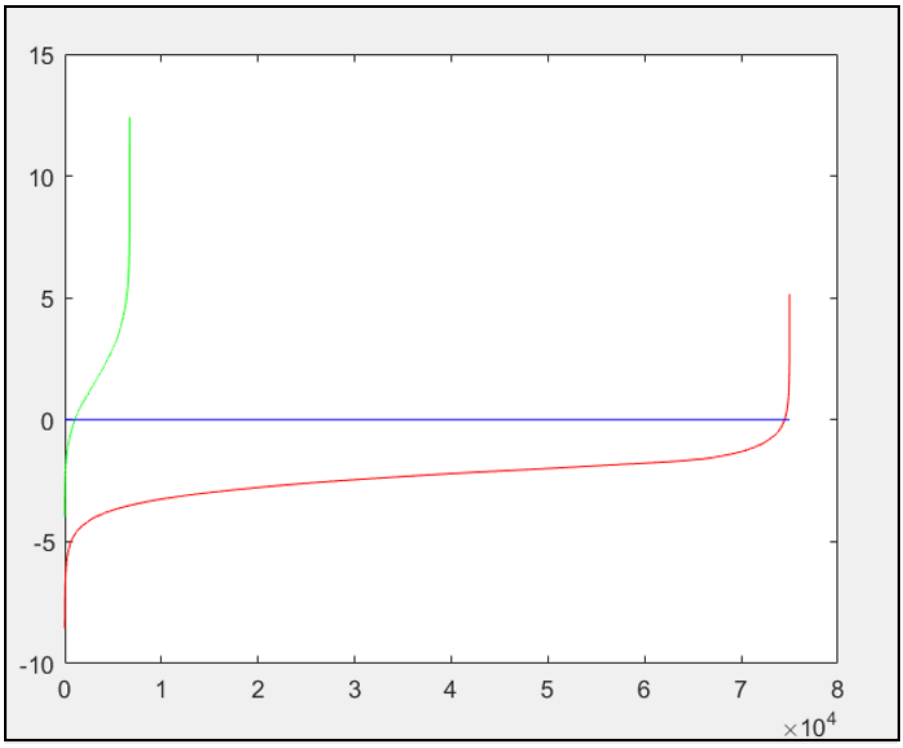
Then, after normalizing our positive training data with the mean face, I calculated eigenfaces by using this normalized matrix to create a covariance matrix. Then, thresholding procedure is applied to keep M eigenvectors (called eigenfaces hereafter) with highest eigenvalues. With a threshold of 0.9, M turned out to be 48, while it is 74 with a threshold of 0.93. However, we can say that after some level, the contribution of latter eigenfaces is just at minimal level.

You can see the first three eigenfaces with the highest eigenvalues below, again which carry the characteristics of a face accurately enough:



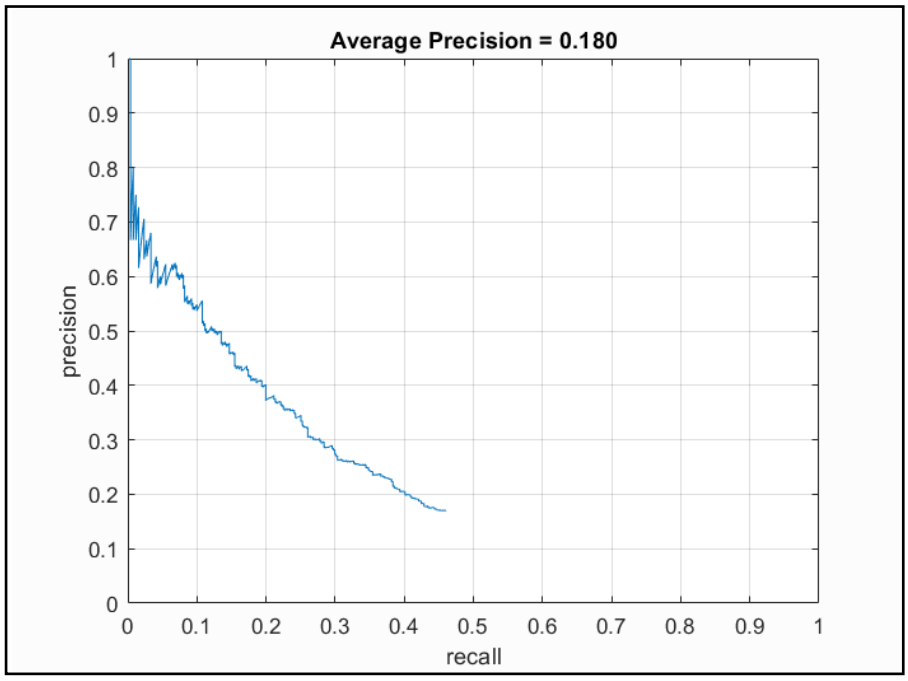
Then this eigenfaces are used for calculating the eigenface coefficients for both positive and negative training data. Then, those coefficients were fed into a support vector machine to learn a classifier between faces and non-faces. Before this, a kernel trick on the input data was realized. Each data point was expanded to 5 different dimensions via Chi-Square

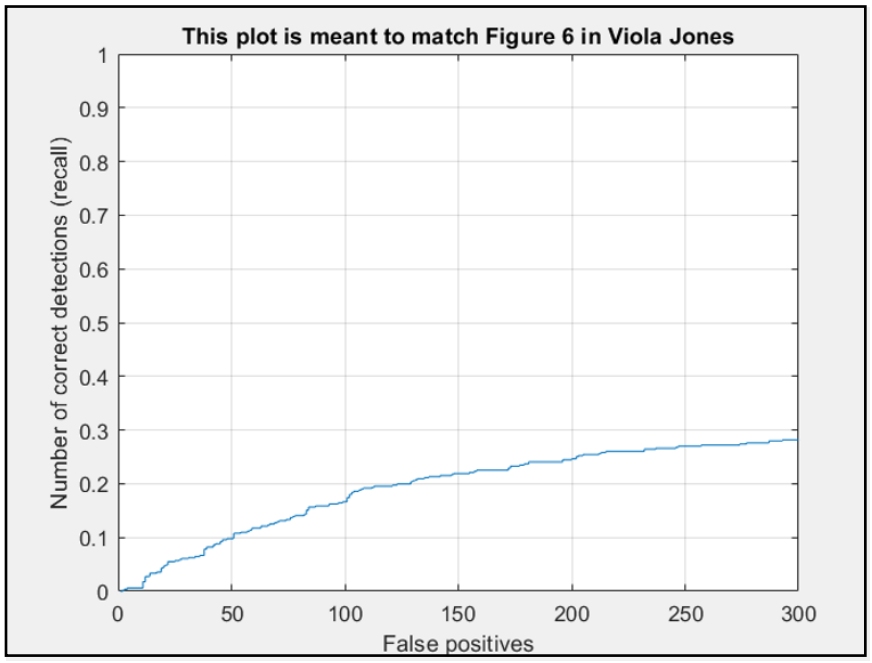
expansion, using VLFeat. At this point, it was noticed that SVM output can vary even though the input data is exactly the same. Most of the time, the accuracy was around 0.95, with an approximately equal distribution of false positives and false negatives. Then, the confidences were calculated for each training data along with their labels and a plot was created to evaluate the performance of the SVM. Our training plot shows that there is nothing wrong with the evaluation on training data, which is a must for any problem before moving on to testing:



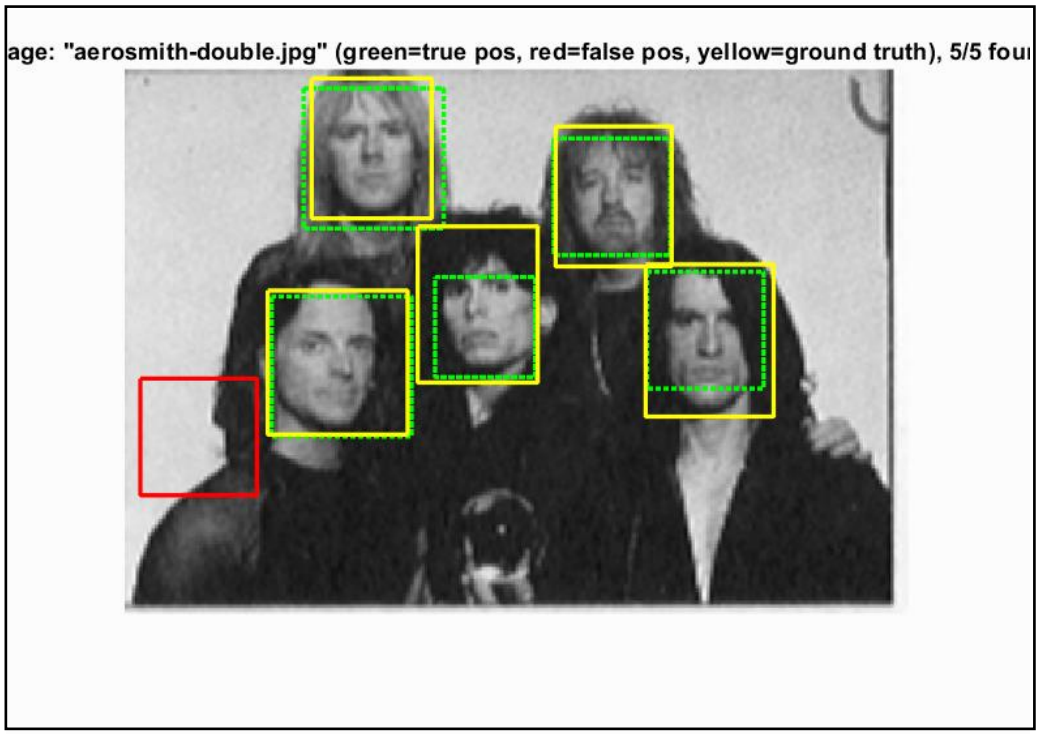
On the test data, I noticed that some of non-face images were detected as faces (obviously the SVM parameters w and b decide this), regardless of the input size, the step size, or the threshold. Furthermore, there were some non-detected faces in some images even though there were a lot of false positives. There might be several reasons behind this issue. First of all, we work with 36x36 cropped and straight faces only. If we look at the test data, we can say that the images are highly compressed and quantized, most of the faces have angles, the color of the skin varies across the test images, there are drawn, animated, or fake faces in most of the images. In fact, some of those faces are quite hard to detect with human eyes. The second reason could be that our training data could not generalize this classification enough to work on other cases, where illumination, viewing angle, colors and intensities are different. We could also say that there are pixel-wise similarities between our trained faces and the false positives in the images. Changing the threshold sometimes block out detecting faces, while still allowing false positives.

The third reason could be that this pipeline is a better choice to use with Histogram of Oriented Gradients that our Principal Components Analysis. We could see very promising results with no extra effect, with no hard-negative mining or any tricks with HOG, just under one minute. Testing phase for eigenfaces is quite costly in terms of computation, since we should run a 5 nested for loops; one for images, one for scales, two for image dimensions, and the last one for eigenface coefficient calculations. Below you can see my resulting plots, using 75,000 negative data, using a step size of 4, a threshold of 0.7, and using [0.97, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1] as our scale matrix:





Here are some images with precise detections, using 75,000 negative data, using a step size of 4, a threshold of 0.7, and using [0.97, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1] as our scale matrix:



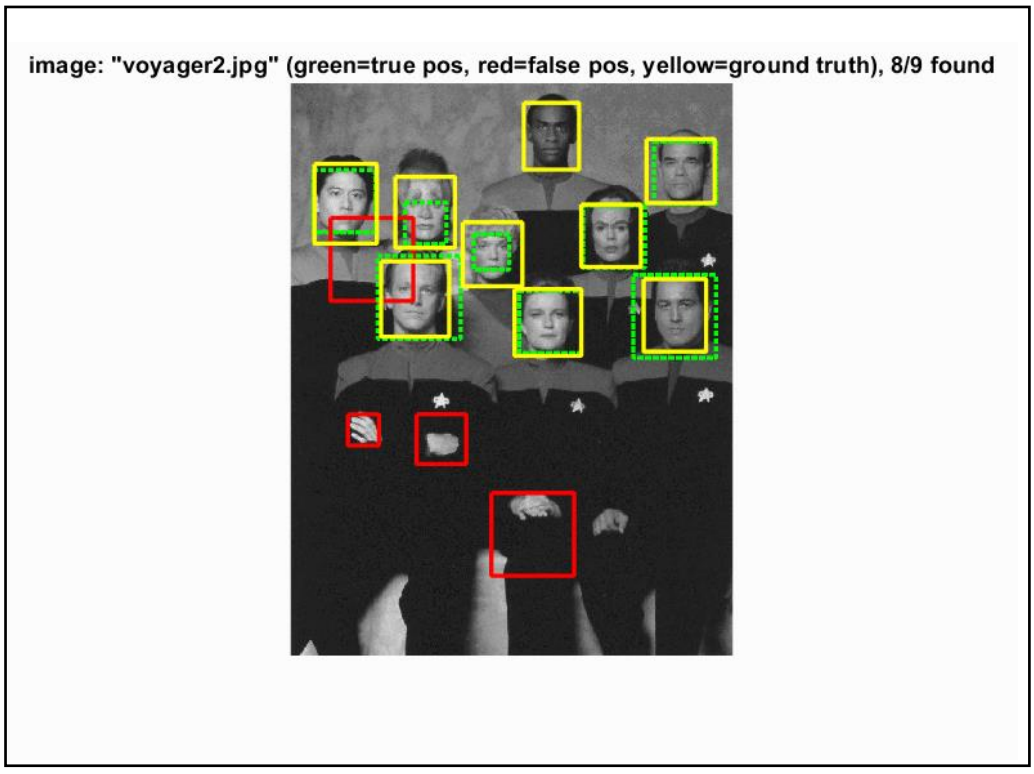


image: "trekcolr.jpg" (green=true pos, red=false pos, yellow=ground truth), 3/3 found

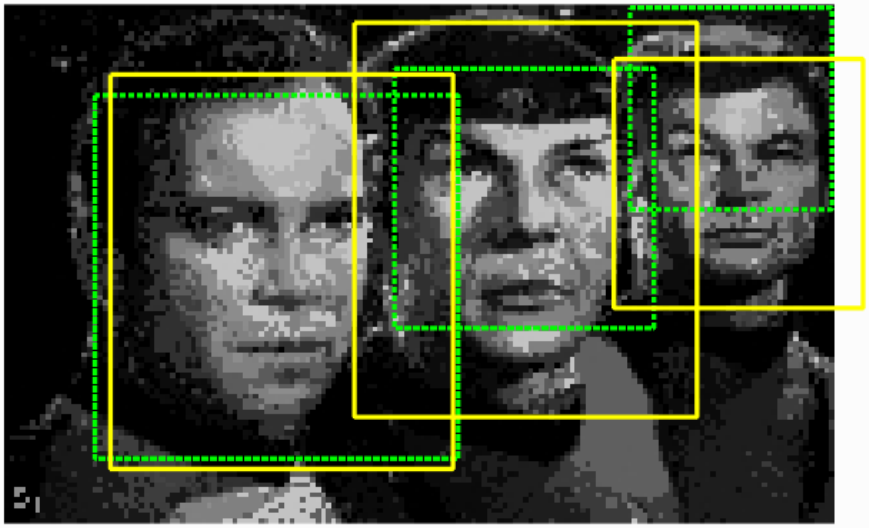
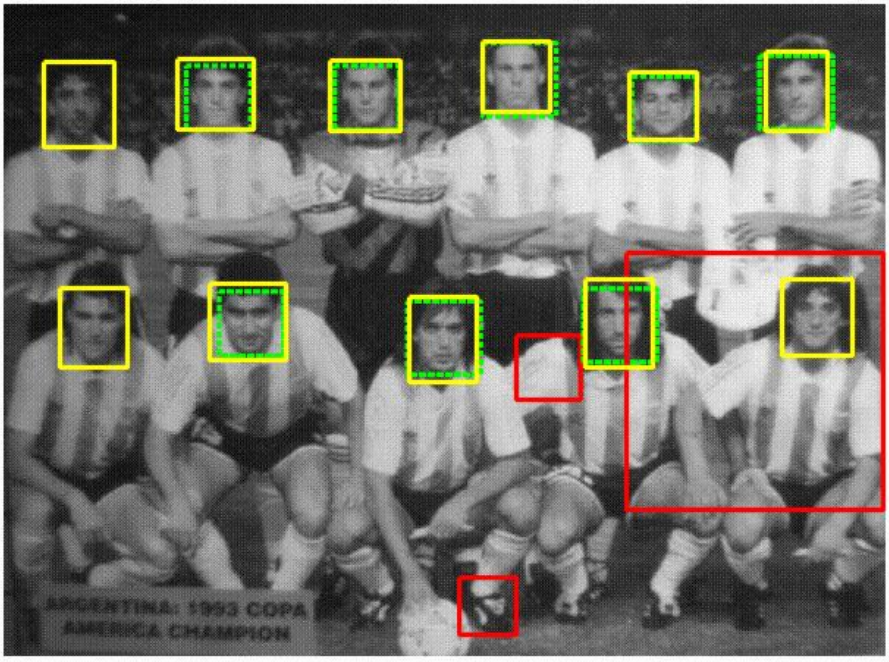
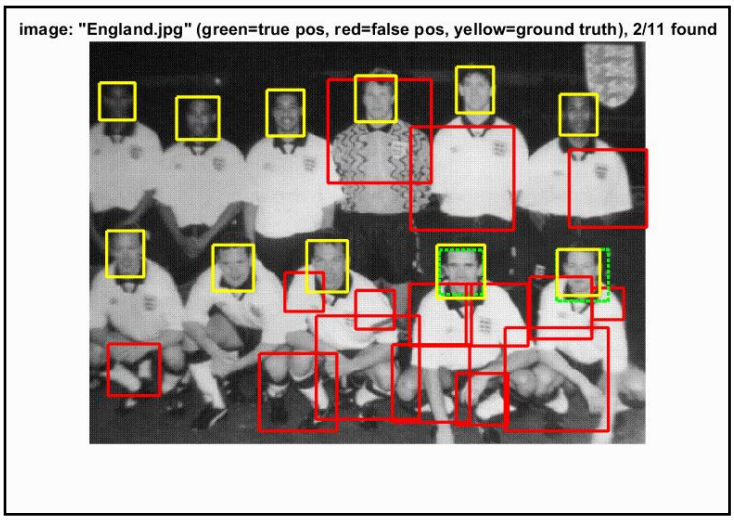


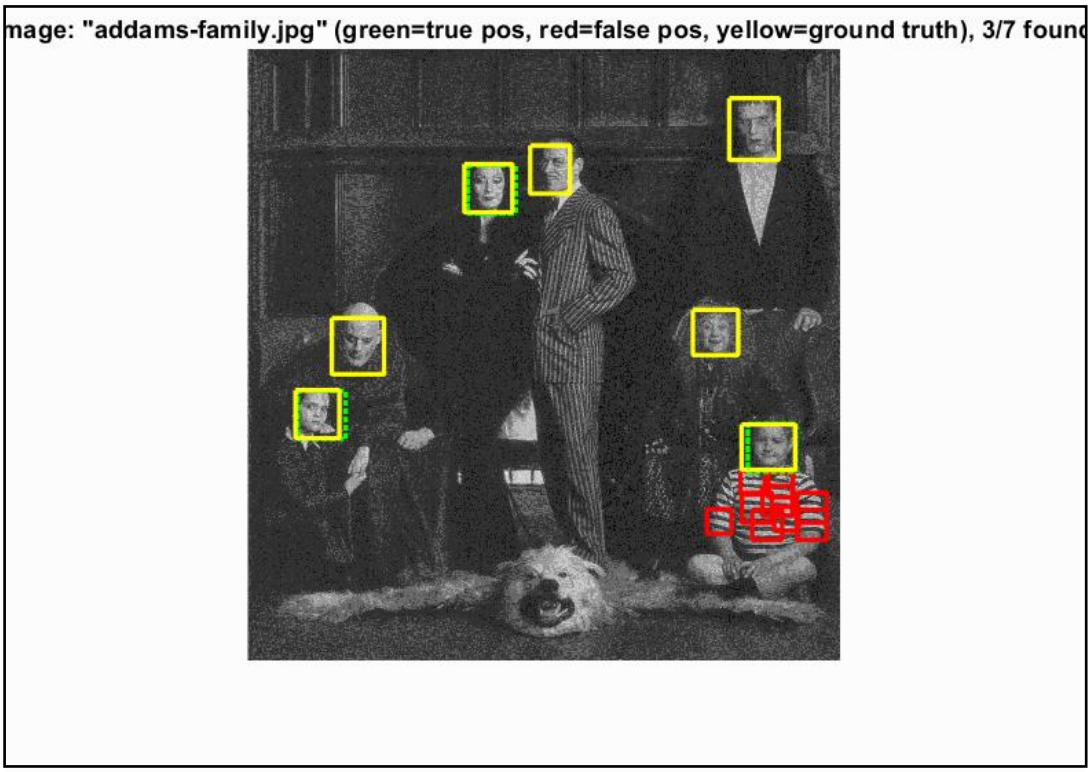
image: "Argentina.jpg" (green=true pos, red=false pos, yellow=ground truth), 8/11 found



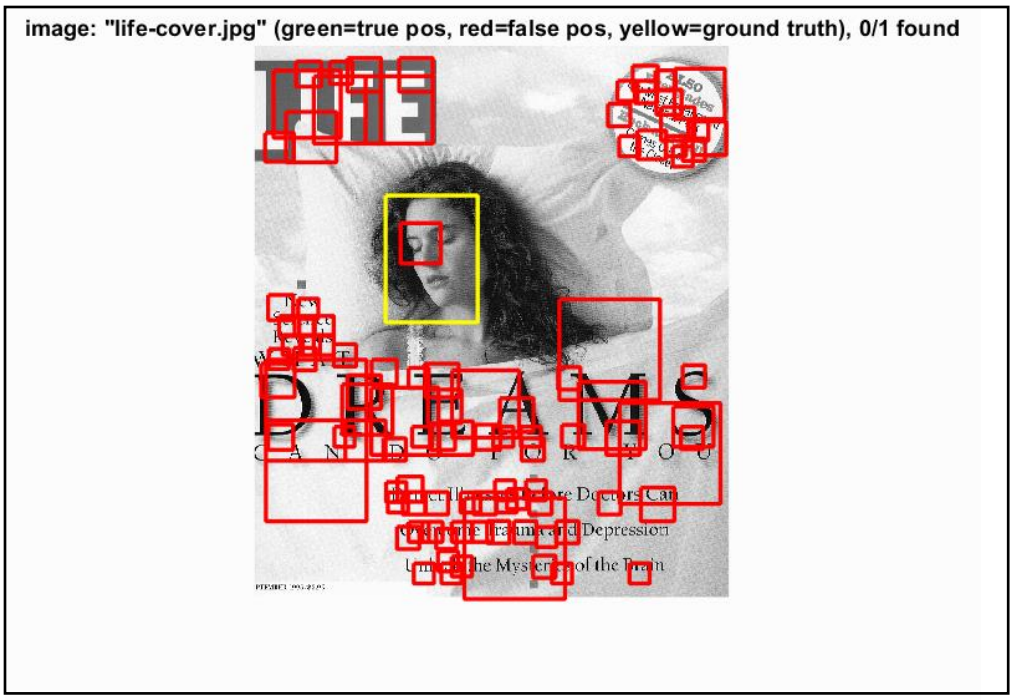


This image is deliberately added as a representative to make a comparison with the extra credit case (additional positive training data).

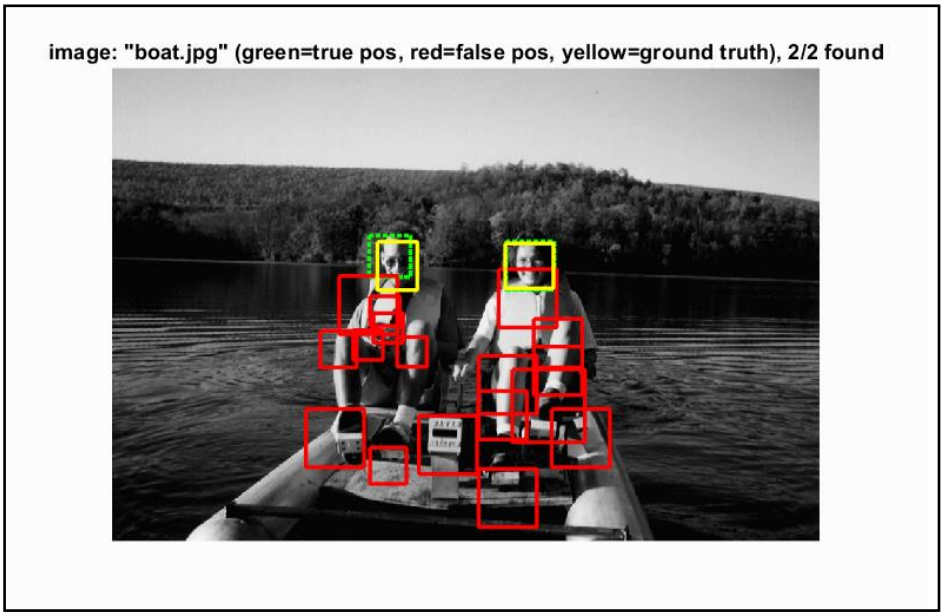
For these images above (except England.jpg), there is no delusional patterns to fool our classifier, hence the detections were almost complete and correct. Nearly half of the output is like this. However, we can see some patterns in false detections. To inspect this issue, here are some poor outputs, on which I will discuss any reasons:



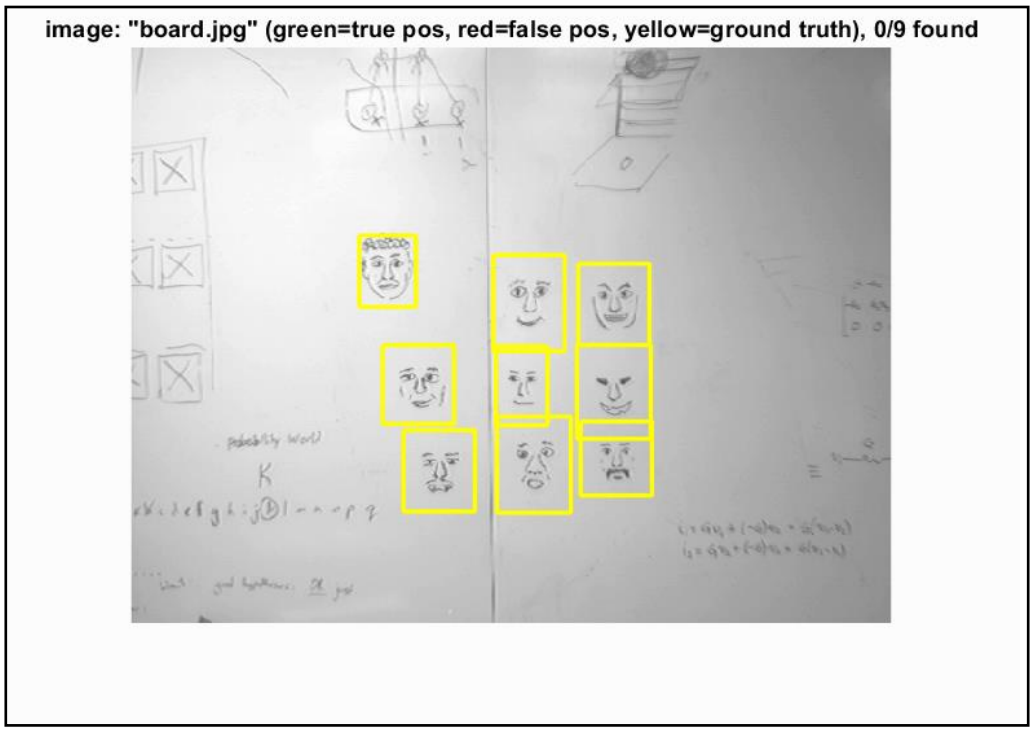
On the above image, we can see that the false positive detections lie on the boy's t-shirt. The reason it detects many false positives could be the striped texture of t-shirt, making the detector to predict those windows as frames. If we increase the threshold, face detections deteriorate, while the t-shirt is still getting detected.



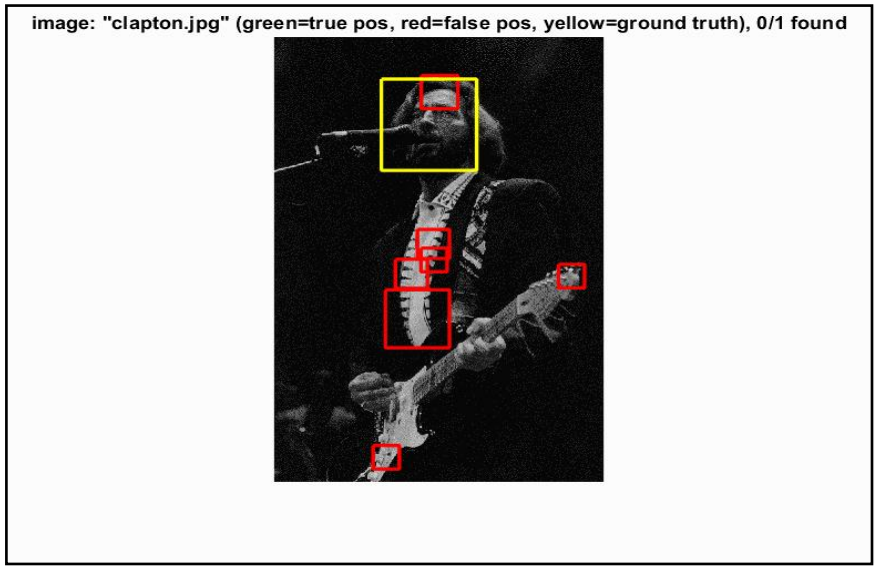
On this image above, an extreme number of false positives is detected. On a pixel-wise detection scheme, our classifier basically confuses white areas with darker surroundings or parts in it with actual faces. The face is viewed from an angle, so it was hard to detect with our pixel-wise detector.



Again, on the image above, we can see that whiter areas with shady surroundings, such as the t-shirt parts, knees and legs, the sign, was detected as faces. The detector was still able to detect ground truth faces properly.



This picture above is one of the hardest test cases for our classifier. Since our features are linear coefficients of a pixel-wise detection scheme, it was nearly impossible to detect hand-drawn faces, whose resemblance are minimal to our training data.

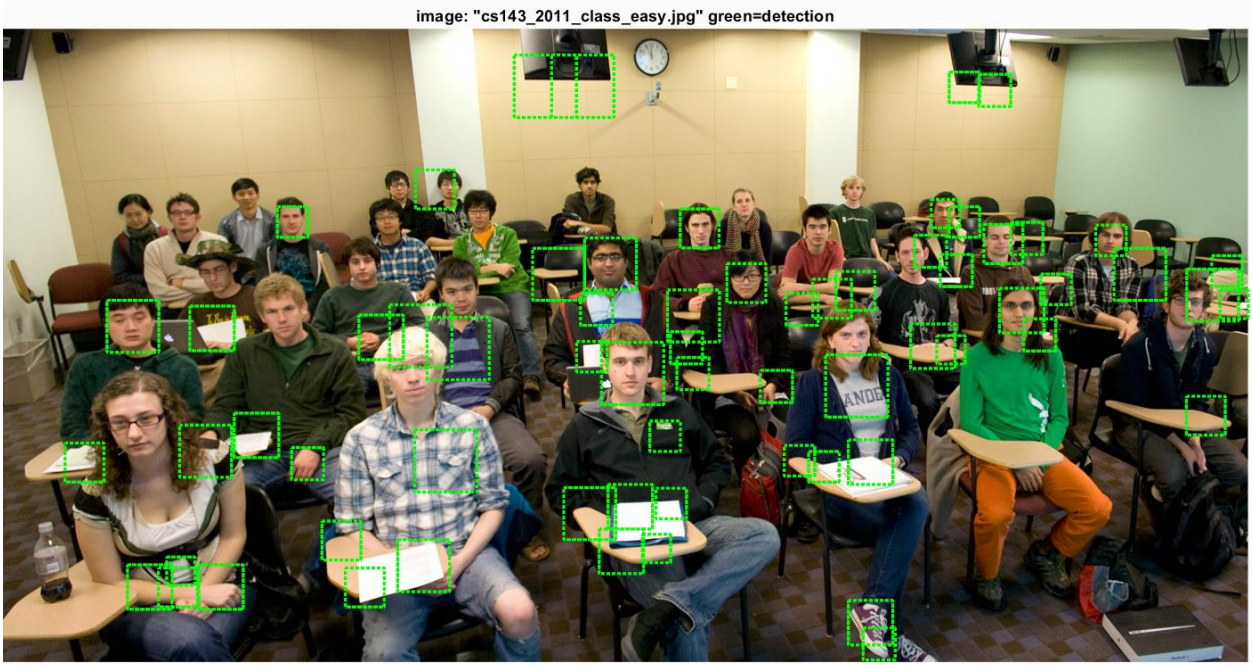


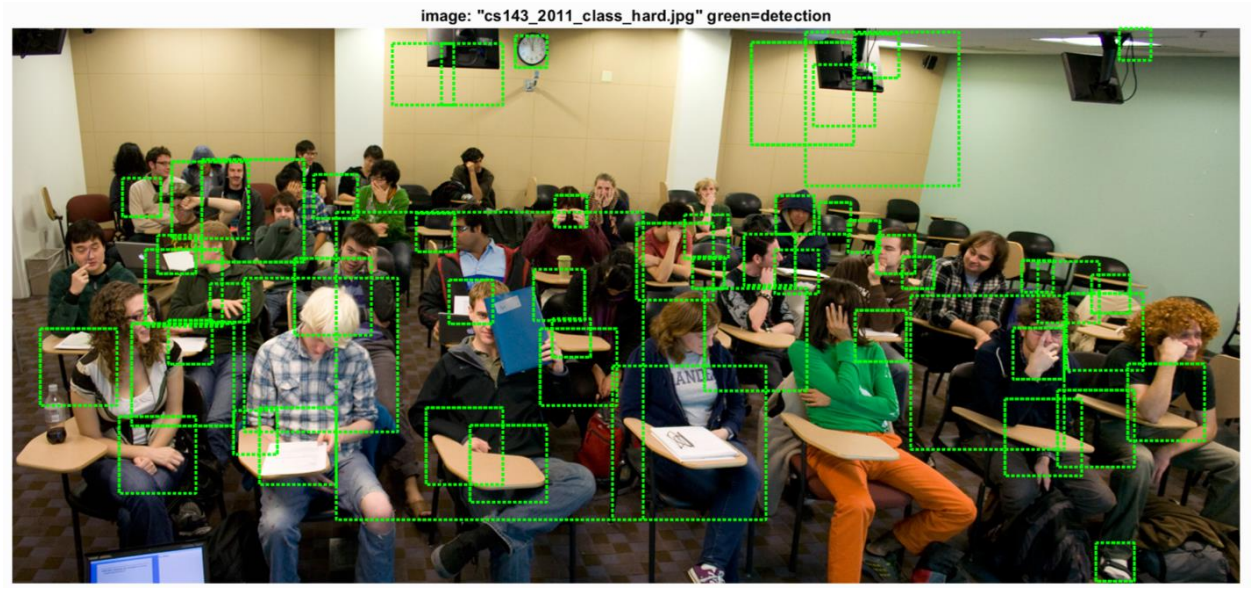
On the image above, it can be seen again that many areas with its white center and darker surroundings were detected, while Eric Clapton's face was not because the presence of the microphone as an obstacle, and of the viewing angle with intensity changes.

Some other output visualizations are attached alongside my code.

Performance on Extra Test Scenes

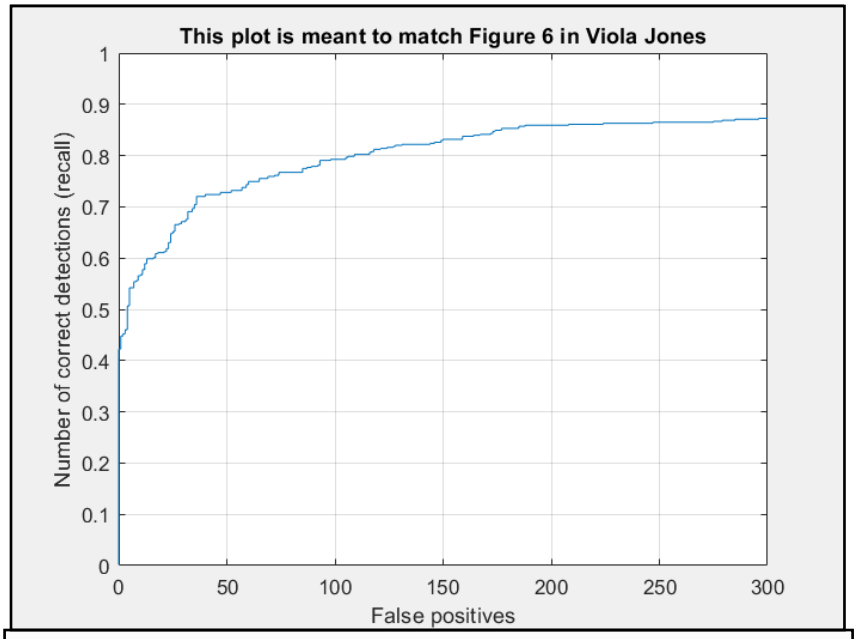
Extra classroom scenes were tested using a different visualization script and using the corresponding path. Same parameters were used except the scale matrix. Since there are lots of smaller faces in the image, I also added scale parameters which are greater than one, to further zoom in the image, and removed some smaller scales since we are not looking for enormous faces. Since our features are pixel-wise, the detector had almost zero accuracy in hard cases, where each person tries to hide from detection. In contrast, there are detections in easy images. I used scale = [1.18 1.1 1.02 0.97, 0.8, 0.7 0.6 0.5] in my detections. Here are some results:

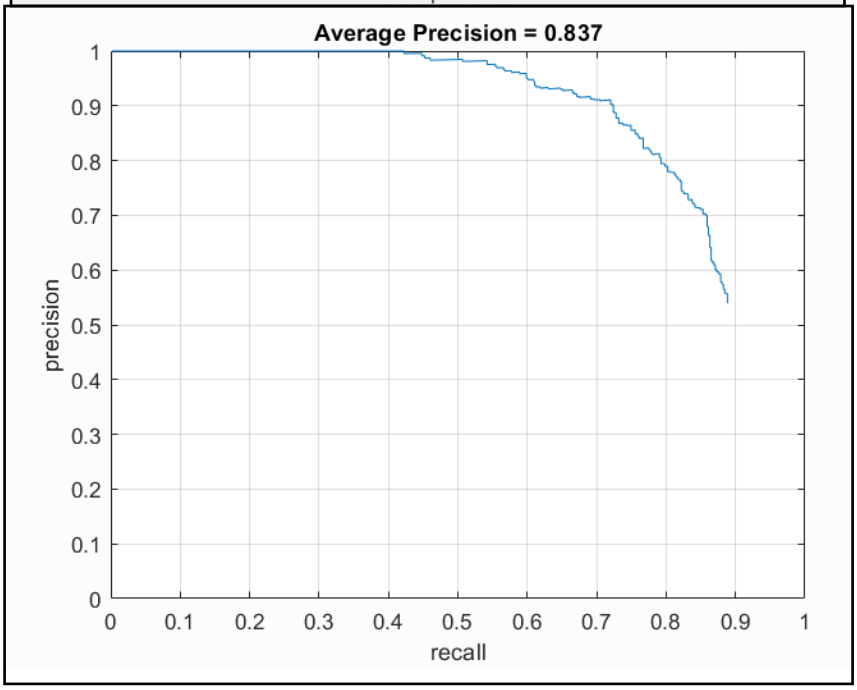




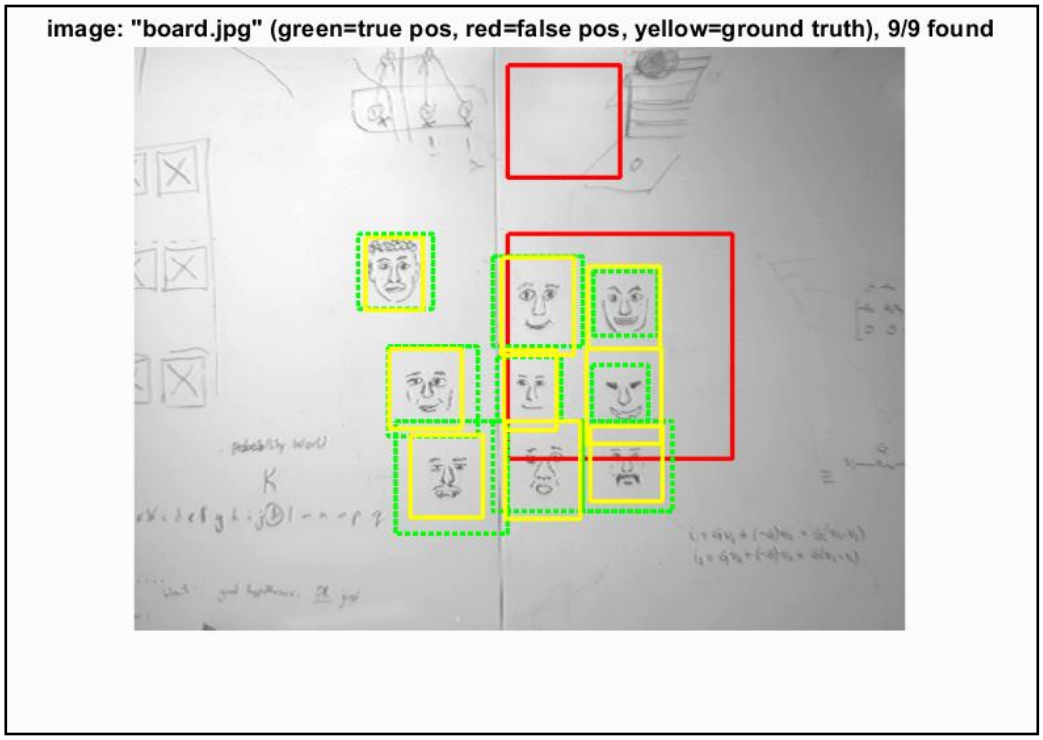
Discussion and Comparison with HOG

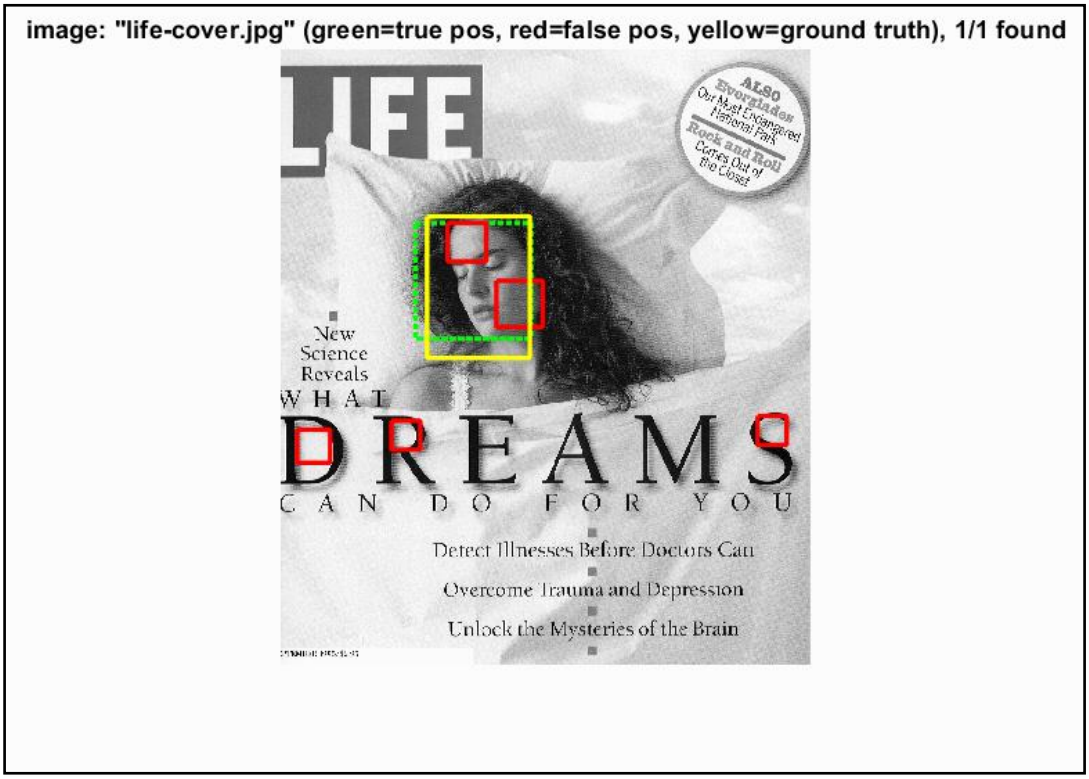
The starter code given to us can detect with around .895 precision in minutes. One thing to notice is that it rapidly searches between test images, while our sliding window moves much slower because of the pixel size step. Calculation of eigenface coefficients for each window is again costly in terms of computation. Training results are slightly more accurate for HOG. Since the features of HOG method contain more robust information, it is more accurate on the test set. We can easily observe that it performs quite well on images. Here are the statistics belonging to HOG method:

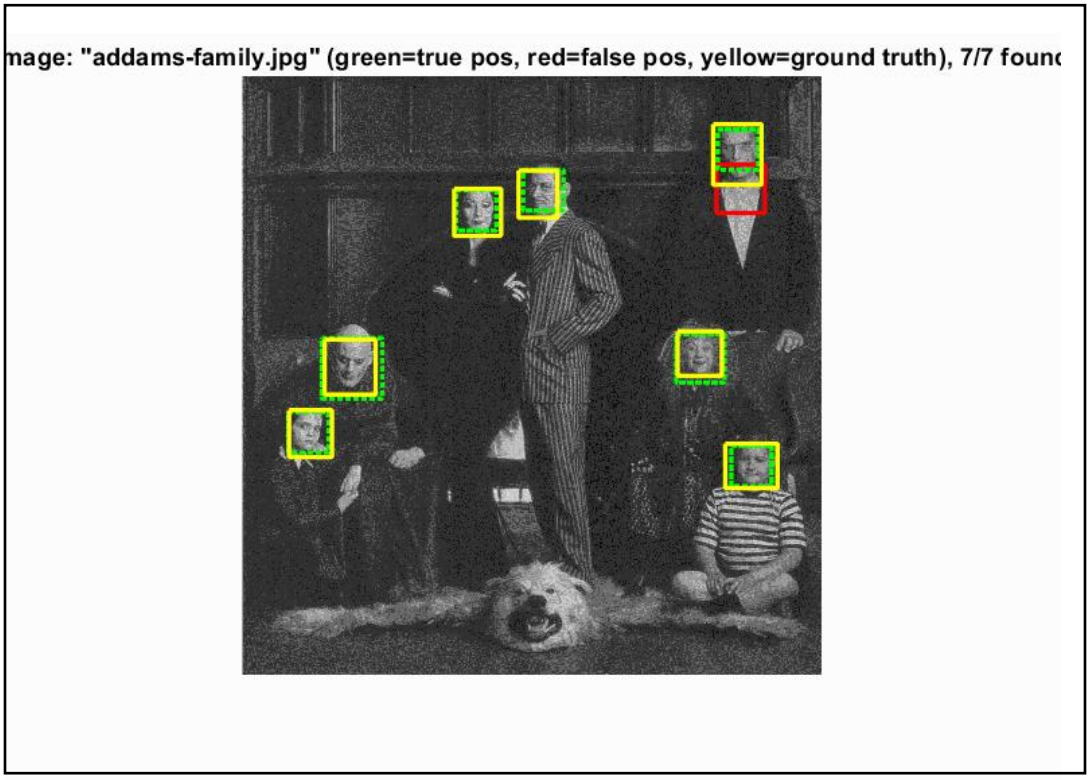




On the images below, you can see HOG outperforms our outputs with high false positives due to its robustness:







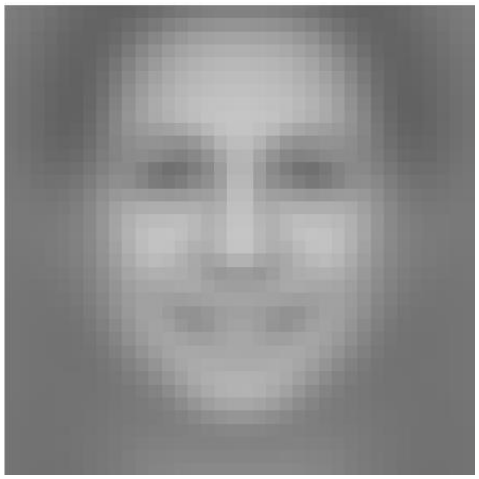
Let's discuss our detector further. Given the visualization of mean face, eigenfaces, and the result of SVM training, we can safely say that our pipeline works flawlessly until that point. The kernel trick also helps the accuracy increase a bit more. Kernel trick can be very useful when mining hard negatives, since the task becomes highly non-linear at that point.

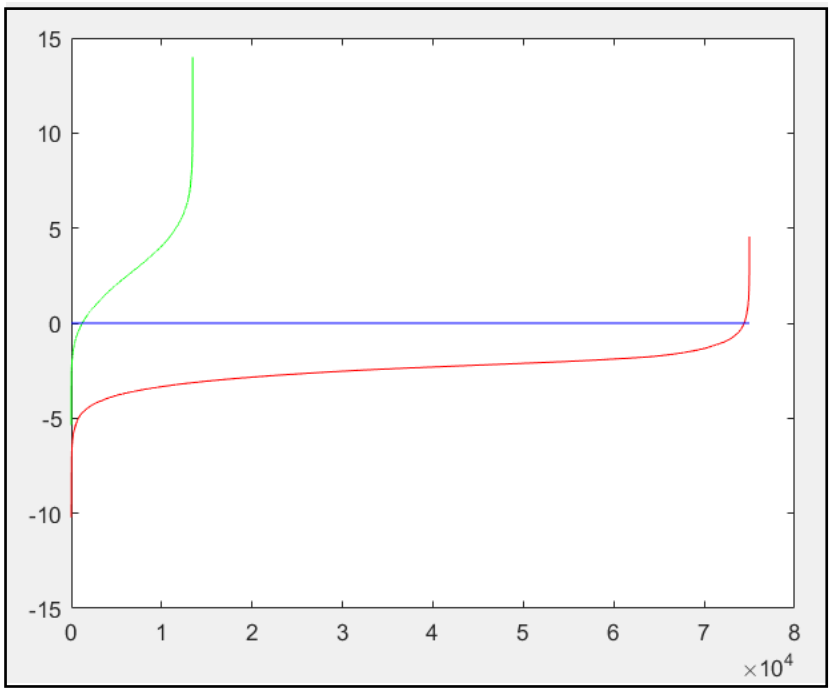
Our detector can detect almost all the faces with lower threshold values, however; we see that the number of false positives also increase. I believe that there is a natural limit on the number of faces we can detect with a positive threshold; a negative threshold would be conceptually wrong. If inspected, it can be noticed that false positives are mostly lighter areas surrounded by some darker pixels, which our detector thought as a face. This could be because our faces are too small, and in some sense, they are basically lighter areas at the center with darker areas around it. The detector is unable to classify between dark points inside the face (eyes, lips) and confuses faces with simple light patches of images. This difference could be because the negative training data were in high resolution, while test data suffered from compression and old imaging techniques.

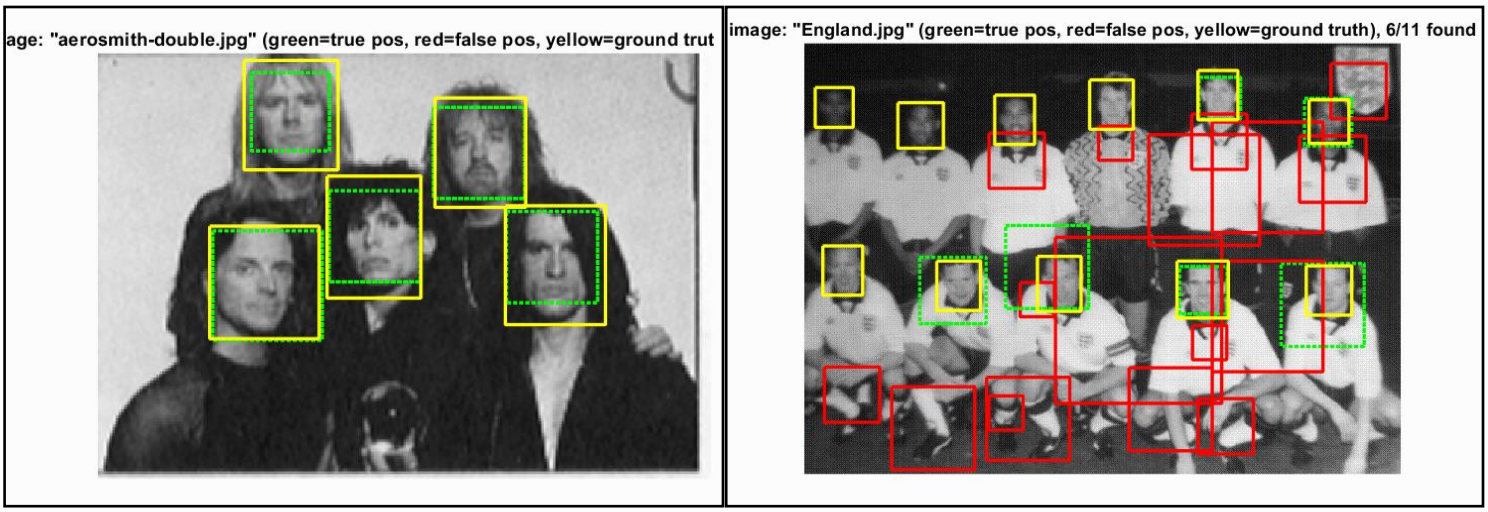
Extra Credit: Utilizing New Positive Training Data

For this task I did a simple but efficient trick. I wrote a script (MirrorPositiveImages.m) to mirror the contents of each of our 6,713 faces vertically (same as rotating 180 degrees around y-axis). This increased the total number of positive training data to 13,426. When we utilize this new data to train our support vector machine, we can see that both the accuracy result on training data and the performance on the test data has increased, although again leaving us with a "false positive" problem, just as in the normal case. You can see that false positive rate is quite low compared to the normal case, on training data. Mean face below is almost the same as the normal mean face, which is not surprising since we are only working on symmetric images, and our mean face was already an almost-symmetric one.

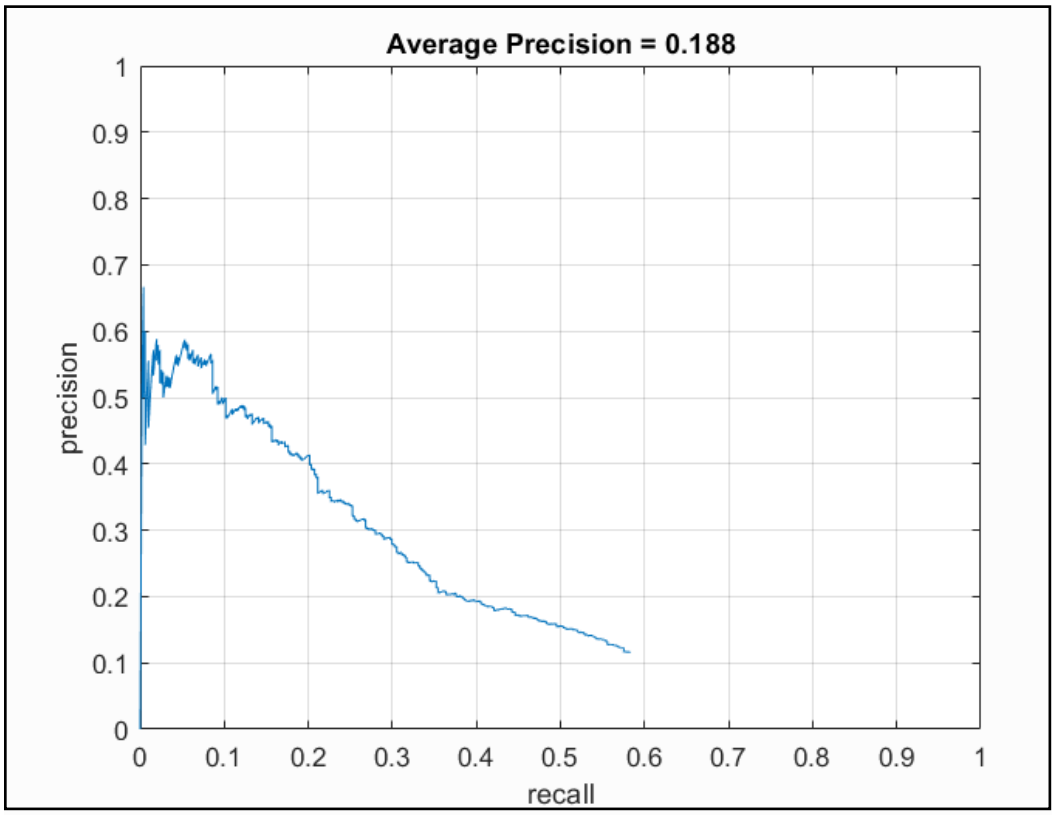
```
accuracy:    0.980
true  positive rate: 0.138
false positive rate: 0.007
true  negative rate: 0.842
false negative rate: 0.014
```

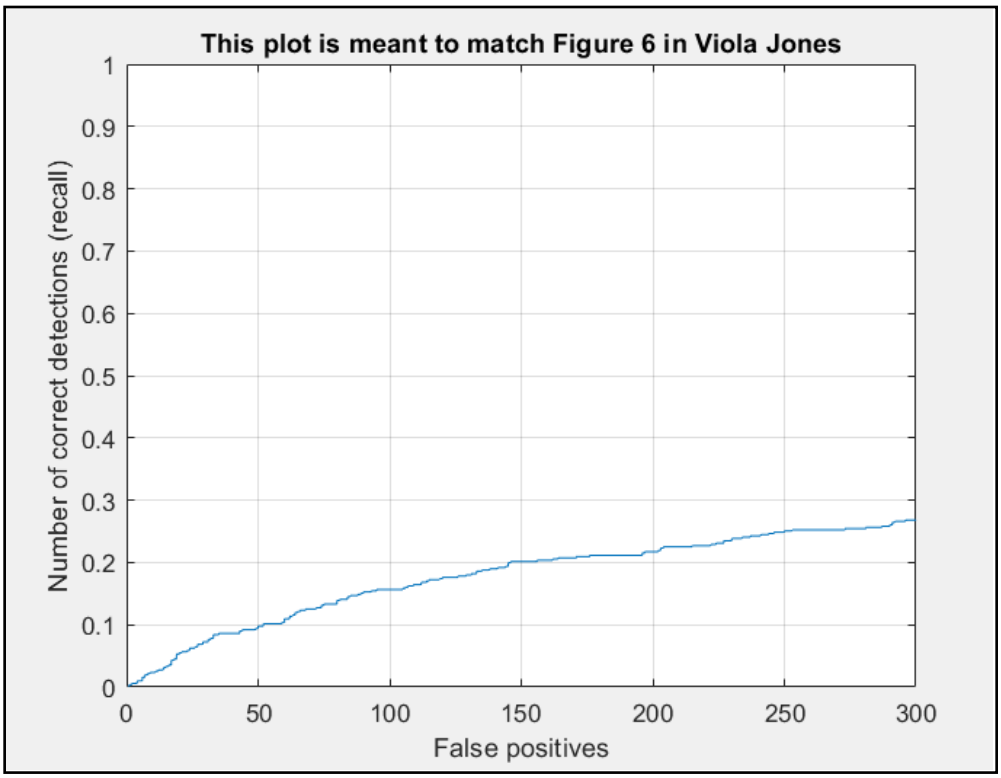






Among the pictures I have shown above, these two are some of the examples with a superior output (others are equivalent). If we look over the detections, we see that more faces are detected compared to our original pipeline, along with a more of false positives, which ultimately decreases our average precision. It is also noted that extra faces have contributed to have slightly better training with SVM. Recall-precision graph and detection graph are shown below:




This plot is meant to match Figure 6 in Viola Jones
Number of correct detections (recall)
False positives
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
50
100
150
200
250
300