COMP 408/508 - ASSIGNMENT 3

REPORT

Özgür Taylan Özcan

Algorithms

- **get_training_features:** In step 1, I read the positive images, convert them into type single, then compute hog features for each. In step 2, I read negative images, convert them into grayscale and type single, get random samples from each, then compute hog features for each sample.

- **svm_training:** First I merge positive and negative features to get a single feature vector. Then I take the transpose of this vector to be able to use in the vl_svmtrain function. I form the positive label vector consisting of ones, and formed the negative label vector consisting of negative ones. At the end, I apply vl_svmtrain function on the feature and label vectors, to train the classifier and get weight & bias values. I use the lambda value as 0.00001, because it resulted in good accuracy.

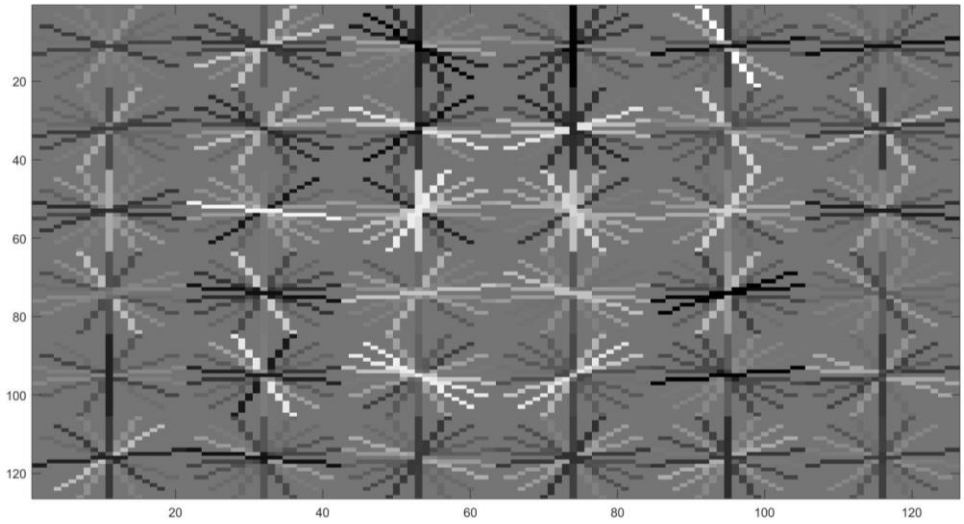
- **classifier_performance:** I merge positive and negative feature vectors. Then I apply the classifier on this merged feature vector to compute the confidence vector containing confidence values for each feature. After this, I iterate over confidence values for positive and negative features to catch the true/false positives/negatives. At the end, I calculate the values for accuracy, recall, tn_rate and precision, as stated in the description.

- **run_detector:** I first convert the rgb images into grayscale and then convert to type single. I run a loop to process images in different scales. For each scale, I compute hog features applying vl_hog function on the scaled image using the given hog_cell_size. Then I loop over the hog cells to simulate the sliding window. For each window, I get the relevant hog matrix and turn it into a feature vector. Then I calculate score value for the window, using the classifier (weight and bias). If the score is greater than the previously defined threshold value, I calculate the boundaries of the box (window) and add it to boxes vector. I also add the score value into confidences vector.
  In my implementation, the scale starts from 1.25 and is multiplied by 0.9 at each step, until it is below 0.25. This way, too small or large faces can also be detected.
  I set the threshold value to 0.8, because I found out that it produces the best results after trying some different values.

Training dataset

I used the positive and negative images as given in the training dataset. In total, I used 6713 positive images and 10000 negative images (after sampling) for training the classifier.

Visualization of the classifier



Performance metrics for training

My metrics are as follows:

- Accuracy: 0.9993
- Precision: 0.9997
- Recall: 0.9987
- Tn rate: 0.9998

The values for these metrics are computed by using true positive, true negative, false positive and false negative values. All of these 4 metrics show different aspects of the training algorithm. Higher rate means better trained classifier. My metric values are pretty good. They are all close to 1.

Additional Information

I directly applied multiple scale detection and did not apply single scale detection. So I cannot state the differences between them.

I first started by setting the initial scale value to 1. However, I realized that some small faces in the images cannot be detected. Therefore, I changed the initial scale to 1.25. It decreased the performance a little bit, since it takes more time to process larger images. However, the accuracy of the classifier is improved.

Figures showing precision-recall curves

As shown in Figure 1, I get 0.852 average precision, which is good. Furthermore, Figure 2 looks similar to Figure 6 in Viola Jones, which shows that I get a good result.

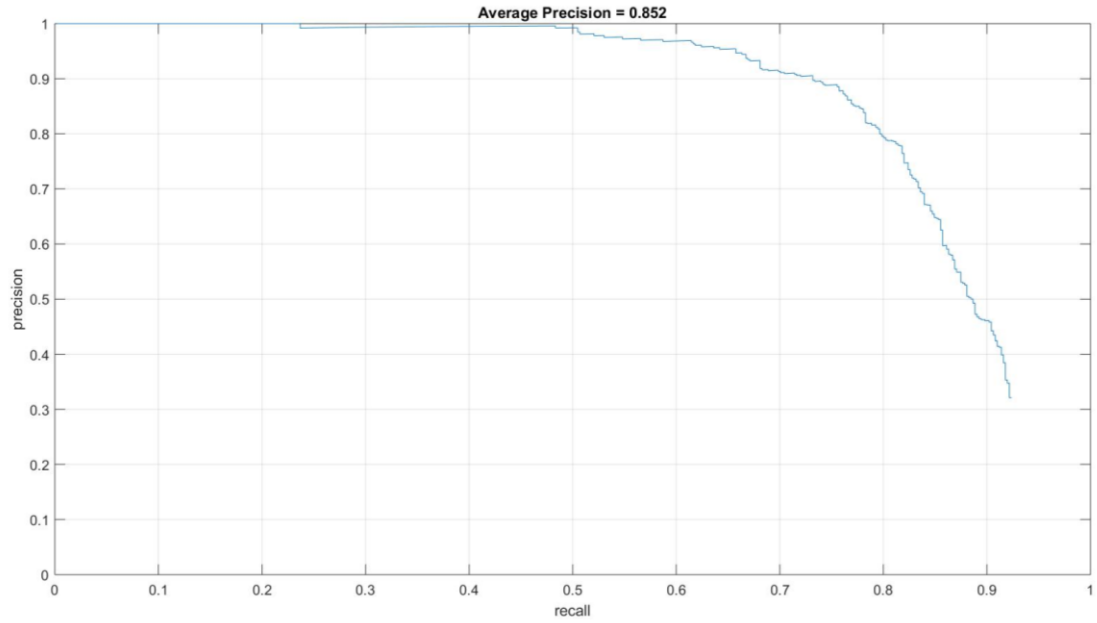


Figure 1

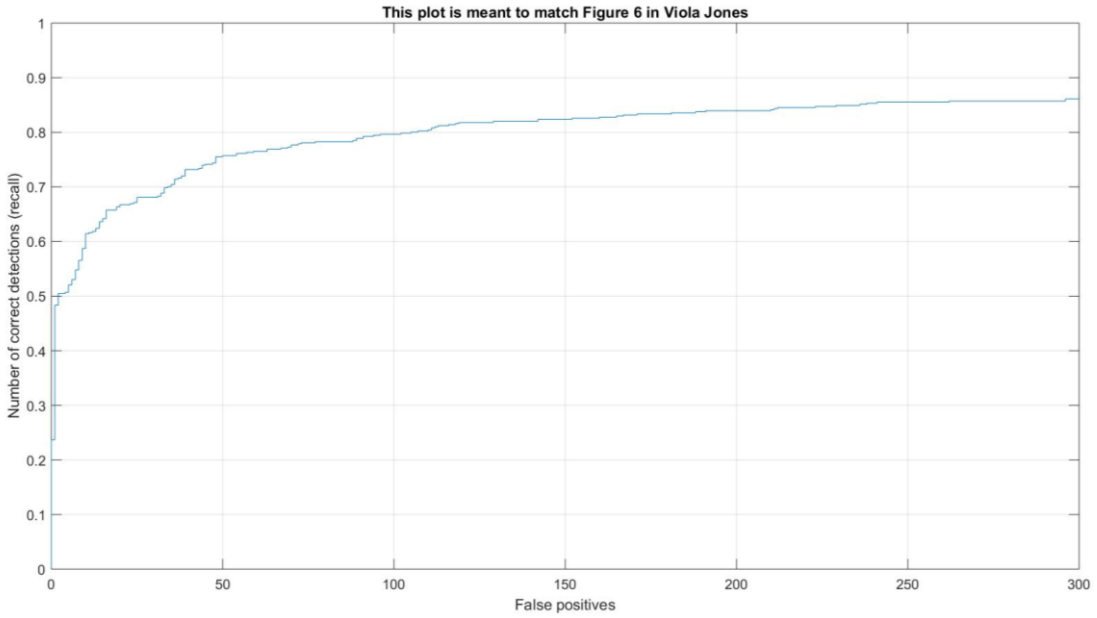
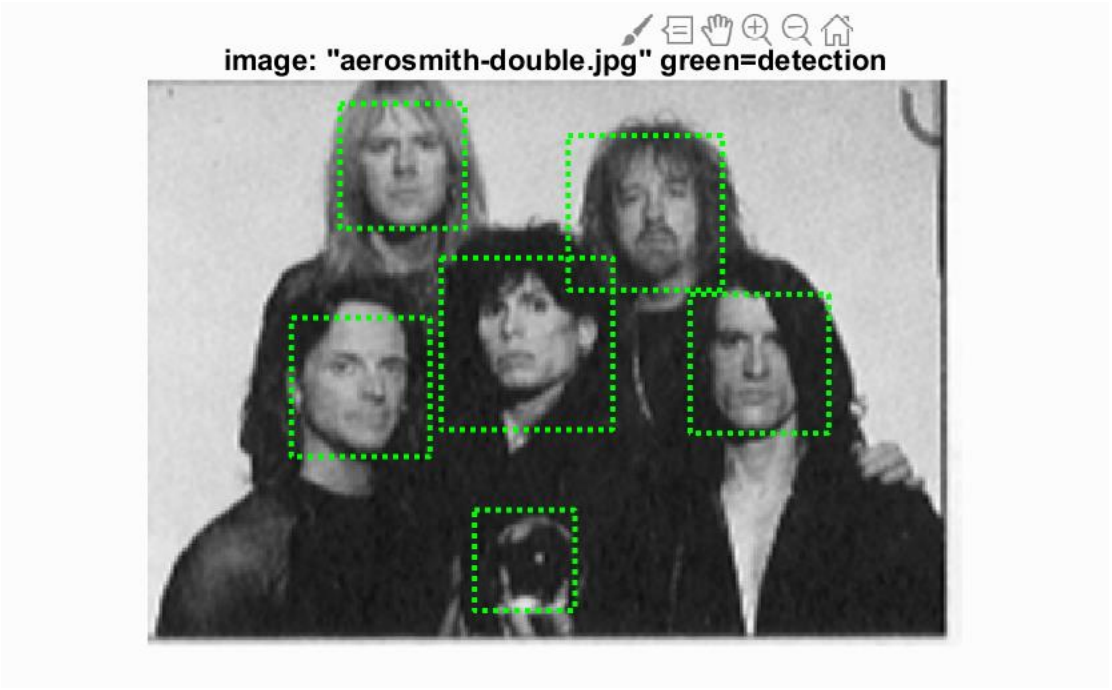
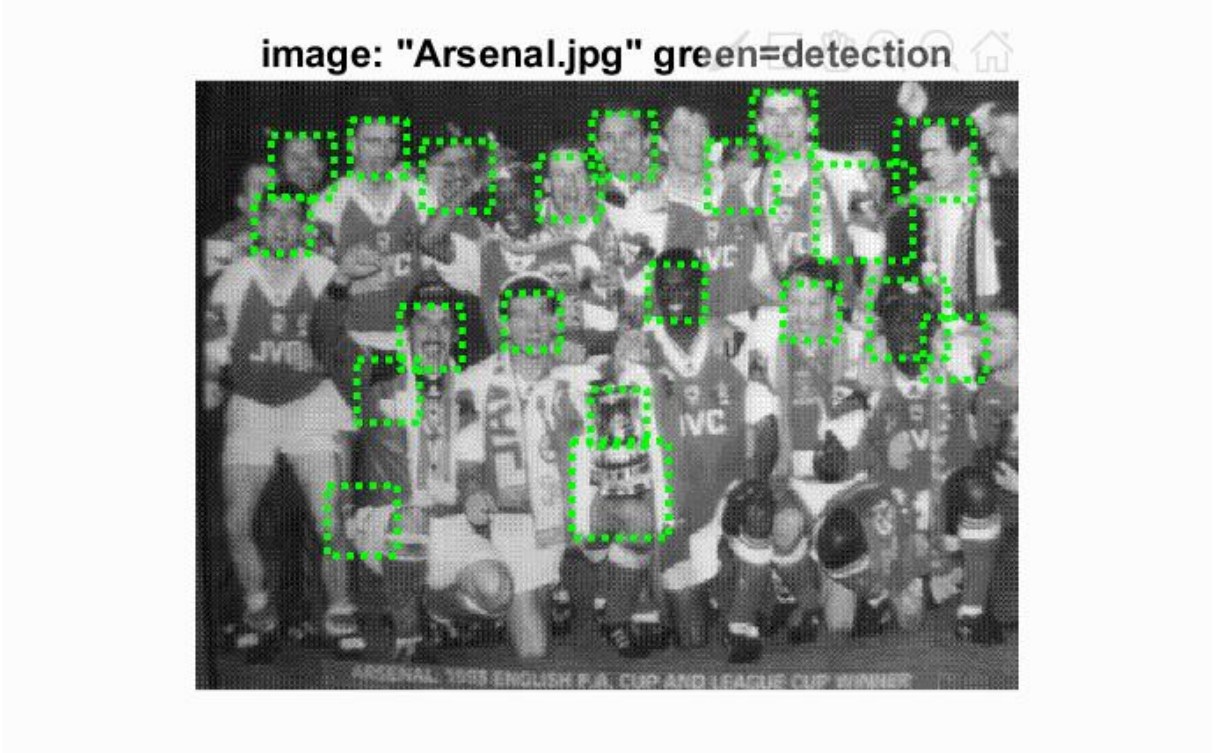


Figure 2

Some output images produced by my algorithm





Some output images from extra test data