RealEyes homework solution

(for machine learning intern position)

1 Overview

During solving the homework I went through the following steps:

- Getting familiar the basics of with OpenCV library and Viola-Jones method
- Implementing face and face-part detection
- Checking the results on a few images with different content and style to have some intuitions how the detections behave
- Creating a dataset of images (labelled with face or non-face) in order to test the face-detection and the proposed algorithm (and splitting it to test and train datasets)
- Testing the accuracy, precision and recall of the face detection with recommended parameters on the test dataset
- Detection face parts on train and test dataset and creating features from the results based on my intuition developed from the early testing
- Implementing a classifier for face detection using the features created from face-part detection. The aim is to get better result than the "default" face detection in order to be able to test the default face detection correctness by this new method on a non-labelled dataset or to use this new method for face detection.
- Optimizing parameters of the new classifier by training on train dataset and testing on test dataset
- Drawing consequences

2 Implementing face and face part detection, getting some intuitions how they behave

Having implemented the VJ face and face-part detections I tested several cascades and its parameters on a smaller dataset I created. The dataset contains images of face with background, without background, frontal faces and faces from a little angle. I also incorporated face-like, but non-human face images.

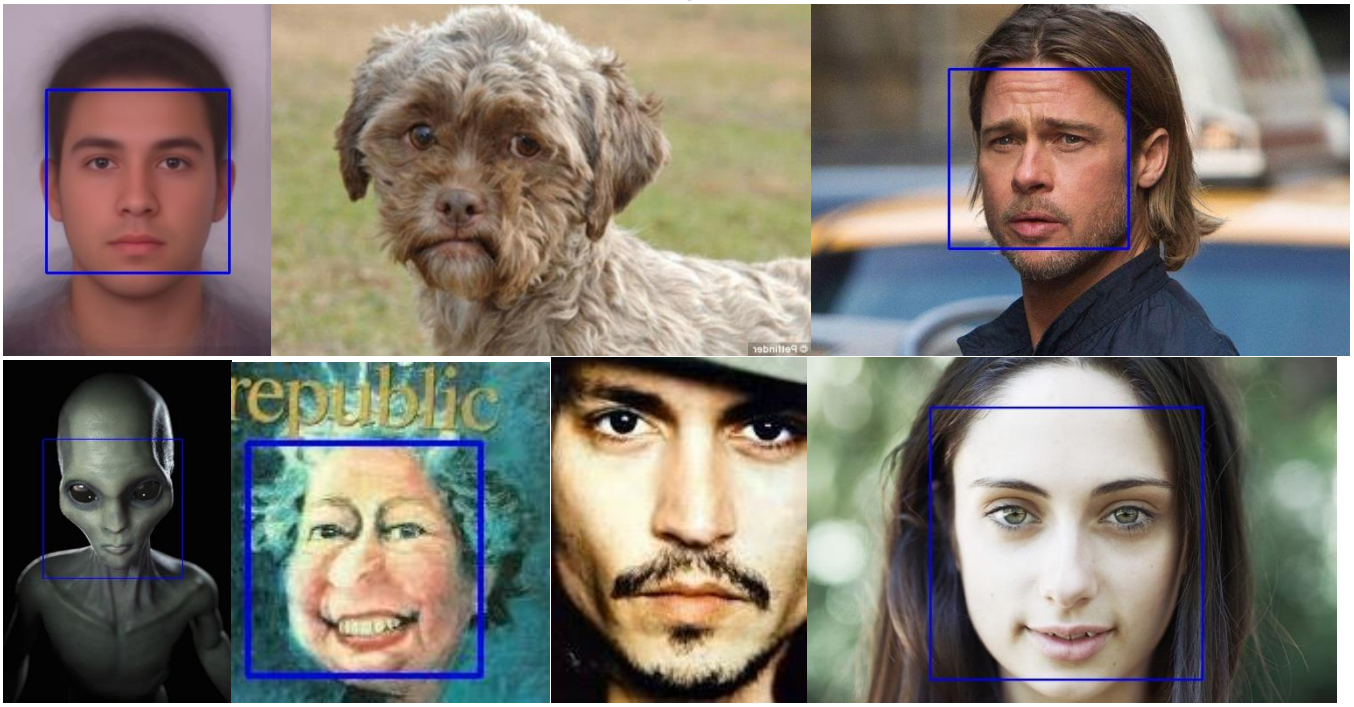
After the test I realized that face part detection may not be so accurate, that is to say, finding the good detection sensibility is not easy, since it either detects several mouths, eyes, etc. or only one mouth, two eyes, yet in this latter case it tends to skip detection on several images of faces. For my purpose, a bit higher sensibility might be better, so that the results can be applied effectively as features for a new classifier, still, it is a question to be analysed more thoroughly.

Tested cascades

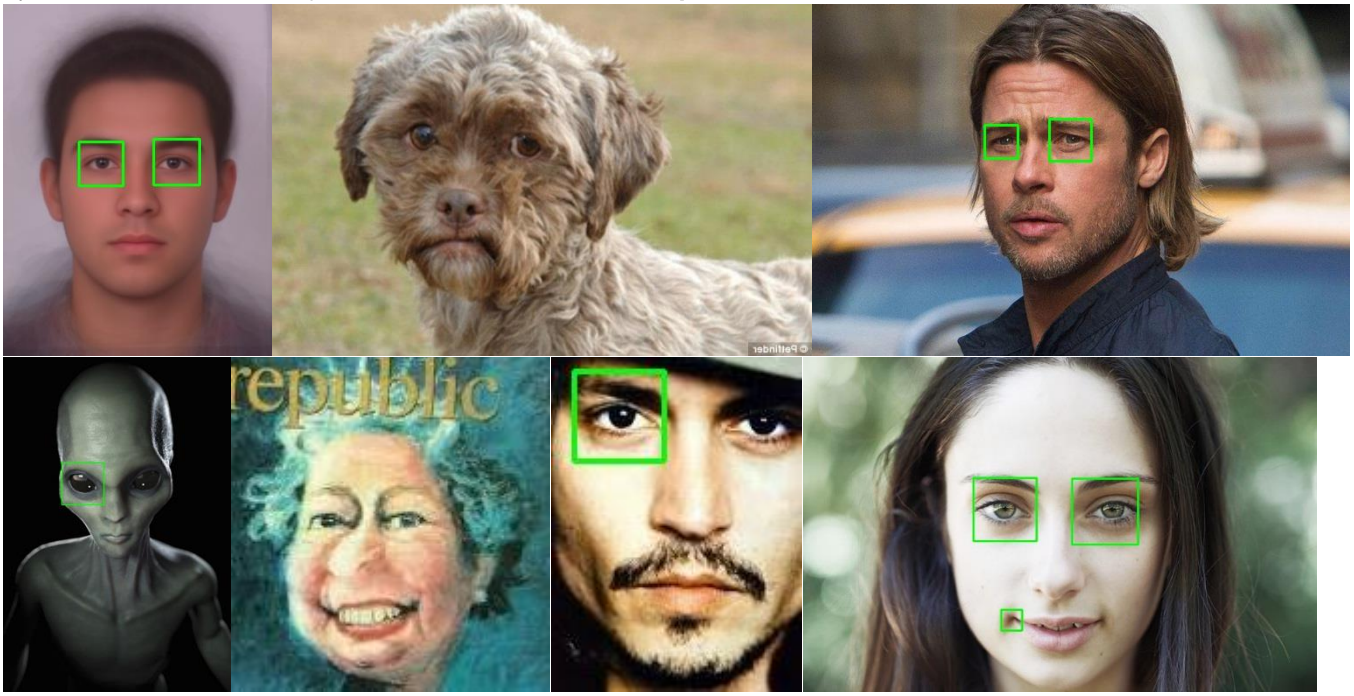
```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default_.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye_.xml')
l_eye_cascade = cv2.CascadeClassifier('haarcascade_lefteye_2splits.xml')
l_eye_cascade2 = cv2.CascadeClassifier('left_eye_modesto_train.xml')
oneeye_cascade = cv2.CascadeClassifier('haarcascade_one_eye.xml')
frontaleyes_cascade = cv2.CascadeClassifier('frontalEyes35x16.xml')
mouth_cascade = cv2.CascadeClassifier('Mouth.xml')
nose_cascade = cv2.CascadeClassifier('Nariz.xml')
```

Some images of the test

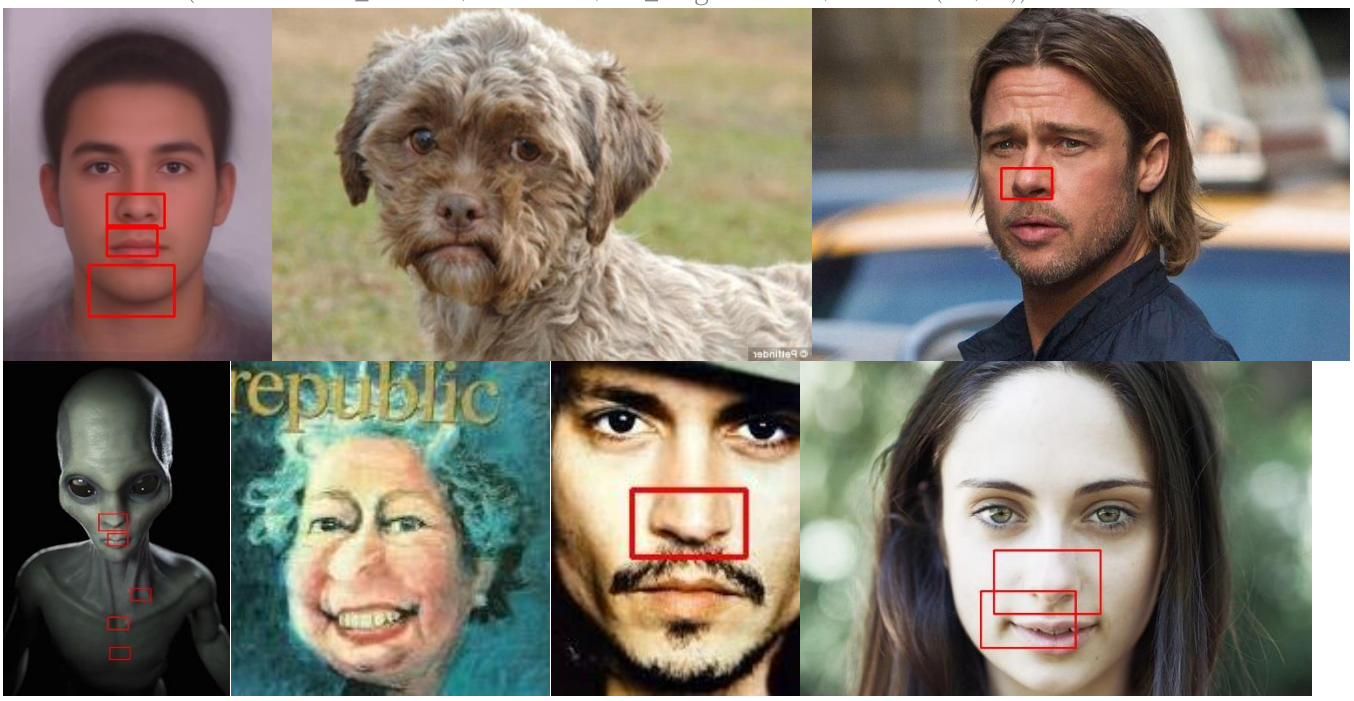
Face detection (cascade=face_cascade, scalef=1.3,min_neighbors=5,minsize=(30,30)):



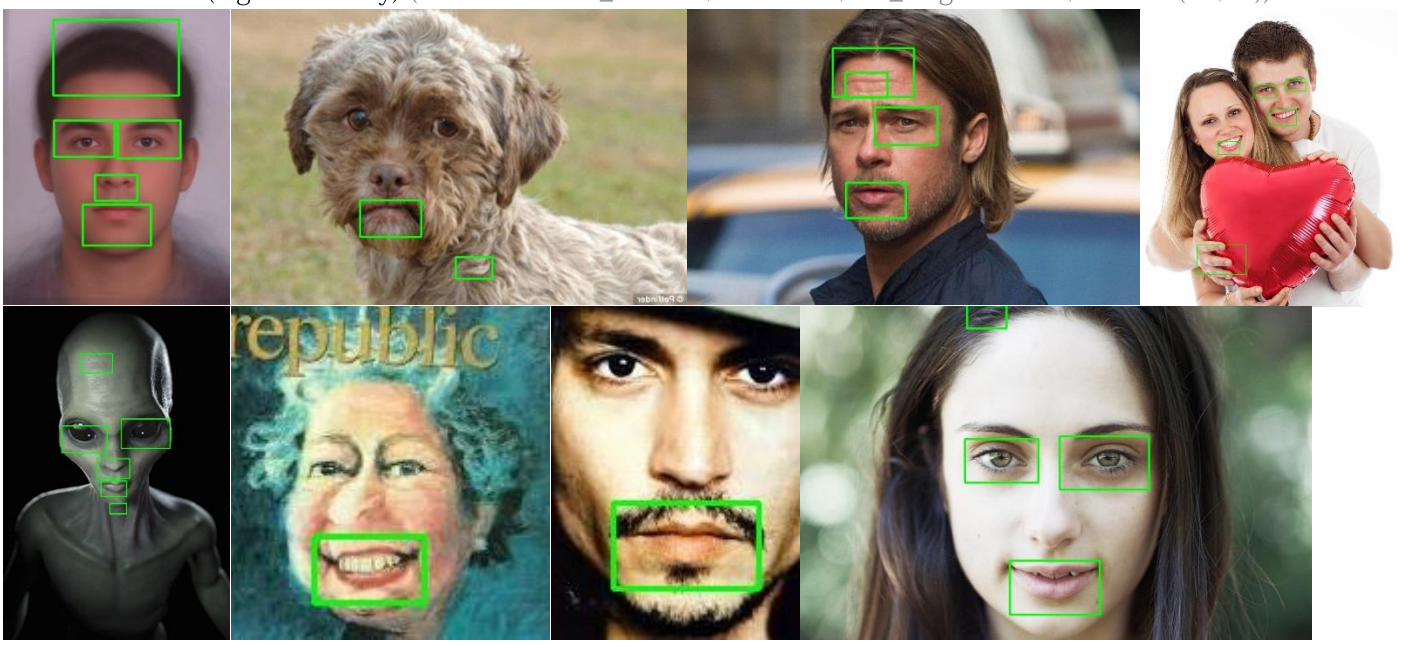
Eyes detection (cascade=eye_cascade, scalef=1.3,min_neighbors=5,minsize=(20,20)):



Nose detection (cascade=nose_cascade, scalef=1.1,min_neighbors=10,minsize=(20,20)):



Mouth detection (high sensibility) (cascade=mouth_cascade, scalef=1.2,min_neighbors=10,minsize=(20,20)):



Mouth detection (low sensibility) (cascade=mouth_cascade, scalef=1.2,min_neighbors=100,minsize=(20,20)):

3 Creating a dataset of images

Images of faces

I downloaded frontal faces with background images from two databases (bioIDface and Caltech databases)

Images on non-faces

It was a bit surprising that I have not found a database for this purpose (images without human faces for face recognition in order to be able to count the false positives), thus I have incorporated images from different databases/sources:

- images of backgrounds (without people and animals, both outdoor and indoor),
- images of wolfs and dogs,
- images of monkeys,
- images of distorted cartoon faces (I think it is a matter of question whether a cartoon face counts as a face or not. In my opinion these are not be detected as faces).

4 Testing face detection accuracy on test dataset

Detecting faces on the test dataset with recommended parameters.

Face detection testing results:

True positives:	418
False positives:	13
True negatives:	193
False negatives:	63
Accuracy:	0.89
Precision:	0.97
Recall:	0.87

Classifier:

```
faces = cascade.detectMultiScale(gray, scaleFactor=1.3,
        minNeighbors=5 ,minSize=(30,30))
```

5 Implementing new face detection from features based on face-part detection

Creating features from face-part detection results

After my experiences from the early tests I decided to incorporate 3 types of face detection and create features from them. Applied face-part detections:

- Eyes detection (cascade=eye_cascade, scalef=1.3,min_neighbors=5,minsize=(20,20))
- Mouth detection (low sensibility) (cascade=mouth_cascade, scalef=1.2,min_neighbors=100,minsize=(20,20))
- Mouth detection (high sensibility) (cascade=mouth_cascade, scalef=1.2,min_neighbors=10,minsize=(20,20)

After getting the rectangles from the face-part detections, I created these features:

- Number of eyes/mouths
- min/max/mean/median of the diagonals of the rectangles, for both the eyes and mouths
- Standard deviation of rectangles divided by the median of rectangles (to describe the spreading of the detected rectangles in relation with the scale, that can be characterized by the median of the rectangles' size)
- Distances between the centre of the eyes and the centre of the mouths

Implementing and testing a classifier for face detection using features from face parts

I implemented a classifiers using the features listed above. The aim was to decrease the number of false positives and false negatives. I tested RandomForest and SVM classifiers. Having optimized the parameters RandomForest gave a good result in decreasing false negatives (recall score: 0.93), yet SVM resulted in both better precision and recall score, then „default" face detection.

Face detection from face parts testing result

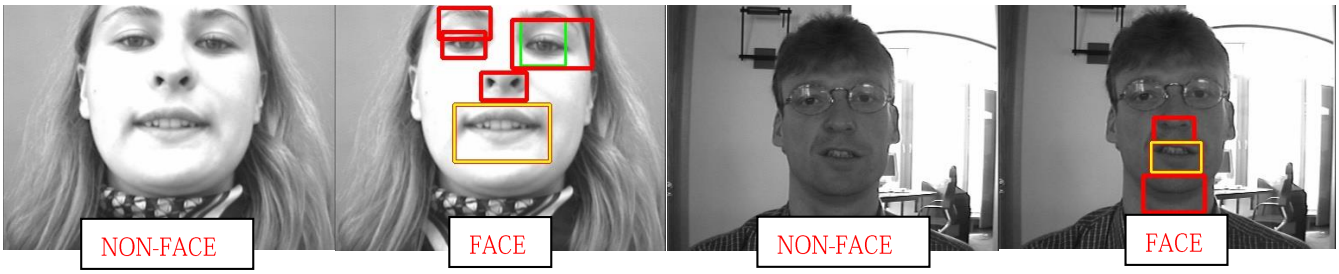
True positives:	434
False positives:	9
True negatives:	197
False negatives:	47
Accuracy:	0.92
Precision:	0.98
Recall:	0.90

Classifier:

```
clf = Pipeline([
    ('tr',StandardScaler()),
    ('clf_',  SVC(kernel='rbf', C=5000, gamma=0.00002,class_weight='balanced' ))
])
```

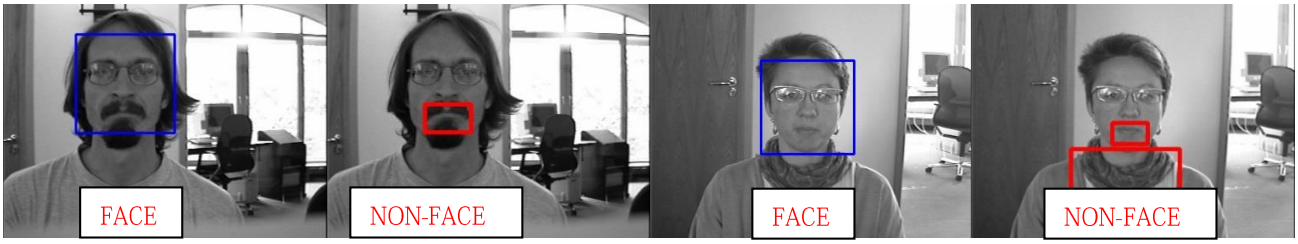
Examples for face detection comparing "default" and proposed method

Examples for cases where face-detection-from-face-parts is better than „default" face detection: (55 cases)



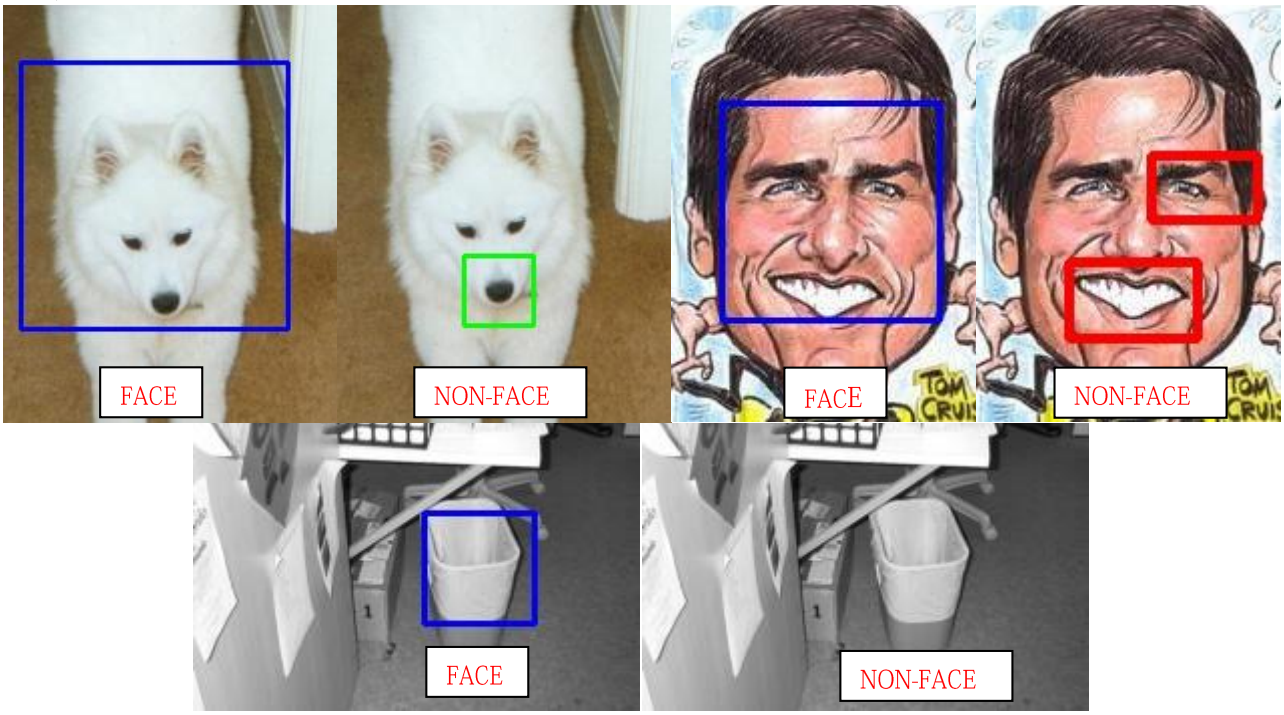
- Red Face – Non-face labels relate to the face detection results
- Rectangles:
  - red for mouth detection with high sensibility,
  - yellow for mouth detection with low sensibility,
  - green for eyes detection,
  - blue for the default face detection

Examples for cases where face-detection-from-face-parts is worse than „normal” face detection: (39 cases)

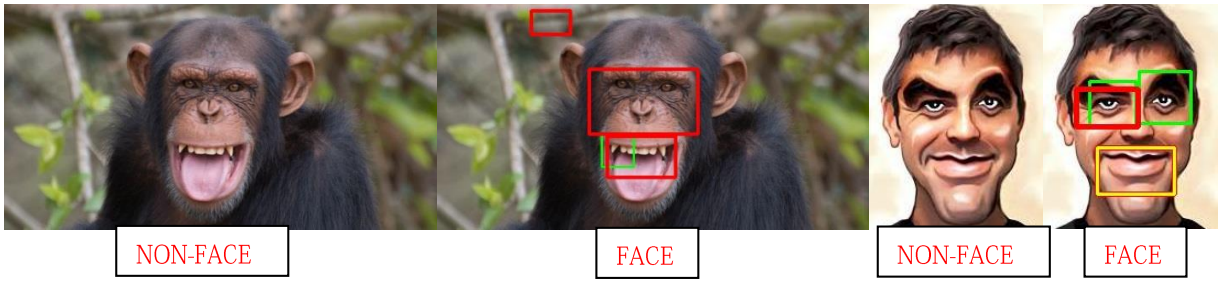


(most cases are images of these two people. The guestion of glasses is to be further investigated)

Examples for cases where face-detection-from-face-parts is better than „normal” face detection: (10 cases)



Examples for cases where face-detection-from-face-parts is worse than „normal” face detection: (6 cases)



6 Consequences

The proposed new method for face detection (using face-part detection) gave better results on test dataset (higher accuracy, precision and recall), thus for face detection on frontal-face images it may be a better method. However, it has yet to be tested on a bigger dataset. In addition, special cases (glasses, illumination, races, etc.) also needed to be investigated.

The new method may also be a promising way to be able to check the correctness of the "default" face detection method on non-labelled images. Although the new method gave better results, for this purpose its accuracy needs to be increased. Potential ways of improvements:

- involving nose detection
- creating new features from face-part detection
- testing other classification algorithms (adaboost, deep learning, etc.)
- testing feature transformation methods (e.g. PCA)
- testing face-part-detection parameters (scalefactor, minNeighbours)
- training on larger dataset