Ear Detection Using Haar Cascades

Assignment #2

Image Based Biometry 2020/21, Faculty of Computer and Information Science, University of Ljubljana

Urša Zrimšek

I. Introduction

This is a report of second assignment for the course Image Based Biometry. It describes the process of training a Haar Cascade using OpenCV library application called `opencv_traincascade`. The model behind the application is described in [1]. Data preparation, training commands and testing are additionally explained in my GitHub repository, where you can also find the cascades mentioned in this report.

II. Methodology

All the OpenCV aplications used are downloaded from SourceForge. In `README.md` of the mentioned repository you can find all commands I used during training.

For training the cascade we need a set of positive and a set of negative images. Positive images are taken from *AWEForSegmentation* dataset and then samples are created with application `opencv_createsamples.exe`. Apart from images, this application also needs a file, that tells how many ears are on each image and where they are. This file is created by function `positive_description_file()` found in jupyter notebook `data_preparation.ipynb` in the repository. `opencv_createsamples.exe` outputs a `pos.vec` file, containing the positive samples for training of size 24×24. I specified that when running the application.

Getting negative images is a bit more complicated, since those can be anything that doesn't include ears. First I tried building them from positive images, so that I covered all the ears with black and white squares. I used both to have twice as many negative images as positive. A cascade trained with them (in folder *cascade_basic*) detected a lot of false positives, so I decided to search for better negative images. I found a set of gray-scale images on Kaggle. This cascade (in folder *cascade_gs*) was detecting more true positives, but still the number of false positives was about 20 times bigger from true positives. I tried searching for background images similar of backgrounds where people could be, so I used *COCO dataset*, where I was able to search by image labels that tell us what the image contains. I downloaded 2017 Train/Val annotations from COCO webpage and selected images I want to download. I combined objects that should be included on the picture to get a certain place - living room, bedroom, office, kitchen, park,... (implemented in `data_preparation.ipynb`). As I didn't want to have any ears in my negative folder, I trained a cascade only with images that don't include people, but this didn't improve my cascade (in folder *cascade_coco*). It was still detecting a lot of ears on faces and palms of people on test images, so the next step was to also take photos that include people, and to manually delete those that include visible ears (cascade in folder *cascade_coco_people*). Now that proved valuable. The number of true positives increased and false negatives more than halved. We still need to adjust some parameters to get a good detector (that will be discussed in section III), but as goes for negative samples, I think we came to a good set. Later I also tried combining all the images, but the results weren't better, just training took twice as much time (cascade in folder *cascade_combined*).

The description file for negative images only includes location and names of all images. This are the `.txt` files in the repository.

When we have both sets of images and description files, we are ready to train the cascades. We can set a lot of arguments, I let most of them default, and told the application the directory name of the folder to store the cascade, names of files with positive and negative samples, set the number of stages to 10, number of positive samples used in training for every stage to 800 and number of negative samples used to a bit more then total number of them, so it uses all of them. I also needed to input the same sizes of training samples as I did at creating them - 24×24. Later, when optimizing the cascades, I was also changing the parameter `-mode` that selects a type of Haar features set used in training. The default value *BASIC* uses only upright features, while *ALL* uses the full set of upright and 45 degree rotated feature set. They both improved the cascades (specially the second one), but also slowed the training substantially. Since training already takes 25 minutes for the smallest set of negative images and 4 and a half hours for the biggest, I was testing different combinations only on the first two sets, and then trained the cascade with the best selection of negative images only with default and with best arguments.

When testing, I was accepting a detection as true positive (TP) if the intersection over union with one of the bounding boxes in the annotations was bigger than 0.5, otherwise it was a false positive (FP). I was also calculating *recall*, $TPR = \dfrac{\#TP}{\#\text{ears on images}}$, *precision*, $PPV = \dfrac{\#TP}{\#TP + \#FP}$ and *F1 score*, $F1 = 2 \cdot \dfrac{PPV \cdot TPR}{PPV + TPR}$ to easily compare different cascades.

III. Results

In this section we will compare test results for cascades trained on different data and with different training arguments. We will also try what are the best detection parameters that should be used when detecting ears with this cascades. Test data is also taken from *AWEForSegmentation* dataset and it is different from the training data we used for positive samples.

On Figure 1 we can see how many true and false positives detected the cascades trained with default arguments on different negative sets. For detection we need to specify two parameters: `minNeighbours` specifying how many neighbors each candidate rectangle should have to retain it - higher value results in less detections but with higher quality, and `scaleFactor` specifying how much the image size is reduced at each image scale. I chose `scaleFactor` 1.1 and `minNeighbours` 100, because with anything lower we would have even more false positives. We can see that the best negative set is the one with images from *COCO* that also include people. It has the lowest number of false positives, and only the cascade trained on combined negative images had more true positives, but with more than twice as many false positives and twice as long training time I decided to leave out this dataset in further testing.

On Figure 2 we are comparing cascades trained on the first set of data, with different combinations of training ar-

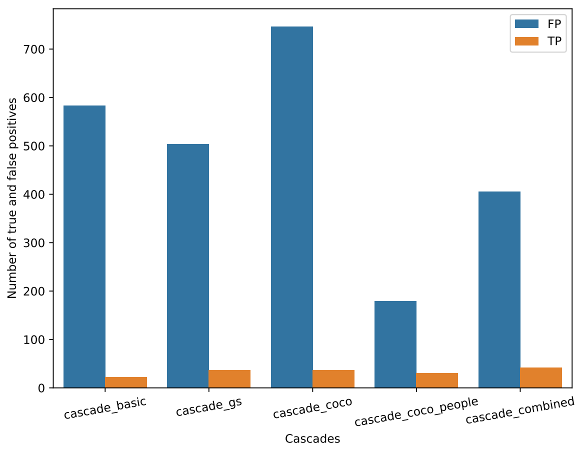

Figure 1. Number of true an false positives detected on test images by different cascades.

guments **mode** (**BASIC** or **ALL** - all in cascade name) and **maxFalseAlarmRate** (0.5 or 0.3 - maxFAR03 in name).

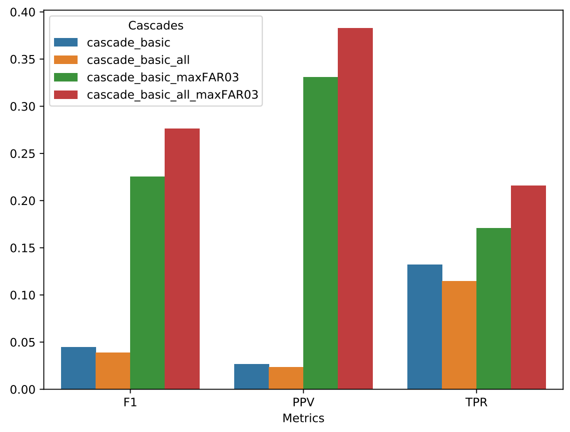

Figure 2. Evaluation of differently trained cascades.

From both plots we can conclude, that we will get the best detector if we train the cascade on data from *COCO* that includes people and use arguments **- mode ALL** and **-maxFalseAlarmRate 0.3**.

We still need to find the optimal detection parameters. Figures 3 and 4 show us F1 scores for different selection of **scaleFactor** and **minNeighbours**.

We can conclude, that if we want best detection, we need to use **cascade_coco_people_all_maxFAR03** with detection parameters **scaleFactor** 1.1 and **minNeighbours** 6.

IV. Conclusion

Bellow you can see the results of the cascade with parameters chosen above.

TP	FP	Precision	Recall	F1 score
71	104	0.41	0.25	0.31

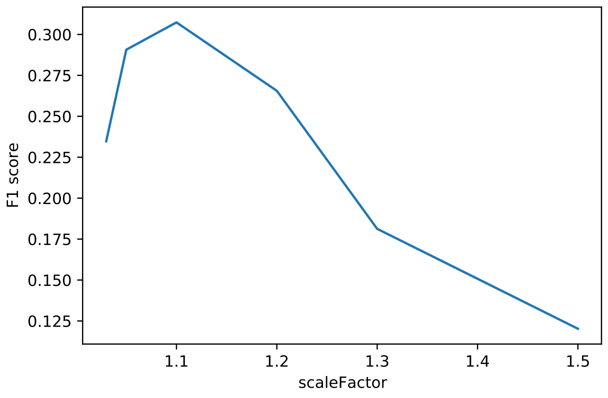

Figure 3. F1 scores of best cascade at different scale factors.

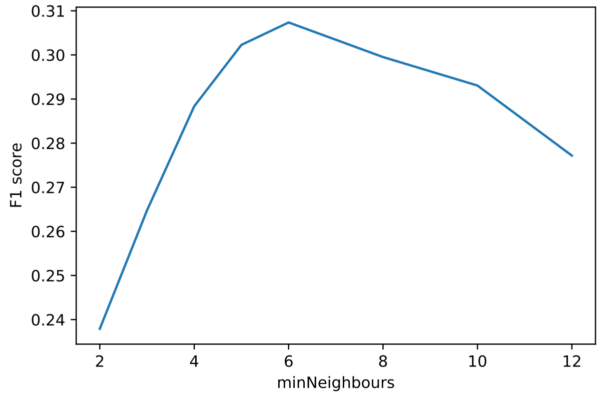

Figure 4. F1 scores of best cascade at different minNeighbours values.

This results are not comparable to the results deep neural networks achieve at detection, but at least we were able to improve the cascades a lot from where we started. The biggest problem is detecting too many false positives, since the main goal from the beginning was to minimize it.

Below we can see some examples of good and bad detections - green squares are the ears detected by our cascade. On the first picture the ear is detected perfectly, on second we can see a lot of false positives, and on the last, Arnold's right ear is detected, but the left that is a bit hidden is not.

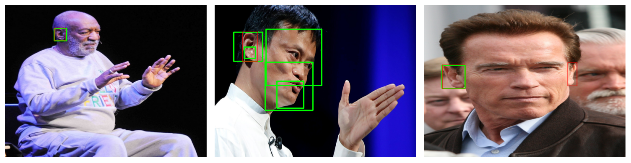

Figure 5. Examples of detection.

References

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1. IEEE, 2001, pp. I–I.