Ear detection/segmentation
Regular Track: Assignment #2

Image Based Biometry 2021/22

Anže Luzar (63170183)
Faculty of Computer and Information Science
University of Ljubljana

***Abstract*—This report describes the implementation of an ear detector for images. The steps of developing detectors are first presented and then I evaluate results for detector.**

I. Introduction

I have chosen detection of ears on the input images for the second assignment within the Image Based Biometry course. I have decided to use different approaches for implementing the detector including VJ and YOLOv5, and also different methods for evaluating my results.

II. Methodology

I have been using the AWE-W dataset [1], where ear locations from images are marked with the corresponding bounding boxes. Then I tried to do some pre-processing (e.g., histogram equalization, grayscaling) in order to observe if they improve the overall performance. My ear detector is written in Python, where I was developing in JetBrains PyCharm IDE. Here, I used mostly OpenCV Python library (see https://pypi.org/project/opencv-python/) and also scikit-learn (see https://pypi.org/project/scikit-learn/). I have implemented two main detectors - cascade using VJ (Viola-Jones) and YOLOv5 and have compared them later on within the evaluation. The evaluation part was done using several approaches such as average IoU (Intersection over Union), confusion matrix, accuracy, precision, recall (sensitivity), F1 score & (mean) average precision [2], [3].

III. Source code

The complete source code can be found in the *src* folder. The data (from AWE-W dataset) in *src/data*, the pre-processing files are in *src/preprocessing*, the detectors are in *src/detectors* and the final evaluation is present in *src/metrics*. The Python requirements are located in *requirements.txt* file and *run_evaluation.py* is used to run the evaluation of the implemented detectors. The code structure is also displayed in Figure 1.

IV. Preprocessing

The pre-processing part included implementing different techniques [4]. I have also used the already prepared histogram equalization method. The other designed methods were image grayscaling, gamma brightness correction in RGB and HSV color spaces, edge enhancement using Canny's edge-detection to sharpen the edges and a bilateral filter for smoothing images while preserving edges.

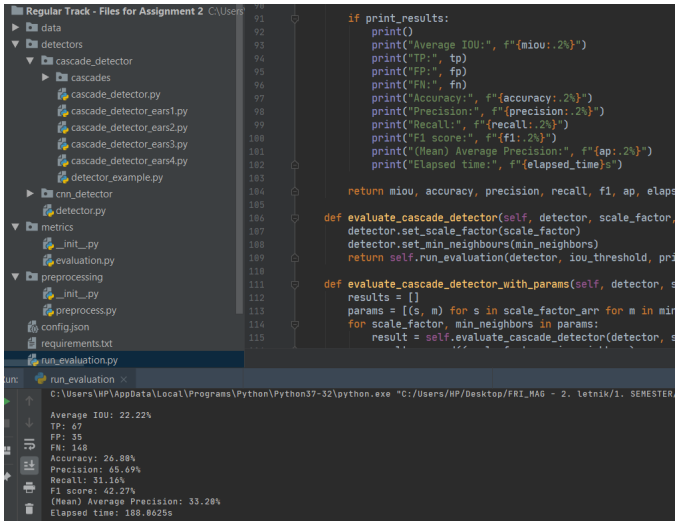

Figure 1: Code structure and files in PyCharm.

V. VJ cascade detector

This implementation has been done using already trained Viola–Jones cascade detector (VJ is a brute force framework for detection different modalities). The detector is located in *detectors/cascade_detector/cascade_detector_ears1.py*. Here, I have aimed to improve the existing cascade detector, which was detecting only faces. The detection of ears required having two Haar feature-based cascade classifiers - one Haar cascase for the left ear and the other for the right ear. The cascade classifiers were obtained from GitHub [5], [6] and are stored as XML files (*haarcascade_mcs_leftear.xml* and *haarcascade_mcs_rightear.xml*).

The VJ detection also requires setting the following two parameters:

- *scaleFactor*: parameter, which specifies how much the image size is reduced at each image scale.
- *minNeighbors*: parameter, which defines how many neighbors each candidate rectangle should have to retain it.

When choosing the aforementioned optimal parameters of classification I was trying to choose them in a ways to maximize the obtained F1 score, IoU and mAP. I have realized that some set of parameters doesn't work for every images and that there can be a lot of false positives and misdetections, so I needed to try with different combination of parameters. To calculate the precision and recall values I have each detected bounding box as True-positive (TP), False-positive (FP) or False-negative (FN) based on the IoU threshold (50% for instance) [7]. I have managed to obtain the best IoU when setting *scaleFactor* to 1.01 and *minNeighbors* to 2 (see Table I):

Table I: Best results for VJ detection

scaleFactor: 1.01%
minNeighbors: 2%
Average IOU: 32.71%
TP: 82
FP: 130
FN: 38
Accuracy: 32.80%
Precision: 38.68%
Recall: 68.33%
F1 score: 49.40%
(Mean) Average Precision: 58.80%
Elapsed time: 655.875s

(a) Good detection (b) False detection

Figure 2: Detected ears with VJ cascade detector

I have tried to apply different preprocessing methods but then realized that they only make VJ classification worse so I just kept the image as it is. I have also tried to search for other already implemented VJ detectors and cascade classifiers. I have found a lot of ear detection GitHub repositories from previous years (see [8], [9]). I tried to run those detectors and reuse the VJ classifiers, but the results were even worse than with the classifier I used. I have found out that my VJ classifier performs well when detecting ears from images with monotonous background behind the person (see Figure 2). On the contrary, the detection can be really bad when having backgrounds with different sections and colours.

I have compared the results for different values of *scaleFactor*

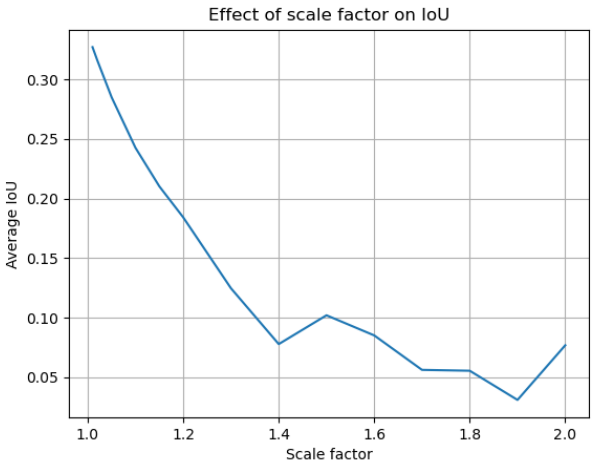

Figure 3: IoU based on scale factor.

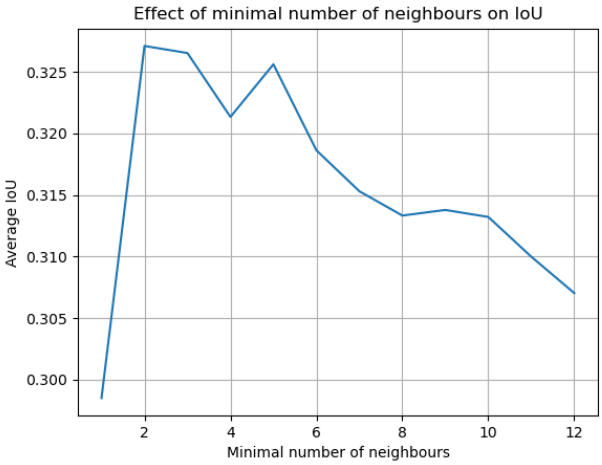

Figure 4: IoU based on min number of neighbours.

and *minNeighbors* (see Figure 3 and Figure 4). I also found out that execution time decreases by increasing *scaleFactor* increases by increasing *minNeighbors*.

VI. YOLOv5

In this part I have used YOLOv5 [10] AI architecture to train data and to detect ears using the model. Here I have followed instructions from https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data. I had to split the initial data a little more: the training set was left intact and I have split the test data into two parts: test data and validation data. So, the whole data was split like 75-15-10 (train-test-val). I have also normalized all the annotations (with prepared bash and python scripts) because box coordinates must be in normalized *xywh* format (from 0-1), so I have divided x and w by image width, and y and h by image height.

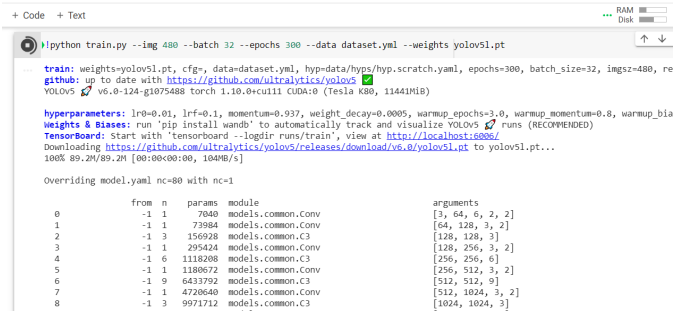

Figure 5: YOLOv5 model training in Google Colab environment.

The model training was done within Colaboratory [11] environment. I have prepared *dataset.yml* file (with train, test and val YAML entries), uploaded normalized data and used the scripts from the YOLOv5 GitHub repository. I have set batch size to 16 and used 100 epochs with YOLOv5s pretrained model to train the data. For the second training I used batch size 32, 100 epochs and YOLOv5m model (take a look at Figure 5). Model was then tested similarly as for the VJ detection. The first training results were bad with Average IoU only 29.60% and mAP 69.20% (shown in Figure 6). The second training

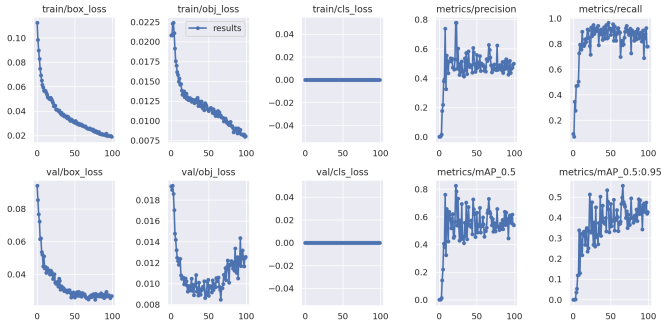

Figure 6: First YOLOv5 training and testing results (16 batches, 100 epochs, YOLOv5s).

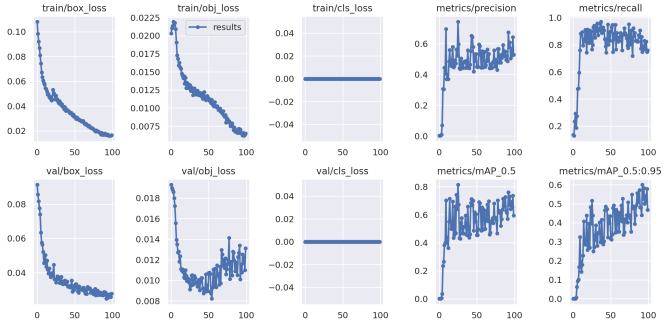

Figure 7: Second YOLOv5 training and testing results (32 batches, 100 epochs, YOLOv5m).

results were a little bit better with Average IoU 33.97% and mAP 73.20% (see Figure 7 and Table 2). I have continued with training and used different parameters, so at the end all my models were the following:

- model trained with batch size 8, 150 epochs and YOLOv5l pre-trained model,
- model trained with batch size 16, 100 epochs and YOLOv5s pre-trained model,
- model trained with batch size 16, 300 epochs and YOLOv5s pre-trained model,
- model trained with batch size 16, 300 epochs and YOLOv5m pre-trained model, and
- model trained with batch size 32, 100 epochs and YOLOv5m pre-trained model.

VII. Results and discussion

Within this ear detection assignment I have managed to implement and apply VJ and YOLOv5 ear detectors and compared their results based on different parameters. As expected, the results for detection are not the best possible. Looking at different measures (especially IoU) it is evident that the models could be improved. The results of VJ cascade detection could be improved by optimizing the training parameters and by using more sample images (with different ear positions, lightning conditions and noisy backgrounds) to obtain better VJ classifiers. The results for YOLOv5 were also not the best possible and could be improved by using more annotated images with higher resolution and also by testing different training parameters (for example we could use machines with more GPU resources and set higher batch size and epochs and use the most accurate YOLOv5x pretrained model). During the training I had a lot of problems with Google Colab loosing connection to runtime, so another improvement could be done by training on my local GPU.

Table II: Best results for YOLOv5 detection

Average IOU: 33.97%
TP: 79
FP: 157
FN: 14
Accuracy: 31.60%
Precision: 33.47%
Recall: 84.95%
F1 score: 48.02%
(Mean) Average Precision: 73.20%
Elapsed time: 194.904255188s

References

[1] "AWE dataset." [Online]. Available: http://ears.fri.uni-lj.si/datasets.html#awe-full
[2] A. Fawzy Gad, "Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall," 2020. [Online]. Available: https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy
[3] ——, "Evaluating Object Detection Models Using Mean Average Precision (mAP)," 2020. [Online]. Available: https://blog.paperspace.com/mean-average-precision/
[4] P. Canuma, "Image Pre-processing," 2018. [Online]. Available: https://prince-canuma.medium.com/image-pre-processing-c1aec0be3edf
[5] "ViolaJonesCascades," 2019. [Online]. Available: https://github.com/otsedom/ViolaJonesCascades
[6] M. Castrillón Santana, J. Lorenzo Navarro, and D. Hernández Sosa, "An study on ear detection and its applications to face detection," in *Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA)*, La Laguna, Spain, November 2011.
[7] I. Ralašić, "A Better mAP for Object Detection," 2021. [Online]. Available: https://towardsdatascience.com/a-better-map-for-object-detection-32662767d424
[8] N. Bevk, "OpenCV ear detection," 2020. [Online]. Available: https://github.com/nebevk/opencv_ear_detection
[9] J. Nastran, "ear_detect," 2020. [Online]. Available: https://github.com/NastranJurij/ear_detect/
[10] "YOLOv5." [Online]. Available: https://github.com/ultralytics/yolov5
[11] "Colaboratory." [Online]. Available: https://colab.research.google.com/?utm_source=scs-index