UNIVERSITY OF PADOVA

Computer Vision

Intermediate project report

Marek Tatýrek
Mateusz Miroslaw Lis
Abioye Obaloluwa Peter

Academic Year 2024-2025

Contents

1 Introduction

1.1 Notes

To make code more clear, we use:

- `snake_case` for variables
- `PascalCase` for classes
- `camelCase` for functions

2 Overview

We decided to use class design, where classes are logicaly separated depending on task. Dataflow between classes is provided by several user structures. Data are processed squentialy. Unfortunately used dataflow is not that clear because we havent been able to fully follow originally defined structure.

For loading data we have two classes `ImageLoader()` and `VideoLoader()`, derived from class `InputSource()`. From user side the classes have common interface and usage is the same. Important is method `hasNextFrame()`, used for driving the while loop in the main, returning true in case there is frame or image that we can read. Class is returning timestamp, cv::Mat with current sample.

As was already said, tasks are performed sequentialy, which is convenient, because we are reading many frames and to store them in the buffer, it will be very memory demanding. Also from the same reason we have to take care about not making deep copies.

We have `Preprocessing()` class, for editing the image before actual detection of haar features.

For detecting features in the images we are using class `HaarDetector()` using haar cascades for finding desired patterns in the images. In our case faces and eyes.

From the detected features we need to make evaluation and decide which shot type are we having. For this purpouse we have class `FeatureEvaluator()` that outputs structure `classification_result` with information if is curent sample wide shot, medium or close up.

Because we want to make statictic from the data, we made `FilmStatistics()` class. It makes sense to use this class only on video data. At the init we provide configuration structure `FilmStatisticsEvalConfig`, with many settings. We can export time sequences to `.csv` file or we can use getters to get the time sequences and use them in the code.

For graphical output of the statstic data we have `ResultDisplayer()` class, which is inputting all data types got from `FilmStatistics()` getters and returning `cv::mat` with plots.

3 Implementation details

4 Evaluation

5 Dataset description

6 Results summary

7 Individual contributions

8 Conclusion

References