cmc

Release 0.1.0

Daniel Helm

May 29, 2020

CONTENTS

This python package is developed within the project Visual History of the Holocaust[1] (VHH) started in Januray 2019. The major objective of this package is to provide interfaces and functions to classify image sequences (shots) in one out the four classes: Extreme Long Shot (ELS), Long Shot (LS), Medium Shot (MS) or Close-Up shot (CU). Those classes are the most significant cinematographic camera settings and represent the distance between an subject and the camera.

This software package is installable and designed to reuse it in customized applications such as the vhh_core[2] package. This module represents the main controller in the context of the VHH project.

This documentation provides an API description of all classes, modules and member functions as well as the required setup descriptions.

[1] https://www.vhh-project.eu/
[2] https://github.com/dahe-cvl/vhh_core

CHAPTER

ONE

METHODOLOGY

CHAPTER

TWO

PACKAGE OVERVIEW

The following list give an overview of the folder structure of this python repository:

name of repository: vhh_stc

- **ApiSphinxDocumentation/**: includes all files to generate the documentation as well as the created documentations (html, pdf)
- **config/**: this folder includes the required configuration file
- **stc/**: this folder represents the shot-type-classification module and builds the main part of this repository
- **Demo/**: this folder includes a demo script to demonstrate how the package have to be used in customized applications
- **Develop/**: includes scripts to train and evaluate the pytorch models. Furthermore, a script is included to create the package documentation (pdf, html)
- **README.md**: this file gives a brief description of this repository (e.g. link to this documentation)
- **requirements.txt**: this file holds all python lib dependencies and is needed to install the package in your own virtual environment
- **setup.py**: this script is needed to install the stc package in your own virtual environment

CHAPTER

THREE

SETUP INSTRUCTIONS

This package includes a setup.py script and a requirements.txt file which are needed to install this package for custom applications. The following instructions have to be done to used this library in your own application:

Requirements:

- Ubuntu 18.04 LTS
- CUDA 10.1 + cuDNN
- python version 3.6.x

Create a virtual environment:

- create a folder to a specified path (e.g. /xxx/vhh_stc/)
- python3 -m venv /xxx/vhh_stc/

Activate the environment:

- source /xxx/vhh_stc/bin/activate

Checkout vhh_stc repository to a specified folder:

- git clone https://github.com/dahe-cvl/vhh_stc

Install the stc package and all dependencies:

- change to the root directory of the repository (includes setup.py)
- python setup.py install

Note: You can check the success of the installation by using the commend *pip list*. This command should give you a list with all installed python packages and it should include *vhh_stc*

Note: Currently there is an issue in the *setup.py* script. Therefore the pytorch libraries have to be installed manually by running the following command: *pip install torch==1.5.0+cu101 torchvision==0.6.0+cu101 -f https://download.pytorch.org/whl/torch_stable.html*

CHAPTER

FOUR

API DESCRIPTION

This section gives an overview of all classes and modules in *cmc* as well as an code description.

4.1 Configuration class

class cmc.Configuration.**Configuration**(*config_file: str*)
Bases: object

This class is needed to read the configuration parameters specified in the configuration.yaml file. The instance of the class is holding all parameters during runtime.

Note: e.g. ./config/config_vhh_test.yaml

the yaml file is separated in multiple sections config['Development'] config['PreProcessing'] config['CmcCore'] config['Evaluation']

whereas each section should hold related and meaningful parameters.

---

loadConfig()
Method to load configurables from the specified configuration file

4.2 CMC class

class cmc.CMC.**CMC**(*config_file: str*)
Bases: object

Main class of camera movements classification (cmc) package.

exportCmcResults(*fName*, *cmc_results_np: numpy.ndarray*)
Method to export cmc results as csv file.

Parameters

- **fName** – [required] name of result file.
- **cmc_results_np** – numpy array holding the camera movements classification predictions for each shot of a movie.

loadSbdResults(*sbd_results_path*)
Method for loading shot boundary detection results as numpy array

Note: Only used in debug_mode.

Parameters **sbd_results_path** – [required] path to results file of shot boundary detection module (vhh_sbd)

Returns numpy array holding list of detected shots.

runOnSingleVideo(*shots_per_vid_np=None*, *max_recall_id=- 1*)
Method to run cmc classification on specified video.

Parameters

- **shots_per_vid_np** – [required] numpy array representing all detected shots in a video (e.g. sid | movie_name | start | end)
- **max_recall_id** – [required] integer value holding unique video id from VHH MMSI system

4.3 OpticalFlow class

class cmc.OpticalFlow.**OpticalFlow**(*video_frames=None*, *fPath=''*, *debug_path=''*, *sf=0*, *ef=1*, *mode=0*, *pan_classifier=<cmc.OpticalFlow.AngleClassifier object>*, *tilt_classifier=<cmc.OpticalFlow.AngleClassifier object>*, *sensitivity=20*, *specificity=3*, *border=50*, *number_of_features=100*, *angle_diff_limit=20*, *config=None*)
Bases: object

This class is used for optical flow calculation.

__init__(*video_frames=None*, *fPath=''*, *debug_path=''*, *sf=0*, *ef=1*, *mode=0*, *pan_classifier=<cmc.OpticalFlow.AngleClassifier object>*, *tilt_classifier=<cmc.OpticalFlow.AngleClassifier object>*, *sensitivity=20*, *specificity=3*, *border=50*, *number_of_features=100*, *angle_diff_limit=20*, *config=None*)
Constructor.

Parameters

- **video_frames** – This parameter holds a valid numpy array representing a range of frames (e.g. NxWxHxchannels).
- **fPath** – This parameter holds a valid path consisting of the absolute path and the correct video name.
- **debug_path** – This parameter specifies a valid path to store results in debug mode.
- **sf** – This parameter represents the starting frame index.
- **ef** – This parameter represents the ending frame index.
- **mode** – This parameter represents runtime mode (e.g. DEBUG_MODE=1 or SAVE_MODE=2).
- **pan_classifier** – This parameter holds a valid object of class type AngleClassifier.
- **tilt_classifier** – This parameter holds a valid object of class type AngleClassifier.
- **sensitivity** – This parameter is used to configure the optical flow algorithm.
- **specificity** – This parameter is used to configure the optical flow algorithm.

- **border** – This parameter is used to configure the optical flow algorithm.
- **number_of_features** – This parameter is used to configure the optical flow algorithm.
- **angle_diff_limit** – This parameter is used to configure the optical flow algorithm.
- **config** –

optical_flow(*prev_frame*, *prev_feat*, *curr_frame*)
This method is used to calculate the optical flow between two consecutive frames.

Parameters

- **prev_frame** – This parameter must hold a valid numpy frame (previous).
- **prev_feat** – This parameter must hold valid features of the previous frame.
- **curr_frame** – This parameter must hold a valid numpy frame (current).

Returns This method returns two arrays including the features of the previous frame as well as of the current frame.

run()
This method is used to run the optical flow calculation process.

Returns This method returns a separate list for each movement class and holds the predicted frame ranges of both.

run_eval()
This method is used to run the optical flow calculation to evaluate specified videos.

run_manual_evaluation()
This method is used to run optical flow process in DEBUG mode. A valid X-Server is needed to visualize the frame player.

4.4 PreProcessing class

class cmc.PreProcessing.**PreProcessing**(*config_instance:* cmc.Configuration.Configuration)
Bases: object

This class is used to pre-process frames.

applyTransformOnImg(*image: numpy.ndarray*) → numpy.ndarray
This method is used to apply the configured pre-processing methods on a numpy frame.

Parameters **image** – This parameter must hold a valid numpy image (WxHxC).

Returns This methods returns the preprocessed numpy image.

convertRGB2Gray(*img: numpy.ndarray*)
This method is used to convert a RBG numpy image to a grayscale image.

Parameters **img** – This parameter must hold a valid numpy image.

Returns This method returns a grayscale image (WxHx1).

crop(*img: numpy.ndarray*, *dim: tuple*)
This method is used to crop a specified region of interest from a given image.

Parameters

- **img** – This parameter must hold a valid numpy image.

- **dim** – This parameter must hold a valid tuple including the crop dimensions.

Returns This method returns the cropped image.

resize(*img: numpy.ndarray*, *dim: tuple*)
This method is used to resize a image.

Parameters

- **img** – This parameter must hold a valid numpy image.
- **dim** – This parameter must hold a valid tuple including the resize dimensions.

Returns This method returns the resized image.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

5.1 References

INDEX

Symbols

A

C

E

L

O

P

R