VHH Plugin Package: Shot Boundary Detection (vhh_sbd)

Release 1.0.0

Daniel Helm

Jun 07, 2020

CONTENTS

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

name of repository: vhh_sbd

- **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 evaluate the implemented approach as well as several helper scripts used during** development stage. 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

ONE

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_sbd/)
- python3 -m venv /xxx/vhh_sbd/

Activate the environment:

- source /xxx/vhh_sbd/bin/activate

Checkout vhh_sbd repository to a specified folder:

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

Install the sbd 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_sbd*

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

TWO

API DESCRIPTION

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

2.1 Configuration class

class sbd.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['SbdCore'] config['Evaluation']

whereas each section should hold related and meaningful parameters.

---

loadConfig()

Method to load configurables from the specified configuration file

2.2 CandidateSelection class

class sbd.DeepSBD.**CandidateSelection**(*config_instance:* sbd.Configuration.Configuration)

Bases: object

This class is used for sbd candidate selection. It detects frames ranges of about 16 frames which includes an abrupt cut. The loaded model is pre-trained on the deepsbd dataset.

run(*video_path*)

This method is used to run the candidate selection process.

Parameters **video_path** – This parameter must hold a valid path to a video file.

Returns This method returns a numpy array with a list of all detected frames ranges.

2.3 Evaluation class

class sbd.Evaluation.**Evaluation**(*config_file: str*)

Bases: object

This class is used to evaluate the implemented algorithm.

calculateEvaluationMetrics()

This method is used to calculate the evaluation metrics.

Returns This methods returns a numpy array including a list of the calculated metrics (precision, recall, ...).

calculateMetrics(*tp_cnt*, *fp_cnt*, *tn_cnt*, *fn_cnt*)

This method is used to calculate the evaluation metrics precision, recall and f1score.

Parameters

- **tp_cnt** – This parameter must hold a valid integer representing the tp counter.
- **fp_cnt** – This parameter must hold a valid integer representing the fp counter.
- **tn_cnt** – This parameter must hold a valid integer representing the tn counter.
- **fn_cnt** – This parameter must hold a valid integer representing the fn counter.

Returns This method returns the scores for precision, recall, accuracy, f1_score, tp_rate and fp_rate.

calculateSimilarityMetric(*results_np: numpy.ndarray*, *threshold=4.5*)

This method is used to calculate the similarity metrics based on the pre-calculated raw results.

Parameters

- **results_np** – This parameter must hold a valid numpy array.
- **threshold** – This parameter holds a threshold. (default: 4.5)

Returns This method returns a numpy array including the final shot boundaries.

evaluation(*result_np*, *vid_name*)

This method is needed to evaluate the gathered results for a specified video.

Parameters

- **result_np** – This parameter must hold a valid numpy array.
- **vid_name** – This parameter represents a video name.

Returns This method returns the calculated TP, TN, FP and FN counters.

export2CSV(*data_np: numpy.ndarray*, *header: str*, *filename: str*, *path: str*)

This method is used to export the gathered results to a csv file.

Parameters

- **data_np** – This parameter holds a valid numpy array.
- **header** – This parameter holds a csv header line (first line in the file - semicolon seperated).
- **filename** – This parameter must hold a valid file name.
- **path** – THis parameter must hold a valid file path.

exportMovieResultsToCSV(*fName*, *res_np*)

This method is used to export video results to csv file.

Parameters

- **filepath** – This parameter must hold a valid file_path.
- **res_np** – This parameter must hold a valid numpy array containing the final results.

loadRawResultsFromCsv(*filepath*)
This method is used to load raw results from csv file.

Parameters **filepath** – This parameter must hold a valid file_path.

Returns This method returns a numpy array containing the raw_results.

loadRawResultsFromNumpy(*filepath*)
This method is used to load raw results from numpy array.

Parameters **filepath** – This parameter must hold a valid file_path.

Returns This method returns a numpy array containing the raw_results.

loadResultsFromCSV(*filepath*)
This method is used to load final results from csv file.

Parameters **filepath** – This parameter must hold a valid file_path.

Returns This method returns a numpy array containing the final results.

plotPRCurve(*results_np*)
This method is needed to create and plot the precision_recall curve.

Parameters **results_np** – This parameter must hold a vaild numpy array including the precision and recall scores.

plotROCCurve(*results_np*)
This method is needed to create and plot the roc curve.

Parameters **results_np** – This parameter must hold a vaild numpy array including the precision and recall scores.

run()
This method is needed to run the evaluation process.

2.4 PreProcessing class

class sbd.PreProcessing.**PreProcessing**(*config_instance:* sbd.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.

applyTransformOnImgSeq(*img_seq: numpy.ndarray*) → numpy.ndarray

claHE(*img: numpy.ndarray*)
This method is used to calculate the Contrast Limited Adaptive Histogram Equalization.

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

Returns This method returns the pre-processed image.

classicHE(*img: numpy.ndarray*)
This method is used to calculate the classic histogram equalization.

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

Returns This method returns the pre-processed 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.

2.5 SBD class

class sbd.SBD.**SBD**(*config_file: str*)
Bases: object

Main class of shot boundary detection (sbd) package.

calculateDistance(*x*, *y*)
This method is used to calculate the distance between 2 feature vectors.

Parameters

- **x** – This parameter represents a feature vector (one-dimensional)
- **y** – This parameter represents a feature vector (one-dimensional)

Returns This method returns the similarity score of a specified distance metric.

convertShotBoundaries2Shots(*shot_boundaries_np: numpy.ndarray*)
This method converts a list with detected shot boundaries to the final shots.

Parameters **shot_boundaries_np** – This parameter must hold a numpy array with all detected shot boundaries.

Returns This method returns a numpy list with the final shots.

exportFinalResultsToCsv(*shot_l: list*, *name: str*)
This method is used to export the final results to a csv file (semicolon seperated). :param shot_l: This parameter must hold a valid array list including the final results list. :param name: This parameter represents the name of the csv list.

exportRawResultsAsCsv_New(*results_np: numpy.ndarray*)
This method is used to export the raw results to a csv file.

Parameters **results_np** – This parameter must hold a valid numpy list including the raw results.

exportRawResultsAsNumpy(*results_np: numpy.ndarray*)
This method is used to export the raw results to a numpy file.

Parameters **results_np** – This parameter must hold a valid numpy list including the raw results.

runOnFolder()
This method is used to run sbd on all video files included in a specified folder.

Returns This method returns a numpy list of all detected shots in all videos.

runOnSingleVideo(*video_filename*, *max_recall_id=- 1*)
Method to run sbd on specified video.

Parameters

- **video_filename** – This parameter must hold a valid video file path.
- **max_recall_id** – [required] integer value holding unique video id from VHH MMSI system

runWithCandidateSelection(*candidates_np*)
This method is used to run sbd with candidate selection mode.

Parameters **candidates_np** – THis parameter must hold a valid numpy list including all pre-selected candidates.

Returns This method returns a numpy list with all detected shots in a video.

runWithoutCandidateSelection(*src_path*, *vid_name*)
This method is used to run sbd without candidate selection mode.

Parameters

- **src_path** – THis parameter must hold a valid path to the video file.
- **vid_name** – This parameter must hold a valid videofile name.

Returns This method returns a numpy list with all detected shots in a video.

2.6 Shot class

class sbd.Shot.**Shot**(*sid*, *movie_name*, *start_pos*, *end_pos*)
Bases: object

This class represents on shot and contains shot properties such as start/end frame index of a shot, shot-id and video_name.

convert2String()
This method is used to convert all properties of a shot into a semicolon-separated string. :return:

printShotInfo()
This method is used to print all properties of a shot.

2.7 Video class

class sbd.Video.**Video**
Bases: object

This class represents on video and contains properties such as dimensions, length, format, video name ... .

getFrame(*frame_id: int*) → numpy.ndarray
This method is used to return one frame specified with a given frame index.

Parameters **frame_id** – This parameter must hold a valid integer frame index.

Returns This method returns a numpy frame with a specified index (position).

load(*vidFile: str*)
This method is used to load a video of a specified storage.

Parameters **vidFile** – This parameter must hold a valid video file path.

printVIDInfo()
This method is used to print all video properties.

2.8 Utils module

class sbd.utils.**STDOUT_TYPE**
Bases: object

This class represents message types.

ERROR = 2

INFO = 1

sbd.utils.**getCommandLineParams**()
This function is used to read commandline parameters (e.g. just used in development stage) :return: list of parameters.

sbd.utils.**printCustom**(*msg: str*, *type: int*)
This function represents a customized print function (error/info msg). :param msg: Message to print. :param type: Type of message (info or error).

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

3.1 References

PYTHON MODULE INDEX

S

INDEX

S

V