Keras Video Generator

Patrice Ferlet

Nov 22, 2019

CONTENTS:

CHAPTER

ONE

VIDEO GENERATOR PACKAGE

Provides generators for video sequence that can be injected in Time Distributed layer.

CHAPTER

TWO

VIDEOFRAMEGENERATOR - SIMPLE GENERATOR

A simple frame generator that takes distributed frames from videos. It is useful for videos that are scaled from frame 0 to end and that have no noise frames.

class keras_video.generator.**VideoFrameGenerator**(*rescale=0.00392156862745098*, *nb_frames: int = 5*, *classes: list = []*, *batch_size: int = 16*, *use_frame_cache: bool = False*, *target_shape: tuple = (224, 224)*, *shuffle: bool = True*, *transformation: keras.preprocessing.image.ImageDataGenerator = None*, *split: float = None*, *nb_channel: int = 3*, *glob_pattern: str = './videos/{classname}/*.avi'*, *_validation_data: list = None*)

Create a generator that return batches of frames from video

- rescale: float fraction to rescale pixel data (commonly 1/255.)
- nb_frames: int, number of frames to return for each sequence
- classes: list of str, classes to infer
- batch_size: int, batch size for each loop
- **use_frame_cache: bool, use frame cache (may take a lot of memory for** large dataset)
- shape: tuple, target size of the frames
- shuffle: bool, randomize files
- transformation: ImageDataGenerator with transformations
- split: float, factor to split files and validation
- nb_channel: int, 1 or 3, to get grayscaled or RGB images
- **glob_pattern: string, directory path with '{classname}' inside that** will be replaced by one of the class list
- _validation_data: already filled list of data, do not touch !

You may use the "classes" property to retrieve the class list afterward.

The generator has that properties initialized:

- classes_count: number of classes that the generator manages
- files_count: number of video that the generator can provides
- classes: the given class list

- **files: the full file list that the generator will use, this** is usefull if you want to remove some files that should not be used by the generator.

get_validation_generator()
Return the validation generator if you've provided split factor

on_epoch_end()
Called by Keras after each epoch

CHAPTER

THREE

SLIDING FRAMES

That module provides the SlidingFrameGenerator that is helpful to get more sequence from one video file. The goal is to provide decayed sequences for the same action.

class keras_video.sliding.**SlidingFrameGenerator**(**args*, *sequence_time: int = None*, ***kwargs*)

SlidingFrameGenerator is useful to get several sequence of the same "action" by sliding the cursor of video. For example, with a video that have 60 frames using 30 frames per second, and if you want to pick 6 frames, the generator will return:

- one sequence with frame *[0, 5, 10, 15, 20, 25]*
- then *[1, 6, 11, 16, 21, 26])*
- and so on to frame 30

If you set *sequence_time* parameter, so the sequence will be reduce to the given time.

params:

- **sequence_time: int seconds of the sequence to fetch, if None, the entire** vidoe time is used

from VideoFrameGenerator:

- rescale: float fraction to rescale pixel data (commonly 1/255.)
- nb_frames: int, number of frames to return for each sequence
- classes: list of str, classes to infer
- batch_size: int, batch size for each loop
- **use_frame_cache: bool, use frame cache (may take a lot of memory for** large dataset)
- shape: tuple, target size of the frames
- shuffle: bool, randomize files
- transformation: ImageDataGenerator with transformations
- split: float, factor to split files and validation
- nb_channel: int, 1 or 3, to get grayscaled or RGB images
- **glob_pattern: string, directory path with '{classname}' inside that** will be replaced by one of the class list

get_validation_generator()

Return the validation generator if you've provided split factor

on_epoch_end()

Called by Keras after each epoch

CHAPTER

FOUR

OPTICAL FLOW GENERATOR

Warning: This module is not stable !

The purpose of that module is to return optical flow sequences from a video.

Several methods are defined:

- **Use standard optical flow** METHOD_OPTICAL_FLOW=1
- **Use optical flow as a mask on video** METHOD_FLOW_MASK=2
- **Use absolute diff mask on video** METHOD_DIFF_MASK=3
- **Use abs diff** METHOD_ABS_DIFF=4

class keras_video.flow.**OpticalFlowGenerator**(**args*, *nb_frames=5*, *method=1*, *flowlevel=3*, *iterations=3*, *winsize=15*, ***kwargs*)

Generate optical flow sequence from frames in videos. It can use different methods.

params:

- **method: METHOD_OPTICAL_FLOW, METHOD_FLOW_MASK, METHOD_DIFF_MASK,** METHOD_ABS_DIFF
- flowlevel: integer that give the flow level to calcOpticalFlowFarneback
- iterations: integer number of iterations for calcOpticalFlowFarneback
- winsize: flow window size for calcOpticalFlowFarneback

from VideoFrameGenerator:

- rescale: float fraction to rescale pixel data (commonly 1/255.)
- nb_frames: int, number of frames to return for each sequence
- classes: list of str, classes to infer
- batch_size: int, batch size for each loop
- **use_frame_cache: bool, use frame cache (may take a lot of memory for** large dataset)
- shape: tuple, target size of the frames
- shuffle: bool, randomize files
- transformation: ImageDataGenerator with transformations
- split: float, factor to split files and validation
- nb_channel: int, 1 or 3, to get grayscaled or RGB images

- **glob_pattern: string, directory path with '{classname}' inside that** will be replaced by one of the class list

absdiff(*images*)
Get absolute differences between 2 images

diff_mask(*images*)
Get absolute diff mask, then merge frames and apply the mask

flow_mask(*images*)
Get optical flow on images, then merge images and apply the mask

get_validation_generator()
Return the validation generator if you've provided split factor

make_optical_flow(*images*)
Process Farneback Optical Flow on images

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

k

INDEX