cvbase Documentation

Release 0.5.5

Kai Chen

Oct 12, 2018

Contents

CHAPTER 1

Introduction

cvbase is a miscellaneous set of tools which maybe helpful for computer vision research. It comprises the following parts.

- IO helpers
- Image/Video operations
- OpenCV wrappers for python2/3 and opencv 2/3
- Timer
- Progress visualization
- Plotting tools
- Object detection utils

Try and start with

```
pip install cvbase
```

See documentation for more features and usage.

1.1 Some popular features

There are some popular features such as progress visualization, timer, video to frames/frames to videos.

- Progress visualization

  If you want to apply a method to a list of items and track the progress, track_progress is a good choice. It will display a progress bar to tell the progress and ETA.

```
import cvbase as cvb

def func(item):
```

(continues on next page)

(continued from previous page)

```
    # do something
    pass

tasks = [item_1, item_2, ..., item_n]

cvb.track_progress(func, tasks)
```

The output is like the following.

There is another method `track_parallel_progress`, which wraps multiprocessing and progress visualization.

```
import cvbase as cvb

def func(item):
    # do something
    pass

tasks = [item_1, item_2, ..., item_n]

cvb.track_parallel_progress(func, tasks, 8)
# 8 workers
```

- Timer

  It is convinient to computer the runtime of a code block with `Timer`.

  ```
  import time

  with cvb.Timer():
      # simulate some code block
      time.sleep(1)
  ```

  Or try a more flexible way.

  ```
  timer = cvb.Timer()
  # code block 1 here
  print(timer.since_start())
  # code block 2 here
  print(timer.since_last_check())
  print(timer.since_start())
  ```

- Video/Frames conversion

  To split a video into frames.

  ```
  video = cvb.VideoReader('video_file.mp4')
  video.cvt2frames('frame_dir')
  ```

  Besides `cvt2frames`, `VideoReader` wraps many other useful methods to operate a video like a list object, like

  ```
  video = cvb.VideoReader('video_file.mp4')
  len(video)  # get total frame number
  video[5]  # get the 6th frame
  for img in video:  # iterate over all frames
      print(img.shape)
  ```

To generate a video from frames, use the `frames2video` method.

```
video = cvb.frames2video('frame_dir', 'out_video_file.avi', fps=30)
```

- Video editing (needs ffmpeg)

  To cut a video.

  ```
  cvb.cut_video('input.mp4', 'output.mp4', start=3, end=10)
  ```

  To join two video clips.

  ```
  cvb.concat_video(['clip1.mp4', 'clip2.mp4'], 'output.mp4')
  ```

  To resize a video.

  ```
  cvb.resize_video('input.mp4', 'resized.mp4', (360, 240))
  # or
  cvb.resize_video('input.mp4', 'resized.mp4', ratio=2)
  ```

  To convert the format of a video.

  ```
  cvb.convert_video('input.avi', 'output.mp4', vcodec='h264')
  ```

CHAPTER 2

IO

This module offers some methods for data load/dump and file operations.

2.1 Data load/dump

`cvbase` provides a universal api for loading and dumping data, currently supported formats are json, yaml and pickle.

```python
import cvbase as cvb

# load data from a file
data = cvb.load('test.json')
data = cvb.load('test.yaml')
data = cvb.load('test.pickle')
# load data from a file-like object
with open('test.json', 'r') as f:
    data = cvb.load(f)

# dump data to a string
json_str = cvb.dump(data, format='json')
# dump data to a file with a filename (infer format from file extension)
cvb.dump(data, 'out.pickle')
# dump data to a file with a file-like object
with open('test.yaml', 'w') as f:
    data = cvb.dump(data, f, format='yaml')
```

2.2 Load list from a text file

For example a.txt is a text file with 5 lines.

```
a
b
c
d
e
```

Then use `list_from_file` to load the list from a.txt.

```python
import cvbase as cvb

cvb.list_from_file('a.txt')
# output ['a', 'b', 'c', 'd', 'e']
cvb.list_from_file('a.txt', offset=2)
# output ['c', 'd', 'e']
cvb.list_from_file('a.txt', max_num=2)
# output ['a', 'b']
cvb.list_from_file('a.txt', prefix='/mnt/')
# output ['/mnt/a', '/mnt/b', '/mnt/c', '/mnt/d', '/mnt/e']
```

2.3 Load dict from a text file

For example a.txt is a text file with 5 lines.

```
1 cat
2 dog cow
3 panda
```

Then use `dict_from_file` to load the list from a.txt.

```python
import cvbase as cvb

cvb.dict_from_file('a.txt')
# output {'1': 'cat', '2': ['dog', 'cow'], '3': 'panda'}
cvb.dict_from_file('a.txt', key_type=int)
# output {1: 'cat', 2: ['dog', 'cow'], 3: 'panda'}
```

2.4 File/Directory operations

Use `check_file_exist` to check if a file exists, if not, a `FileNotFoundError` or `IOError` will be thrown out.

Use `mkdir_or_exist` to check if a directory exists, the directory will be created if not exists.

Use `scandir` to scan a directory for all files or files will certain suffix.

```python
import cvbase as cvb

# scan the folder "test" for all files
for filename in cvb.scandir('test'):
    print(filename)
# scan the folder "test" for all jpg files
for filename in cvb.scandir('test', '.jpg'):
    print(filename)
```

CHAPTER 3

Image

This module provides some image processing methods.

3.1 Read/Write/Show

To read or write images files, use `read_img` or `write_img`.

```python
import cvbase as cvb

img = cvb.read_img('test.jpg')
img_ = cvb.read_img(img) # nothing will happen, img_ = img
cvb.write_img(img, 'out.jpg')
```

To read images from bytes

```python
import cvbase as cvb

with open('test.jpg', 'rb') as f:
    data = f.read()
img = cvb.img_from_bytes(data)
```

To show an image file or a loaded image

```python
cvb.show_img('tests/data/color.jpg')

for i in range(10):
    img = np.random.randint(256, size=(100, 100, 3), dtype=np.uint8)
    cvb.show_img(img, win_name='test image', wait_time=200)
```

3.2 Resize

There are lots of resize methods. All resize_* methods have a parameter `return_scale`, if this param is `False`, then the return value is merely the resized image, otherwise is a tuple (resized_img, scale).

```
import cvbase as cvb

# resize to a given size
cvb.resize(img, (1000, 600), return_scale=True)
# resize to the same size of another image
cvb.resize_like(img, dst_img, return_scale=False)
# resize by a ratio
cvb.resize_by_ratio(img, 0.5)
# resize so that the max edge no longer than 1000, short edge no longer than 800
# without changing the aspect ratio
cvb.resize_keep_ar(img, 1000, 800)
# resize to the maximum size
cvb.limit_size(img, 400)
```

3.3 Color space conversion

Supported conversion methods:

- bgr2gray
- gray2bgr
- bgr2rgb
- rgb2bgr
- bgr2hsv
- hsv2bgr

```
import cvbase as cvb

img = cvb.read_img('tests/data/color.jpg')
img1 = cvb.bgr2rgb(img)
img2 = cvb.rgb2gray(img1)
img3 = cvb.bgr2hsv(img)
```

3.4 Crop

Support single/multiple crop.

```
import cvbase as cvb
import numpy as np

img = cvb.read_img('tests/data/color.jpg')
bboxes = np.array([10, 10, 100, 120])  # x1, y1, x2, y2
patch = cvb.crop_img(img, bboxes)
bboxes = np.array([[10, 10, 100, 120], [0, 0, 50, 50]])
patches = cvb.crop_img(img, bboxes)
```

Resizing cropped patches.

```
# upsample patches by 1.2x
patches = cvb.crop_img(img, bboxes, scale_ratio=1.2)
```

3.5 Padding

Pad an image to specific size with given values.

```
import cvbase as cvb

img = cvb.read_img('tests/data/color.jpg')
img = cvb.pad_img(img, (1000, 1200), pad_val=0)
img = cvb.pad_img(img, (1000, 1200), pad_val=[100, 50, 200])
```

CHAPTER 4

Video

This module provides friendly apis to read and convert videos.

```python
import cvbase as cvb

video = cvb.VideoReader('test.mp4')
# access basic info
print(len(video))
print(video.width, video.height, video.resolution, video.fps)
# iterate over all frames
for frame in video:
    print(frame.shape)
# read the next frame
img = video.read()
# read a frame by index
img = video[100]
# split a video into frames and save to a folder
video.cvt2frames('out_dir')
# generate video from frames
cvb.frames2video('out_dir', 'test.avi')
```

There are also some methods for editing videos, which wraps the commands of ffmpeg.

```python
import cvbase as cvb

# cut a video clip
cvb.cut_video('test.mp4', 'clip1.mp4', start=3, end=10, vcodec='h264')
# join a list of video clips
cvb.concat_video(['clip1.mp4', 'clip2.mp4'], 'joined.mp4', log_level='quiet')
# resize a video with the specified size
cvb.resize_video('test.mp4', 'resized1.mp4', (360, 240))
# resize a video with a scaling ratio of 2
cvb.resize_video('test.mp4', 'resized2.mp4', ratio=2)
```

CHAPTER 5

API

5.1 io

class cvbase.io.**AsyncDumper**

run()

Method to be run in sub-process; can be overridden in sub-class

cvbase.io.**dump**(*obj*, *file=None*, *format=None*, ***kwargs*)

Dump contents to json/yaml/pickle strings or files.

This method provides a unified api for dumping to files, and also supports custom arguments for each file format.

Parameters

- **file** (*None or str or file-like object*) – if None, then dump to a str, otherwise to a file specified by the filename or file-like object
- **obj** (*any*) – the python object to be dumped
- **format** (*None or str*) – same as `load()`

Returns True for success, False otherwise

Return type bool

cvbase.io.**load**(*file*, *format=None*, ***kwargs*)

Load contents from json/yaml/pickle files, and also supports custom arguments for each file format.

This method provides a unified api for loading from serialized files.

Parameters

- **file** (*str or file-like object*) – filename or the file-like object
- **format** (*None or str*) – if it is None, file format is inferred from the file extension, otherwise use the specified one. Currently supported formats are “json”, “yaml”, “yml”, “pickle” and “pkl”

Returns The content from the file

5.2 conversion

`cvbase.conversion.`**`is_list_of`**(*in_list*, *expected_type*)
Check whether it is a list of objects of a certain type

`cvbase.conversion.`**`list_cast`**(*in_list*, *dst_type*)
Convert a list of items to some type

`cvbase.conversion.`**`merge_list`**(*in_list*)
Merge a list of list into a single list

Parameters **`in_list`** (*`list`*) – the list of list to be merged

Returns the flat list

Return type list

`cvbase.conversion.`**`slice_list`**(*in_list*, *lens*)
Slice a list into several sub lists by a list of given length

Parameters

- **`in_list`** (*`list`*) – the list to be sliced
- **`lens`** (*`int or list`*) – the expected length of each out list

Returns list of sliced list

Return type list

`cvbase.conversion.`**`to_bool`**(*var*)
Convert a variable to bool type

5.3 image

`cvbase.image.`**`bgr2gray`**(*img*, *keepdim=False*)
Convert a BGR image to grayscale image

Parameters

- **`img`** (*`ndarray or str`*) – either an image or path of an image
- **`keepdim`** (*`bool`*) – if set to False(by default), return the gray image with 2 dims, otherwise 3 dims.

Returns the grayscale image

Return type ndarray

`cvbase.image.`**`bgr2hsv`**(*img*)
Convert a BGR image to HSV image

Parameters **`img`** (*`ndarray or str`*) – either an image or path of an image

Returns the HSV image

Return type ndarray

`cvbase.image.`**`bgr2rgb`**(*img*)
Convert a BGR image to RGB image

Parameters **img** (*ndarray or str*) – either an image or path of an image

Returns the RGB image

Return type ndarray

cvbase.image.**crop_img**(*img*, *bboxes*, *scale_ratio=1.0*, *pad_fill=None*)

Crop image patches

3 steps: scale the bboxes -> clip bboxes -> crop and pad

Parameters

- **img** (*ndarray*) – image to be cropped
- **bboxes** (*ndarray*) – shape (k, 4) or (4,), location of cropped bboxes
- **scale_ratio** (*float*) – scale ratio of bboxes, default by 1.0 (no scaling)
- **pad_fill** (*number or list*) – value to be filled for padding, None for no padding

Returns cropped image patches

Return type list or ndarray

cvbase.image.**gray2bgr**(*img*)

Convert a grayscale image to BGR image

Parameters **img** (*ndarray or str*) – either an image or path of an image

Returns the BGR image

Return type ndarray

cvbase.image.**hsv2bgr**(*img*)

Convert a HSV image to BGR image

Parameters **img** (*ndarray or str*) – either an image or path of an image

Returns the BGR image

Return type ndarray

cvbase.image.**img_from_bytes**(*content*, *flag=<sphinx.ext.autodoc.importer._MockObject object>*)

Read an image from bytes

Parameters

- **content** (*bytes*) – images bytes got from files or other streams
- **flag** (*int*) – same as `read_img()`

Returns image array

Return type ndarray

cvbase.image.**limit_size**(*img*, *max_edge*, *return_scale=False*, *interpolation=<sphinx.ext.autodoc.importer._MockObject object>*)

Limit the size of an image

If the long edge of the image is greater than max_edge, resize the image

Parameters

- **img** (*ndarray*) – input image
- **max_edge** (*int*) – max value of long edge
- **return_scale** (*bool*) – whether to return scale besides the resized image

- **interpolation** (*enum*) – interpolation method

Returns (resized image, scale factor)

Return type tuple

`cvbase.image.`**`pad_img`**(*img*, *shape*, *pad_val*)

Pad an image to a certain shape

Parameters

- **img** (*ndarray*) – image to be padded
- **shape** (*tuple*) – expected padding shape
- **pad_val** (*float or int or list*) – values to be filled in padding areas

Returns padded image

Return type ndarray

`cvbase.image.`**`read_img`**(*img_or_path*, *flag=<sphinx.ext.autodoc.importer._MockObject object>*)

Read an image

Parameters

- **img_or_path** (*ndarray or str*) – either an image or path of an image
- **flag** (*int*) – flags specifying the color type of a loaded image

Returns image array

Return type ndarray

`cvbase.image.`**`resize`**(*img*, *size*, *return_scale=False*, *interpolation=<sphinx.ext.autodoc.importer._MockObject object>*)

Resize image by expected size

Parameters

- **img** (*ndarray*) – image or image path
- **size** (*tuple*) – (w, h)
- **return_scale** (*bool*) – whether to return w_scale and h_scale
- **interpolation** (*enum*) – interpolation method

Returns resized image

Return type ndarray

`cvbase.image.`**`resize_by_ratio`**(*img*, *ratio*, *interpolation=<sphinx.ext.autodoc.importer._MockObject object>*)

Resize image by a ratio

Parameters

- **img** (*ndarray*) – image or image path
- **ratio** (*float*) – scale factor
- **interpolation** (*enum*) – interpolation method

Returns resized image

Return type ndarray

cvbase.image.**resize_keep_ar**(*img*, *max_long_edge*, *max_short_edge*, *return_scale=False*, *interpolation=<sphinx.ext.autodoc.importer._MockObject object>*)

Resize image with aspect ratio unchanged

The long edge of resized image is no greater than max_long_edge, the short edge of resized image is no greater than max_short_edge.

Parameters

- **img** (*ndarray*) – image or image path
- **max_long_edge** (*int*) – max value of the long edge of resized image
- **max_short_edge** (*int*) – max value of the short edge of resized image
- **return_scale** (*bool*) – whether to return scale besides the resized image
- **interpolation** (*enum*) – interpolation method

Returns (resized image, scale factor)

Return type tuple

cvbase.image.**resize_like**(*img*, *dst_img*, *return_scale=False*, *interpolation=<sphinx.ext.autodoc.importer._MockObject object>*)

Resize image to the same size of a given image

Parameters

- **img** (*ndarray*) – image or image path
- **dst_img** (*ndarray*) – the given image with expected size
- **return_scale** (*bool*) – whether to return w_scale and h_scale
- **interpolation** (*enum*) – interpolation method

Returns resized image

Return type ndarray

cvbase.image.**rgb2bgr**(*img*)

Convert a RGB image to BGR image

Parameters **img** (*ndarray or str*) – either an image or path of an image

Returns the BGR image

Return type ndarray

cvbase.image.**rotate_img**(*img*, *angle*, *center=None*, *scale=1.0*, *border_value=0*, *auto_bound=False*)

Rotate an image

Parameters

- **img** (*ndarray or str*) – image to be rotated
- **angle** (*float*) – rotation angle in degrees, positive values mean clockwise rotation
- **center** (*tuple*) – center of the rotation in the source image, by default it is the center of the image.
- **scale** (*float*) – isotropic scale factor
- **border_value** (*int*) – border value
- **auto_bound** (*bool*) – whether to adjust the image size to cover the whole rotated image

Returns rotated image

Return type ndarray

`cvbase.image.`**`scale_size`**`(`*size*, *scale*`)`
Scale a size

Parameters

- **`size`** (*`tuple`*) – w, h
- **`scale`** (*`float`*) – scaling factor

Returns scaled size

Return type tuple

`cvbase.image.`**`write_img`**`(`*img*, *file_path*, *params=None*, *auto_mkdir=True*`)`
Write image to file

Parameters

- **`img`** (*`ndarray`*) – image to be written to file
- **`file_path`** (*`str`*) – file path
- **`params`** (*`None or list`*) – same as opencv imwrite interface
- **`auto_mkdir`** (*`bool`*) – if the parrent folder of file_path does not exist, whether to create it automatically

Returns successful or not

Return type bool

5.4 video

`class` `cvbase.video.`**`VideoReader`**`(`*filename*, *cache_capacity=10*`)`
Video class with similar usage to a list object.

This video warpper class provides convenient apis to access frames. There exists an issue of OpenCV's VideoCapture class that jumping to a certain frame may be inaccurate. It is fixed in this class by checking the position after jumping each time.

Cache is used when decoding videos. So if the same frame is visited for the second time, there is no need to decode again if it is stored in the cache.

Example

```
>>> import cvbase as cvb
>>> v = cvb.VideoReader('sample.mp4')
>>> len(v)  # get the total frame number with `len()`
120
>>> for img in v:  # v is iterable
>>>     cvb.show_img(img)
>>> v[5]  # get the 6th frame
```

`current_frame``()`
Get the current frame (frame that is just visited)

Returns if the video is fresh, return None, otherwise return the frame.

Return type ndarray or None

cvt2frames(*frame_dir*, *file_start=0*, *filename_tmpl='{:06d}.jpg'*, *start=0*, *max_num=0*, *show_progress=True*)

Convert a video to frame images

Parameters

- **frame_dir** (*str*) – output directory to store all the frame images
- **file_start** (*int*) – from which filename starts
- **filename_tmpl** (*str*) – filename template, with the index as the variable
- **start** (*int*) – starting frame index
- **max_num** (*int*) – maximum number of frames to be written
- **show_progress** (*bool*) – whether to show a progress bar

fourcc

str – "four character code" of the video

fps

int – fps of the video

frame_cnt

int – total frames of the video

get_frame(*frame_id*)

Get frame by frame id

Parameters **frame_id** (*int*) – id of the expected frame, 1-based index

Returns return the frame if successful, otherwise None.

Return type ndarray or None

height

int – height of video frames

opened

bool – indicate whether the video is opened

position

int – current cursor position, indicating which frame

read()

Read the next frame

If the next frame have been decoded before and in the cache, then return it directly, otherwise decode and return it, put it in the cache.

Returns return the frame if successful, otherwise None.

Return type ndarray or None

resolution

tuple – video resolution (width, height)

vcap

`cv2.VideoCapture` – raw VideoCapture object

width

int – width of video frames

cvbase.video.**check_ffmpeg**(*func*)

A decorator to check if ffmpeg is installed cvbase.video.**concat_video**(**args*, ***kwargs*)

Concatenate multiple videos into a single one

Parameters

- **video_list** (*list*) – a list of video filenames
- **out_file** (*str*) – output video filename
- **vcodec** (*None or str*) – output video codec, None for unchanged
- **acodec** (*None or str*) – output audio codec, None for unchanged
- **log_level** (*str*) – log level of ffmpeg

cvbase.video.**convert_video**(**args*, ***kwargs*)

Convert a video with ffmpeg

This provides a general api to ffmpeg, the executed command is:

```
ffmpeg -y <pre_options> -i <in_file> <options> <out_file>
```

Options(kwargs) are mapped to ffmpeg commands by the following rules:

- key=val: “-key val”
- key=True: “-key”
- key=False: “”

Parameters

- **in_file** (*str*) – input video filename
- **out_file** (*str*) – output video filename
- **pre_options** (*str*) – options appears before “-i <in_file>”

cvbase.video.**cut_video**(**args*, ***kwargs*)

Cut a clip from a video

Parameters

- **in_file** (*str*) – input video filename
- **out_file** (*str*) – output video filename
- **start** (*None or float*) – start time (in seconds)
- **end** (*None or float*) – end time (in seconds)
- **vcodec** (*None or str*) – output video codec, None for unchanged
- **acodec** (*None or str*) – output audio codec, None for unchanged
- **log_level** (*str*) – log level of ffmpeg

cvbase.video.**frames2video**(*frame_dir*, *video_file*, *fps=30*, *fourcc='XVID'*, *filename_tmpl='{:06d}.jpg'*, *start=0*, *end=0*, *show_progress=True*)

Read the frame images from a directory and join them as a video

Parameters

- **frame_dir** (*str*) – frame directory
- **video_file** (*str*) – output video filename
- **fps** (*int*) – fps of the output video

- **fourcc** (*str*) – foutcc of the output video, this should be compatible with the output file type
- **filename_tmpl** (*str*) – filename template, with the index as the variable
- **start** (*int*) – starting frame index
- **end** (*int*) – ending frame index
- **show_progress** (*bool*) – whether to show a progress bar

cvbase.video.**resize_video**(**args*, ***kwargs*)

Resize a video

Parameters

- **in_file** (*str*) – input video filename
- **out_file** (*str*) – output video filename
- **size** (*tuple*) – expected (w, h), eg, (320, 240) or (320, -1)
- **ratio** (*tuple or float*) – expected resize ratio, (2, 0.5) means (w*2, h*0.5)
- **keep_ar** (*bool*) – whether to keep original aspect ratio
- **log_level** (*str*) – log level of ffmpeg

5.5 timer

class cvbase.timer.**Timer**(*start=True*, *print_tmpl=None*)

A flexible Timer class.

Example

```
>>> import time
>>> import cvbase as cvb
>>> with cvb.Timer():
>>>     # simulate a code block that will run for 1s
>>>     time.sleep(1)
1.000
>>> with cvb.Timer(print_tmpl='hey it taks {:.1f} seconds'):
>>>     # simulate a code block that will run for 1s
>>>     time.sleep(1)
hey it taks 1.0 seconds
>>> timer = cvb.Timer()
>>> time.sleep(0.5)
>>> print(timer.since_start())
0.500
>>> time.sleep(0.5)
>>> print(timer.since_last_check())
0.500
>>> print(timer.since_start())
1.000
```

is_running

bool – indicate whether the timer is running

since_last_check()

Time since the last checking.

Either *since_start()* or *since_last_check()* is a checking operation.

Returns(float): the time in seconds

since_start()

Total time since the timer is started.

Returns(float): the time in seconds

start()

Start the timer.

exception cvbase.timer.**TimerError**(*message*)

cvbase.timer.**check_time**(*timer_id*)

Add check points in a single line

This method is suitable for running a task on a list of items. A timer will be registered when the method is called for the first time.

Example

```
>>> import time
>>> import cvbase as cvb
>>> for i in range(1, 6):
>>>     # simulate a code block
>>>     time.sleep(i)
>>>     cvb.check_time('task1')
2.000
3.000
4.000
5.000
```

Parameters **timer_id** (*str*) – timer identifier

5.6 progress

class cvbase.progress.**ProgressBar**(*task_num=0*, *bar_width=50*, *start=True*)

A progress bar which can print the progress

cvbase.progress.**track_parallel_progress**(*func*, *tasks*, *process_num*, *initializer=None*, *initargs=None*, *bar_width=50*, *chunksize=1*, *skip_first=False*, *keep_order=True*)

Track the progress of parallel task execution with a progress bar

The built-in `multiprocessing` module is used for process pools and tasks are done with `Pool.map()` or `Pool.imap_unordered()`.

Parameters

- **func** (*callable*) – the function to be applied to each task
- **tasks** (*tuple of 2 or list*) – a list of tasks
- **process_num** (*int*) – the process(worker) number
- **initializer** (*None or callable*) – see `multiprocessing.Pool` for details
- **initargs** (*None or tuple*) – see `multiprocessing.Pool` for details
- **chunksize** (*int*) – see `multiprocessing.Pool` for details
- **bar_width** (*int*) – width of progress bar

- **skip_first** (*bool*) – whether to skip the first sample when calculating fps
- **keep_order** (*bool*) – if True, `Pool.imap()` is used, otherwise `Pool.imap_unordered()` is used

Returns the results

Return type list

`cvbase.progress.`**`track_progress`**(*func*, *tasks*, *bar_width=50*, ***kwargs*)

Track the progress of tasks execution with a progress bar

Tasks are done with a simple for-loop.

Parameters

- **func** (*callable*) – the function to be applied to each task
- **tasks** (*tuple of 2 or list*) – a list of tasks
- **bar_width** (*int*) – width of progress bar

Returns the results

Return type list

5.7 visualize

class `cvbase.visualize.`**`Color`**(**args*, ***kwargs*)

Color associated with RGB values

8 colors in total: red, green, blue, cyan, yellow, magenta, white and black.

`cvbase.visualize.`**`draw_bboxes`**(*img*, *bboxes*, *colors=(0, 255, 0)*, *top_k=0*, *thickness=1*, *show=True*, *win_name=”*, *wait_time=0*, *out_file=None*)

Draw bboxes on an image

Parameters

- **img** (*str or ndarray*) – the image to be shown
- **bboxes** (*list or ndarray*) – a list of ndarray of shape (k, 4)
- **colors** (*list or Color or tuple*) – a list of colors, corresponding to bboxes
- **top_k** (*int*) – draw top_k bboxes only if positive
- **thickness** (*int*) – thickness of lines
- **show** (*bool*) – whether to show the image
- **win_name** (*str*) – the window name
- **wait_time** (*int*) – value of waitKey param
- **out_file** (*str or None*) – the filename to write the image

`cvbase.visualize.`**`draw_bboxes_with_label`**(*img*, *bboxes*, *labels*, *top_k=0*, *bbox_color=(0, 255, 0)*, *text_color=(0, 255, 0)*, *thickness=1*, *font_scale=0.5*, *show=True*, *win_name=”*, *wait_time=0*, *out_file=None*)

Draw bboxes with label text in image

Parameters

- **img** (*str or ndarray*) – the image to be shown

- **bboxes** (*list or ndarray*) – a list of ndarray of shape (k, 4)
- **labels** (*str or list*) – label name file or list of label names
- **top_k** (*int*) – draw top_k bboxes only if positive
- **bbox_color** (Color *or tuple*) – color to draw bboxes
- **text_color** (Color *or tuple*) – color to draw label texts
- **thickness** (*int*) – thickness of bbox lines
- **font_scale** (*float*) – font scales
- **show** (*bool*) – whether to show the image
- **win_name** (*str*) – the window name
- **wait_time** (*int*) – value of waitKey param
- **out_file** (*str or None*) – the filename to write the image

cvbase.visualize.**show_img**(*img*, *win_name=”*, *wait_time=0*)

Show an image

Parameters

- **img** (*str or ndarray*) – the image to be shown
- **win_name** (*str*) – the window name
- **wait_time** (*int*) – value of waitKey param

5.8 det

cvbase.det.bbox_ops.**bbox2roi**(*bbox_list*, *stack=True*)

Convert bboxes to rois by adding index at the first col.

Parameters

- **bbox_list** (*list*) – a list of ndarray (k_i, 4)
- **stack** (*bool*) – whether to stack all the rois

Returns rois of shape (sum_k, 4)

Return type ndarray or list

cvbase.det.bbox_ops.**bbox_clip**(*bboxes*, *img_shape*)

Limit bboxes to fit the image size

Parameters

- **bboxes** (*ndarray*) – shape (. . . , 4*k)
- **img_shape** (*tuple*) – (height, width)

cvbase.det.bbox_ops.**bbox_denormalize**(*deltas, means=[0, 0, 0, 0], stds=[1, 1, 1, 1]*)

Denormalize bbox deltas

Parameters

- **deltas** (*ndarray*) – shape(. . . , 4*k)
- **means** (*ndarray or list*) – shape(4,) or (4*k,)
- **stds** (*ndarray or list*) – shape(4,) or (4*k,)

Returns denormalized deltas, same shape as input deltas

Return type ndarray

`cvbase.det.bbox_ops.`**`bbox_flip`**(*bboxes*, *img_shape*)

Flip bboxes horizontally

Parameters

- **`bboxes`** (*`ndarray`*) – shape (…, 4*k)
- **`img_shape`** (*`tuple`*) – (height, width)

`cvbase.det.bbox_ops.`**`bbox_normalize`**(*deltas, means=[0, 0, 0, 0], stds=[1, 1, 1, 1]*)

Normalize bbox deltas

Parameters

- **`deltas`** (*`ndarray`*) – shape(…, 4*k)
- **`means`** (*`ndarray or list`*) – shape(4,) or (4*k,)
- **`stds`** (*`ndarray or list`*) – shape(4,) or (4*k,)

Returns normalized deltas, same shape as input deltas

Return type ndarray

`cvbase.det.bbox_ops.`**`bbox_overlaps`**(*bboxes1*, *bboxes2*, *mode='iou'*)

Calculate the ious between each bbox of bboxes1 and bboxes2

Parameters

- **`bboxes1`** (*`ndarray`*) – shape (n, 4)
- **`bboxes2`** (*`ndarray`*) – shape (k, 4)
- **`mode`** (*`str`*) – iou (intersection over union) or iof (intersection over foreground)

Returns shape (n, k)

Return type ious(ndarray)

`cvbase.det.bbox_ops.`**`bbox_perturb`**(*bbox*, *offset_ratio*, *num*, *clip_shape=None*, *min_iou=None*, *max_iou=None*, *max_try=20*)

Perturb a bbox around it to generate more bboxes

Parameters

- **`bbox`** (*`ndarray`*) – shape(4,)
- **`offset_ratio`** (*`float`*) – max offset ratio (w.r.t the bbox w and h)
- **`num`** (*`int`*) – number of bboxes to be generated
- **`clip_shape`** (*`None or tuple`*) – (h, w)
- **`min_iou`** (*`float`*) – minimum iou of perturbed bboxes with original bbox
- **`max_iou`** (*`float`*) – maximum iou of perturbed bboxes with original bbox

Returns perturbed bboxes of shape (num, 4)

Return type ndarray

`cvbase.det.bbox_ops.`**`bbox_scaling`**(*bboxes*, *scale*, *clip_shape=None*)

Scaling bboxes and clip the boundary(optional)

Parameters

- **bboxes** (*ndarray*) – shape(…, 4)
- **scale** (*float*) – scaling factor
- **clip** (*None or tuple*) – (h, w)

Returns scaled bboxes

Return type ndarray

`cvbase.det.bbox_ops.`**`bbox_transform`**(*proposals, gt, means=[0, 0, 0, 0], stds=[1, 1, 1, 1]*)

Calculate regression deltas from proposals and ground truths

dx = (gx - px) / pw, dw = log(gw / pw)

Parameters

- **proposals** (*ndarray*) – shape (…, 4)
- **gt** (*ndarray*) – shape (…, 4) or (1.., 4)

Returns same shape as proposals

Return type ndarray

`cvbase.det.bbox_ops.`**`bbox_transform_inv`**(*bboxes, deltas, means=[0, 0, 0, 0], stds=[1, 1, 1, 1]*)

Get ground truth bboxes from input bboxes and deltas

gw = pw * exp(dw), gx = px + dx * pw

Parameters

- **bboxes** (*ndarray*) – shape (…, 4) [x1, y1, x2, y2]
- **deltas** (*ndarray*) – shape (…, 4*k) [dx, dy, dw, dh]

Returns same shape as input deltas

Return type ndarray

`cvbase.det.bbox_ops.`**`flat2list`**(*bbox_flat, num_classes=30*)

Convert a flat array to a list of array

Parameters

- **bbox_flat** (*list*) – shape (sum_n, k+1)
- **num_classes** (*int*) – num of classes, length of list

Returns a list of (n, k) arrays

Return type list

`cvbase.det.bbox_ops.`**`list2flat`**(*bbox_list*)

Convert a list of bboxes to a numpy array with one col added

Parameters **bbox_list** (*list*) – a list of (n, k) arrays

Returns shape (sum_n, k+1)

Return type ndarray

`cvbase.det.eval.`**`average_precision`**(*recalls, precisions, mode='area'*)

Calculate average precision (for single or multiple scales)

Parameters

- **recalls** (*ndarray*) – shape (num_scales, num_dets) or (num_dets,)
- **precisions** (*ndarray*) – shape (num_scales, num_dets) or (num_dets,)

- **mode** (*str*) – 'area' or '11points', 'area' means calculating the area under precision-recall curve, '11points' means calculating the average precision of recalls at [0, 0.1, . . . , 1]

Returns calculated average precision

Return type float or ndarray

`cvbase.det.eval.`**`bbox_recalls`**(*gts*, *proposals*, *proposal_nums=None*, *iou_thrs=None*, *print_summary=True*)

Calculate recalls

Parameters

- **gts** (*list or ndarray*) – a list of arrays of shape (n, 4)
- **proposals** (*list or ndarray*) – a list of arrays of shape (k, 4) or (k, 5)
- **proposal_nums** (*int or list of int or ndarray*) – top N proposals
- **thrs** (*float or list or ndarray*) – iou thresholds

Returns recalls of different ious and proposal nums

Return type ndarray

`cvbase.det.eval.`**`eval_map`**(*det_results*, *gt_bboxes*, *gt_labels*, *gt_ignore=None*, *scale_ranges=None*, *iou_thr=0.5*, *dataset=None*, *print_summary=True*)

Evaluate mAP of a dataset

Parameters

- **det_results** (*list*) – a list of list, [[cls1_det, cls2_det, . . .], . . .]
- **gt_bboxes** (*list*) – ground truth bboxes of each image, a list of K*4 array
- **gt_labels** (*list*) – ground truth labels of each image, a list of K array
- **gt_ignore** (*list*) – gt ignore indicators of each image, a list of K array
- **scale_ranges** (*list or None*) – a list of tuples, [(min1, max1), (min2, max2), . . .]
- **iou_thr** (*float*) – IoU threshold
- **dataset** (*None or str*) – dataset name, there are minor differences in metrics for different datsets, e.g. "voc07", "voc12", "det", "vid"
- **print_summary** (*bool*) – whether to print the mAP summary

Returns (mAP, [dict, dict, . . .])

Return type tuple

`cvbase.det.eval.`**`get_cls_results`**(*det_results*, *gt_bboxes*, *gt_labels*, *gt_ignore*, *class_id*)

Get det results and gt information of a certain class.

`cvbase.det.eval.`**`plot_iou_recall`**(**args*, ***kwargs*)

Plot IoU-Recalls curve

Parameters

- **recalls** (*ndarray or list*) – shape (k,)
- **iou_thrs** (*ndarray or list*) – same shape as *recalls*

`cvbase.det.eval.`**`plot_num_recall`**(**args*, ***kwargs*)

Plot Proposal_num-Recalls curve

Parameters

- **recalls** (*ndarray or list*) – shape (k,)
- **proposal_nums** (*ndarray or list*) – same shape as *recalls*

cvbase.det.eval.**print_map_summary**(*mean_ap*, *results*, *dataset=None*)

Print mAP and results of each class

Parameters

- **mean_ap** (*float*) – calculated from *eval_map*
- **results** (*list*) – calculated from *eval_map*
- **dataset** (*None or str or list*) – get label names by dataset, see *cvbase.read_labels()*

cvbase.det.eval.**print_recall_summary**(*recalls*, *proposal_nums*, *iou_thrs*, *row_idxs=None*, *col_idxs=None*)

Print recalls in a table

Parameters

- **recalls** (*ndarray*) – calculated from *bbox_recalls*
- **proposal_nums** (*ndarray or list*) – top N proposals
- **iou_thrs** (*ndarray or list*) – iou thresholds
- **row_idxs** (*ndarray*) – which rows(proposal nums) to print
- **col_idxs** (*ndarray*) – which cols(iou thresholds) to print

cvbase.det.eval.**set_recall_param**(*proposal_nums*, *iou_thrs*)

Check proposal_nums and iou_thrs and set correct format

cvbase.det.eval.**tpfp_default**(*det_bboxes*, *gt_bboxes*, *gt_ignore*, *iou_thr*, *area_ranges=None*)

Check if detected bboxes are true positive or false positive.

Parameters

- **det_bbox** (*ndarray*) – the detected bbox
- **gt_bboxes** (*ndarray*) – ground truth bboxes of this image
- **gt_ignore** (*ndarray*) – indicate if gts are ignored for evaluation or not
- **iou_thr** (*float*) – the iou thresholds

Returns (tp, fp), two arrays whose elements are 0 and 1

Return type tuple

cvbase.det.eval.**tpfp_imagenet**(*det_bboxes*, *gt_bboxes*, *gt_ignore*, *default_iou_thr*, *area_ranges=None*)

Check if detected bboxes are true positive or false positive.

Parameters

- **det_bbox** (*ndarray*) – the detected bbox
- **gt_bboxes** (*ndarray*) – ground truth bboxes of this image
- **gt_ignore** (*ndarray*) – indicate if gts are ignored for evaluation or not
- **default_iou_thr** (*float*) – the iou thresholds for medium and large bboxes
- **area_ranges** (*list or None*) – gt bbox area ranges

Returns two arrays (tp, fp) whose elements are 0 and 1

Return type tuple

`cvbase.det.labels.`**`read_labels`**(*dataset_or_file*)

Read labels from file or list

5.9 optflow

`cvbase.optflow.io.`**`dequantize_flow`**(*dx*, *dy*, *max_val=0.02*, *denorm=True*)

Recover flow from quantized flow

Parameters

- **dx** (*ndarray*) – quantized dx
- **dy** (*ndarray*) – quantized dy
- **max_val** (*float*) – maximum value used when quantizing.
- **denorm** (*bool*) – whether to multiply flow values with width/height

Returns dequantized dx and dy

Return type tuple

`cvbase.optflow.io.`**`quantize_flow`**(*flow*, *max_val=0.02*, *norm=True*)

Quantize flow to [0, 255] (much smaller size when dumping as images)

Parameters

- **flow** (*ndarray*) – optical flow
- **max_val** (*float*) – maximum value of flow, values beyond [-max_val, max_val] will be truncated.
- **norm** (*bool*) – whether to divide flow values by width/height

Returns quantized dx and dy

Return type tuple

`cvbase.optflow.io.`**`read_flow`**(*flow_or_path*, *quantize=False*, **args*, ***kwargs*)

Read an optical flow map

Parameters

- **flow_or_path** (*ndarray or str*) – either a flow map or path of a flow
- **quantize** (*bool*) – whether to read quantized pair, if set to True, remaining args will be passed to *dequantize_flow()*

Returns optical flow

Return type ndarray

`cvbase.optflow.io.`**`write_flow`**(*flow*, *filename*, *quantize=False*, **args*, ***kwargs*)

Write optical flow to file

Parameters

- **flow** (*ndarray*) – optical flow
- **filename** (*str*) – file path
- **quantize** (*bool*) – whether to quantize the flow and save as 2 images, if set to True, remaining args will be passed to *quantize_flow()*

`cvbase.optflow.visualize.`**`flow2rgb`**(*flow*, *color_wheel=None*, *unknown_thr=1000000.0*)

Convert flow map to RGB image

Parameters

- **`flow`** (*`ndarray`*) – optical flow
- **`color_wheel`** (*`ndarray or None`*) – color wheel used to map flow field to RGB colorspace. Default color wheel will be used if not specified
- **`unknown_thr`** (*`str`*) – values above this threshold will be marked as unknown and thus ignored

Returns an RGB image that can be visualized

Return type ndarray

`cvbase.optflow.visualize.`**`make_color_wheel`**(*bins=None*)

Build a color wheel

Parameters **`bins`** (*`list or tuple, optional`*) – specify number of bins for each color range, corresponding to six ranges: red -> yellow, yellow -> green, green -> cyan, cyan -> blue, blue -> magenta, magenta -> red. [15, 6, 4, 11, 13, 6] is used for default (see Middlebury).

Returns color wheel of shape (total_bins, 3)

Return type ndarray

`cvbase.optflow.visualize.`**`show_flow`**(**args*, ***kwargs*)

Show optical flow

Parameters

- **`flow`** (*`ndarray or str`*) – optical flow to be shown
- **`win_name`** (*`str`*) – the window name
- **`wait_time`** (*`int`*) – value of waitKey param

CHAPTER 6

Indices and tables

- genindex
- search

Python Module Index

Index