svkcore Documentation

Release v0.0.3

jianghuazhao

Mar 26, 2023

CONTENTS:

CHAPTER

ONE

GET START

1.1 install

1.1.1 pip

```
pip install svkcore
```

1.1.2 from source

```
git clone https://github.com/ZhaoJianghua/simple-vision-kit.git
cd simple-vision-kit
python setup.py install
```

1.2 usage

1.2.1 common

- io: load and save data more convenient

```
from svkcore import common

# save & load json
data = {'data': 1, "message": "hello"}
common.save_json(data, 'user_data.json')
data = common.load_json('user_data.json')

# save & load pickle
data = {'data': np.array([1]), "message": "hello"}
common.save_pickle(data, 'user_data.pkl')
data = common.load_pickle(data, 'user_data.pkl')

# read & write image platform compatile with win system
img = common.cv2imread("path")
common.cv2imwrite("path", img)
```

- collect data: collect from a directory

```python
from svkcore import common

# collect images/annotations
img_ps = common.find_file_recursive(directory="root", suffixes=[".jpg"], ignore_
↪case=True)
print(img_ps[:1])
# get basename head
hd = common.bsn_head("root/xxx/xxxx.jpg")
print(hd)
# collect examples
examples = common.collect_examples(directory="dir", suffixes_list=[[".jpg"], [".xml"]])
print(examples[:1])
# or use collect_pascal_data to collect pascal format data
examples = common.collect_pascal_data(directory="dir")
print(examples[:1])
```

- numpy operations: some useful numpy operations

```python
indexes = common.ndarray_index(shape=[7, 8])
distances = common.points_distance(points0=[[0, 0]], points1=[[1, 1]])
grids = common.generate_grid(panel_size=[1024, 1024], grid_size=(256, 256),
                             overlap_size=(24, 24))
# common.seg2point(seg, max_diameter: int, min_distance, fb_threshold: float = 0.5,
#                  min_fore_count: int = 1, max_fore_count: int = -1,
#                  avg_fore_score: float = 0.55, distance_weights=(1., 1.))
# common.seg2line(seg, fb_threshold=0.5, smooth_width=3, partition_width=20,
#                 partition_height=30)
```

1.2.2 annotation

Pascal and coco annotation load, save and convert.

```python
from svkcore import annotation

# load pascal annotation
ann = annotation.DTAnnotation.load("pascal_annotation_file")
for obj in ann:
    print(obj.name)
    print(obj.bndbox)
    print(obj.difficult)
    obj.name = obj.name + "-new"
ann.dump("pascal_annotation_file_new")

# convert pascal data to coco format
dataset = annotation.DTDataset.load_pascal(annotation_paths=[],
                                           image_paths=[])
dataset.dump_coco("coco_format_dataset_file")
```

1.2.3 shapes

Base shapes which usually be used in image tasks.

```
from svkcore import shapes

# shapes and operations
point0 = shapes.Point([0, 0])
point1 = shapes.Point([1, 1])
points = shapes.Points([point0, point1])
bndbox0 = points.bounding_box()
bndbox1 = shapes.Box([0, 0, 1, 1])
bsize = bndbox1.bsize()
center = bndbox1.center()
polygon = bndbox1.to_polygon()
mask = bndbox1.to_mask()
bndbox2 = polygon.bounding_box()
area = polygon.area()
```

1.2.4 visualize

Visualize part for visualize common shapes.

```
from PIL import Image
from svkcore import visualize
from svkcore import annotation

pil_image = Image.new("RGB", [600, 600])
boxes = [[100, 200, 400, 300]]
visualize.draw_boxes(pil_image, boxes=boxes)
visualize.draw_texts(pil_image, xys=[(100, 200)], texts=["box"])
visualize.draw_points(pil_image, points=[[50, 50]])
visualize.draw_lines(pil_image, lines=[[0, 0], [300, 300]])

ann = annotation.DTAnnotation.load("path")
visualize.draw_annotation(pil_image, ann, name2cls={},
                          add_unknown_name=True)
```

CHAPTER

TWO

SVKCORE PACKAGE

2.1 svkcore.annotation

2.1.1 svkcore.annotation.pascal

class svkcore.annotation.pascal.**DTAnnotation**(*filename: str*, *size*, *objects: List[*DTObject*]*, *segmented=False*, ***kwargs*)

Bases: object

Detection Annotation: An annotation for object detection

dump(*path*)

Dump DTAnnotation to a file

Parameters
path (str) – dumped file path

dumpd()

Dump DTAnnotation to a dict

Returns
a dict contains DTAnnotation information

Return type
dict

static load(*path: str*)

Load DTAnnotation from a file

Parameters
path – file path

Returns
loaded DTAnnotation object

Return type
DTAnnotation

static loadd(*obj: dict*)

Load DTAnnotation from a dict

Parameters
obj – a dict contains DTAnnotation information

Returns
loaded DTAnnotation object

Return type
DTAnnotation

`size_keys = ('width', 'height', 'depth')`

class svkcore.annotation.pascal.**DTDataset**(*annotations*, *images*, *categories*)

Bases: object

Detection dataset: A collection of annotations for object detection

dump_coco(*path*)

Save dataset to coco format

Parameters
path – coco format annotation path

Returns
None

dump_pascal(*annotation_dir*)

Save dataset to pascal format

Parameters
annotation_dir – pascal format annotations directory

Returns
None

dump_yolo(*dataset_dir*)

Save dataset to yolo format

Parameters
dataset_dir – yolo format dataset directory

Returns
None

static load_coco(*annotation_path*, *image_root*)

Load coco format dataset

Parameters

- **annotation_path** – json format annotation file path
- **image_root** – image root directory

Returns
DTDataset object

static load_pascal(*annotation_paths*, *image_paths*)

Load pascal format dataset

Parameters

- **annotation_paths** – a list of pascal format annotation file path
- **image_paths** – a list of image path respect with each annotation file

Returns
DTDataset object

class svkcore.annotation.pascal.**DTObject**(*name*, *bndbox=None*, *polygon=None*, *mask=None*, *pose='Unspecified'*, *truncated=False*, *difficult=False*)

Bases: object

Detection object: base object for object detection

box_keys = ('xmin', 'ymin', 'xmax', 'ymax')

dumpd() → dict

Dump DTObject to a dict

Returns
a dict contains DTObject information

Return type
dict

static loadd(*obj: dict*)

Load DTObject from a dict

Parameters
obj – a dict contains DTObject information

Returns
loaded DTObject object

Return type
DTObject

svkcore.annotation.pascal.**read_annotation**(*annotation_path*)

Read object detection annotation of xml format file

Parameters
annotation_path (str) – file path of annotation

Returns
a dict of detection annotation

Return type
dict

svkcore.annotation.pascal.**write_annotation**(*annotation_path*, *annotation*)

Write object detection annotation to a xml format file

Parameters

- **annotation_path** – file path of annotation
- **annotation** – a dict of detection annotation

2.2 svkcore.common

2.2.1 svkcore.common.common

This module provides common utility functions for simple vision kit. It includes functions for computing md5 hash, aligning file paths, grouping lists, finding files recursively, collecting examples, and encoding/decoding images using base64.

svkcore.common.common.**align_paths**(*paths0*, *paths1*, **args*, *sort=False*, *key_fn=None*)

Align paths base on its name head This function will delete dis-matched paths with no prompt.

Parameters

- **paths0** (*list*) – the first list of file paths
- **paths1** (*list*) – the second list of file paths
- **args** – the others list of file paths
- **sort** (*bool*) – sorted output by key
- **key_fn** (*callable*) – extract align key function, default is *basename_head()*

Returns
align paths in list

Return type
list

svkcore.common.common.**b64decode_image**(*data: bytes*) → Image

Decode bytes data of encoded image to an instance of PIL.Image.Image

Parameters
data (*bytes*) – base64 encoded bytes image data

Returns
An instance of Image.Image represents the decode image

Return type
PIL.Image.Image

svkcore.common.common.**b64encode_image**(*image: Image*, *format: str = 'JPEG'*) → bytes

Convert PIL.Image.Image object to bytes data use base64 encode

Parameters

- **image** (*PIL.Image.Image*) – an instance of PIL.Image.Image
- **format** (*str*) – a string represents image encoding format. could be "JPEG" or "PNG"

Returns
base64 encoded image data

Return type
bytes

svkcore.common.common.**basename_head**(*path*, *sep='.'*, *align_left=False*)

Get basename head of a path

Parameters

- **path** (*str*) – a path
- **sep** (*str*) – a separator str to split path's basename. Default is ".".
- **align_left** (*bool*) – split basename with separator from left or not. Default is False. And this function will split basename from right.

Returns
the head part of basename of `path`

Return type
str

svkcore.common.common.**bsn_head**(*path*, *sep='.'*, *align_left=False*)

Get basename head of a path

Parameters

- **path** (*str*) – a path
- **sep** (*str*) – a separator str to split path's basename. Default is ".".
- **align_left** (*bool*) – split basename with separator from left or not. Default is False. And this function will split basename from right.

Returns
the head part of basename of path

Return type
str

svkcore.common.common.**collect_examples**(*directory*, *suffixes_list*, *ignore_case=False*, *sort=False*, *key_fn=None*)

Collect examples from one directory base on given suffixes list

Parameters

- **directory** (*str*) – Root directory of examples
- **suffixes_list** (*list*) – A list of suffix list for example's each part
- **ignore_case** (*bool*) – ignore case when match file name suffix. Default is False.
- **sort** (*bool*) – sort the result list by key. Default is False.
- **key_fn** (*callable*) – a function use to extract align key, default is *basename_head()*

Returns
all matched file paths list

Return type
list

svkcore.common.common.**collect_pascal_data**(*directory*)

Collect pascal format dataset

Parameters
directory (*str*) – the root directory of a pascal dataset

Returns
two list of images and annotations of the the pascal dataset

Return type
list

svkcore.common.common.**find_file_recursive**(*directory*, *suffixes*, *ignore_case=False*)

Find all files with provided suffixes

Parameters

- **directory** (*str*) – the target directory
- **suffixes** (*Union[str, list]*) – a suffix or a list of suffixes of file to be find
- **ignore_case** (*bool*) – match file suffix in case ignore mode

Returns
a list which returns the path to the file which meets the postfix

Return type
list

svkcore.common.common.**find_files**(*directory*, *suffixes*, *ignore_case=False*)

Find all files with provided suffixes

Parameters

- **directory** (*str*) – the target directory
- **suffixes** (*Union[str, list]*) – a suffix or a list of suffixes of file to be find
- **ignore_case** (*bool*) – match file suffix in case ignore mode

Returns
a list which returns the path to the file which meets the postfix

Return type
list

svkcore.common.common.**get_default_font**(*size: int = 24*) → ImageFont

Get a default PIL.ImageFont.ImageFont instance for show label name on image which could deal with both English and Chinese

Parameters
size (*int*) – font size

Returns
a font object

Return type
PIL.ImageFont.ImageFont

svkcore.common.common.**group**(*lst*, *key*, *value=None*) → Dict[object, list]

Group list by key and return a dict. Each value of the result dict is a list. And each list contains all values with same key.

Parameters

- **lst** (*list*) – a list of objects to be group
- **key** (*Union[list, callable]*) – a list of key objects or a callable function map each item in lst to its key
- **value** (*Union[list, callable]*) – a list of value objects or a callable function map each item in lst to its value. Default is *None* and group will use *lst* as *value*.

Returns
grouped result

Return type
Dict[object, list]

svkcore.common.common.**group_map**(*_group: dict*, *func: Callable*, *with_key: bool = False*)

Do map on a group result

Parameters

- **_group** (*Dict[object, list]*) – a result dict of `group`
- **func** (*Callable*) – a function used to process value or key and value
- **with_key** (*bool*) – `func` process with key and value or only value. Default is *False*, only process value.

Returns
map result dict

Return type
dict

svkcore.common.common.**ifind_file_recursive**(*directory*, *suffixes*, *ignore_case=False*)

Find all files with provided suffixes

Parameters

- **directory** (*str*) – the target directory
- **suffixes** (*Union[str, list]*) – a suffix or a list of suffixes of file to be find
- **ignore_case** (*bool*) – match file suffix in case ignore mode

Returns
a generator which returns the path to the file which meets the postfix

Return type
a generator

svkcore.common.common.**image_md5**(*image: Image*) → str

Compute a image md5 value

Parameters
image (*PIL.Image.Image*) – pillow image object to be hashed by md5

Returns
md5 hash string

Return type
str

svkcore.common.common.**str_md5**(*_bytes: bytes*) → str

Compute string md5 value

Parameters
_bytes (*bytes*) – data in bytes format to be hashed by md5

Returns
md5 hash string

Return type
str

2.2.2 svkcore.common.fileio

This module provides functions for file input/output operations, including copying files, loading and saving json, pickle, csv files, and reading and writing images using OpenCV.

svkcore.common.fileio.**copy_files**(*paths*, *dst_dir*, *src_dir=None*)

Copy files in list `paths` from `src_dir` to `dst_dir`

Parameters

- **paths** (*list*) – a list of paths
- **dst_dir** (*str*) – destination directory
- **src_dir** (*str*) – root directory of all paths

svkcore.common.fileio.**cv2imread**(*path*)

Read image from file path using OpenCV

Parameters
path (*str*) – image file path

Returns
loaded image as numpy array

Return type
numpy.ndarray

svkcore.common.fileio.**cv2imwrite**(*path*, *img*)

This function is a compatible version of *cv2.imwrite*. It writes an image to the specified file path in .jpg format.

Parameters

- **path** (*str*) – output image file path
- **img** (*numpy.ndarray*) – output image data

svkcore.common.fileio.**load_csv**(*path*, *, *with_header=True*, *encoding=None*)

This function is used to load csv format file. It reads the csv file from the given path and returns the rows as a list of lists. If with_header is True, the first row is considered as the header and returned separately. If encoding is provided, it is used to decode the csv file.

Parameters

- **path** (*str*) – csv file path
- **with_header** (*bool*) – return result with data header
- **encoding** (*str*) – csv file encoding like `utf-8`, etc.

Returns

Return type
list

svkcore.common.fileio.**load_json**(*path*, *, *encoding=None*, ***kwargs*)

Load json format file

Parameters

- **path** (*str*) – json file path
- **encoding** (*str*) – json file encoding like `utf-8`, etc.
- **kwargs** – other options for *json.load*

Returns
loaded json object

Type
object

svkcore.common.fileio.**load_pickle**(*path*)

Load object from pickle format

Parameters
path (*str*) – pickle file path

Returns
loaded object

Return type
object

svkcore.common.fileio.**save_csv**(*rows*, *path*, *, *header=None*, *encoding=None*)

This function is used to save data into csv format file. If header is provided, header will place at the first row of the output csv file. If encoding is provided, it is used to decode the csv file.

Parameters

- **rows** (*list*) – a list of data row
- **path** (*str*) – csv file path
- **header** (*list*) – data header
- **encoding** (*str*) – csv file encoding like `utf-8`, etc.

svkcore.common.fileio.**save_json**(*obj*, *path*, *indent=2*, *ensure_ascii=False*, **args*, ***kwargs*)

Save object as json format

Parameters

- **obj** (*object*) – object to be saved as json format
- **path** (*str*) – json file path
- **indent** (*int*) – number of spaces for indentation
- **ensure_ascii** (*bool*) – whether to ensure ascii encoding
- **args** – other options for *json.dump*
- **kwargs** – other options for *json.dump*

svkcore.common.fileio.**save_pickle**(*obj*, *path*)

Save object as pickle format

Parameters

- **obj** (*object*) – object to be saved as pickle format
- **path** (*str*) – pickle file path

2.2.3 svkcore.common.np_ops

Some common numpy operations

svkcore.common.np_ops.**circle_kernel**(*diameter*, *dtype=<class 'numpy.int32'>*)

Create circle kernel

Parameters

- **diameter** (*int*) – diameter of the circle
- **dtype** (*numpy.dtype, optional*) – data type of the kernel

Returns
circle kernel

Return type
numpy.ndarray

svkcore.common.np_ops.**ellipse_kernel**(*ksize*, *dtype=<class 'numpy.int32'>*)

Create ellipse kernel

Parameters

- **ksize** (*tuple*) – kernel size, a tuple of (height, width)
- **dtype** (*numpy.dtype, optional*) – data type of kernel

Returns
ellipse kernel

Return type
numpy.ndarray

svkcore.common.np_ops.**generate_grid**(*panel_size*, *grid_size=None*, *grid_num=None*, *overlap_size=None*, *overlap_ratio=None*, *allow_cross_boundary=False*)

Generate grid for crop image patches

Parameters

- **panel_size** (*bool*) – a tuple of image size (height, width)
- **grid_size** – a tuple of grid size (grid_height, grid_width)
- **grid_num** – a tuple of grid num (grid_rows, grid_column)
- **overlap_size** – a tuple of overlap size
- **overlap_ratio** – a tuple of grid overlap ratio
- **allow_cross_boundary** – allow the last row or column position cross panel or not

Returns
grids [rows, columns, 4], each grid consists (ymin, xmin, ymax, xmax)

Return type
np.ndarray

svkcore.common.np_ops.**ndarray_index**(*shape*)

Create np.ndarray index

Parameters
shape (*Union[list, tuple]*) – index array shape

Returns
np.ndarray index

Return type
np.ndarray

svkcore.common.np_ops.**nms_mask**(*seg*, *ksize=3*, *dtype=None*)

Non-maximum suppression mask for segmentation

Parameters

- **seg** (*np.ndarray*) – segmentation result, probability map between [0.0, 1.0]
- **ksize** (*int*) – kernel size for max pooling
- **dtype** (*np.dtype*) – data type of output mask

Returns
non-maximum suppression mask

Return type
np.ndarray

svkcore.common.np_ops.**points_distance**(*points0*, *points1*, *weights=(1.0, 1.0)*)

Calculate distances between two point array

Parameters

- **points0** (*np.ndarray*) – a numpy array of shape (n, 2) representing the first set of points
- **points1** (*np.ndarray*) – a numpy array of shape (m, 2) representing the second set of points
- **weights** (*tuple*) – a tuple of two floats representing the weights for the distance calculation along the vertical and horizontal axes

Returns
a numpy array of shape (n, m) representing the distances between each pair of points from points0 and points1

Return type
np.ndarray

svkcore.common.np_ops.**seg2line**(*seg*, *fb_threshold=0.5*, *smooth_width=3*, *partition_width=20*, *partition_height=30*)

Find all valid lines for segmentation

Parameters

- **seg** (*np.ndarray*) – point object segmentation result, probability map between [0.0, 1.0]
- **fb_threshold** (*float*) – foreground vs background threshold
- **smooth_width** (*float*) – bin width for line x-coordinate smooth
- **partition_width** (*float*) – line will split when x-coordinate interval greater than partition_width
- **partition_height** (*float*) – line will split when y-coordinate interval greater than partition_height

Returns
a list of lines

Return type
list

svkcore.common.np_ops.**seg2point**(*seg*, *max_diameter: int*, *min_distance*, *fb_threshold: float = 0.5*, *min_fore_count: int = 1*, *max_fore_count: int = -1*, *avg_fore_score: float = 0.55*, *distance_weights=(1.0, 1.0)*)

Find all valid points from segmentation

Parameters

- **seg** (*np.ndarray*) – point object segmentation result, probability map between [0.0, 1.0]
- **max_diameter** (*int*) – max_radius for a point
- **min_distance** (*float*) – min distance between two point center
- **fb_threshold** (*float*) – foreground vs background threshold
- **min_fore_count** (*int*) – The minimum count of foreground pixel
- **max_fore_count** (*int*) – The maximum count of foreground pixel

- **avg_fore_score** (*float*) – The minimum average fore score of foreground
- **distance_weights** (*tuple*) – distance value's weights of vertical and horizontal

Returns
A list of point

Return type
np.ndarray

2.3 svkcore.shapes

2.3.1 svkcore.shapes.shapes

Common shapes for object detection

class svkcore.shapes.shapes.**Box**(*obj*, *dtype=None*)

Bases: *Shape*

Box of 2d. Record top_left and bottom_right corner position of box.

area()

Calculate box area

Returns
box area

Return type
np.float

bsize()

Size of box

Returns
box size in format np.array([width, height])

Return type
np.ndarray

center()

Center point of a box

Returns
center point

Return type
Point

classmethod from_cxywh(*cxywh*)

Create box from format [min-x, min-y, max-x, max-y]

Returns
converted box in format [min-x, min-y, max-x, max-y]

Return type
Box

scale(*scale*)

Scale box

Parameters
scale – scale factor

Returns
a scaled box

Return type
Box

to_cxywh()

Convert box format to [center-x, center-y, width, height]

Returns
converted box in format [center-x, center-y, width, height]

Return type
np.ndarray

to_mask(*size=None*)

Convert box to mask

Parameters
size –

Returns

to_polygon()

Convert box to polygon from top_left and across top_right, bot_right and end to bot_left

Returns
the converted polygon

Return type
Polygon

class svkcore.shapes.shapes.**Boxes**(*obj*, *dtype=None*)

Bases: *Shape*

Collection of Box

areas()

Calculate boxes areas

Returns
boxes areas

Return type
np.ndarray

bsize()

Sizes of boxes

Returns
boxes sizes in format np.array([[width, height], ...])

Return type
np.ndarray

center()

Center points of a boxes

Returns
center points

Return type
Points

classmethod from_cxywh(*cxywh*)

Create box from format [min-x, min-y, max-x, max-y]

Returns
converted box in format [min-x, min-y, max-x, max-y]

Return type
Boxes

scale(*scale*)

Scale boxes

Parameters
scale – scale factor

Returns
a scaled boxes

Return type
Boxes

to_cxywh()

Convert box format to [center-x, center-y, width, height]

Returns
converted boxes in format [center-x, center-y, width, height]

Return type
np.ndarray

class svkcore.shapes.shapes.**Line**(*obj*, *dtype=None*)

Bases: *Points*

Line of 2d

length()

Calculate the length of a continuous line

Returns
the line length

Return type
np.float

class svkcore.shapes.shapes.**Mask**(*obj*, *dtype=None*)

Bases: *Shape*

Mask of 2d

area()

Calculate mask area

Returns
mask area

Return type
np.float

`bounding_box()`

Get the minimum bounding box of this mask

Returns
then minimum bounding box of this mask

Return type
Box

`swap()`

Swap mask shape coordinate order from x-y to y-x

Returns
swapped this mask

Return type
Mask

`class svkcore.shapes.shapes.Point(obj, dtype=None)`

Bases: *Shape*

Point of 2d

`class svkcore.shapes.shapes.Points(obj, dtype=None)`

Bases: *Point*

Collection of Points

`bounding_box()`

Get the minimum bounding box a collection of points

Returns
bounding box

Return type
Box

`class svkcore.shapes.shapes.Polygon(obj, dtype=None)`

Bases: *Shape*

Polygon of 2d

`area()`

Calculate polygon area use mask area calculate

Returns
polygon area

Return type
np.float

`bounding_box()`

Get the minimum bounding box of this polygon

Returns
bounding box

Return type
Box

to_mask(*size=None*)

Convert polygon to mask

Parameters
size – the final mask size. Default is None means use the minimum size that can overlap this mask

Returns
converted mask from polygon

Return type
np.ndarray

class svkcore.shapes.shapes.**Shape**(*obj*, *dtype=None*)

Bases: object

Base class for object detection shapes. Wrap a numpy.array object and add specified operations for each type shape.

numpy()

Get numpy data

Returns
shape data

Return type
np.ndarray

order = 0

swap()

Swap this shape coordinate order from x-y to y-x

Returns
swapped this shape

Return type
Shape

2.4 svkcore.visualize

2.4.1 svkcore.visualize.visualize

svkcore.visualize.visualize.**cv2image2pil**(*cv2_image: ndarray*) → Image

Convert openCV format image to PIL.Image.Image

Parameters
cv2_image – openCV format image instance

Returns
converted Image.Image instance

svkcore.visualize.visualize.**draw_annotation**(*image*, *annotation*, *name2cls*, *color_table=None*, *add_unknown_name=False*)

Draw DTAnnotation to an image

Parameters

- **image** (*PIL.Image.Image*) – A PIL.Image.Image object

- **annotation** (DTAnnotation) – An instance of DTAnnotation
- **name2cls** (*dict*) – a dict of name to its class id number
- **color_table** (*list*) – each class colors
- **add_unknown_name** (*bool*) – whether add a new name to name2cls, default is False

Returns
drew image

Return type
PIL.Image.Image

svkcore.visualize.visualize.**draw_boxes**(*image*, *boxes*, *color='red'*, *width=0*, *fullfill=False*)

Draw boxes on image

Parameters

- **image** – A PIL.Image object
- **boxes** – a list of boxes
- **color** – boxes color
- **width** – line width
- **fullfill** – full fill boxes or not

Returns
drew image

svkcore.visualize.visualize.**draw_boxes_texts**(*image*, *boxes*, *texts*, *width=1*, *color='red'*)

Draw boxes and its text information on image

Parameters

- **image** – A PIL.Image object
- **boxes** – A list of box
- **texts** – A list of text
- **width** – Line width, determines thickness of box and font size of text
- **color** – Color of boxes and texts

Returns
drew image

svkcore.visualize.visualize.**draw_detection_result**(*image*, *boxes*, *classes*, *display_strings*, *color_nums=100*, *scale=-1.0*)

Draw detection result

Parameters

- **image** – A PIL.Image object
- **boxes** – A list of box
- **classes** – A list of detection class index
- **display_strings** – A list of display string
- **color_nums** – The max number of different colors
- **scale** – visualize box and text scale. Default -1.0 means auto adjust scale by input image

Returns
drew image

svkcore.visualize.visualize.**draw_lines**(*image*, *lines*, *color='red'*, *width=0*)

Draw boxes on lines

Parameters

- **image** – A PIL.Image object
- **lines** – a list of lines
- **color** – lines color
- **width** – line width

Returns
drew image

svkcore.visualize.visualize.**draw_masks**(*image*, *masks*, *color='red'*, *alpha=0.5*)

Draw masks on image

Parameters

- **image** – A PIL.Image object
- **masks** – a list of masks
- **color** – mask color
- **alpha** – transport alpha

Returns
drew image

svkcore.visualize.visualize.**draw_points**(*image*, *points*, *color='red'*, *scale=3*, *shape='.'*)

Draw boxes on points

Parameters

- **image** – A PIL.Image object
- **points** – a list of points
- **color** – points color
- **scale** – points scale
- **shape** – visualize shape, ‘t’ for triangle else for circle

Returns
drew image

svkcore.visualize.visualize.**draw_polygons**(*image*, *polygons*, *color='red'*, *width=0*, *fullfill=False*)

Draw polygons on image

Parameters

- **image** – A PIL.Image object
- **polygons** – a list of polygons
- **color** – mask color
- **width** – line width
- **fullfill** – full fill polygon or not

Returns
drew image

svkcore.visualize.visualize.**draw_texts**(*image*, *xys*, *texts*, *color='red'*, *back_color=None*, *font_size=12*, *position='topleft'*, *offset=(0, 0)*, *margin=(0, 0, 0, 0)*)

Draw texts on image

Parameters

- **image** – A PIL.Image object
- **xys** – A list of corner's coordinate where text align
- **texts** – A list of texts
- **color** – Text color
- **back_color** –
- **font_size** – Font size
- **position** – text align position, enum string like: topleft/topright/bottomleft/bottomright/manu
- **offset** – A tuple vector of offset for 'manu' position
- **margin** – A tuple (left, top, right, bottom) denotes the margins to back boarder

Returns
drew image

svkcore.visualize.visualize.**generate_colors**(*num*)

Generate colors for drawing bounding boxes

svkcore.visualize.visualize.**images_gallery**(*image_list: Tuple[Image] | List[Image]*, *n_cols: int = 6*, *n_rows: int | None = None*, *cell_size: Tuple[int] | List[int] = (224, 224)*, *pad: int = 16*, *align: int = 0*, *back_color: Tuple[int] | List[int] | str = 'black'*, *same_scale: bool = False*) → Image

Paste a list of images into one panel for better visualize

Parameters

- **image_list** – A list of Image.Image instance.
- **n_cols** – Max number of images to show in each row. If the number of `image_list` is less than `n_cols`, `n_cols` will be set as the number of `image_list`. Default value is 6.
- **n_rows** – Number of rows to show images. If `n_cols` is set, `n_rows` will be automatically calculated by

$$ceil(\frac{N}{n\_cols})$$

Otherwise `n_cols` will be automatically calculated. Default value is None.

- **cell_size** – The size of cell where each image is placed in.
- **pad** – The pad width/height between two cells.
- **align** – The align mode. Set 0(center), 1(left/up), 2(right/bottom) to choose align mode in each cell.
- **back_color** – Background color.

- **same_scale** – A boolean value indicates whether to use a same scale factor to resize all images in image_list. Set it to be True if you want to visualize the sizes of different images.

Returns
Pasted Image.Image instance.

svkcore.visualize.visualize.**pil2cv2image**(*image: Image*) → ndarray

Convert PIL.Image.Image to openCV format image

Parameters
image – an instance of PIL.Image.Image

Returns
converted openCV format image

CHAPTER

THREE

CHANGE LOGS

3.1 2023/03/10

- add new feature to `image_gallery`: resize all image by same scale
- add setup.py to install package
- add docs

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

W