Image Semantics Documentation

Release 0.0.0

Justin Brooks

Sep 17, 2020

Contents

Warning: Currently a work in progress!

With many image annotation semantics existing in the field of computer vision, it can become daunting to manage. This package provides the ability to convert and visualize many different types of annotation formats for object dectection and localization.

```
$ pip install imantics
```

or use the latest build with

```
$ git clone https://github.com/jsbroks/imantics/
$ cd /imantics
$ pip install imantics
```

Tutorial A quick tutorial with examples of exporting visualize and converting datasets.

API If you are looking for information on a specific function, class or method, this part of the documentation is for you.

CHAPTER 1

Contributing

Yes please! We are always looking for contributions, additions and improvements.

The source is available on GitHub and contributions are always encouraged. Contributions can be as simple as minor tweaks to this documentation, the website or the core.

To contribute, fork the project on GitHub and send a pull request.

CHAPTER 2

Table of Content

2.1 Tutorial

2.2 API

This part of the documentation covers all the interfaces of Image Segmantic.

2.2.1 Annotation Object

class imantics.**Annotation**(*image=None*, *category=None*, *bbox=None*, *mask=None*, *polygons=None*, *id=0*, *color=None*, *metadata={}*, *width=0*, *height=0*)

Annotation is a marking on an image.

This class acts as a level ontop of *BBox*, *Mask* and *Polygons* to manage and generate other annotations or export formats.

area
Qantity that expresses the extent of a two-dimensional figure

array
Numpy array boolean mask repsentation of the annotations

bbox
BBox repsentation of the annotations

coco(*include=True*)
Generates COCO format of annotation

Parameters **include** (*bool*) – True to include all COCO formats, Fale to generate just annotation format

Returns COCO format of annotation

Return type dict

export (*style='coco'*)
Exports object into specified style

classmethod from_bbox (*bbox*, *image=None*, *category=None*)
Creates annotation from bounding box

Parameters

- **image** (*Image*) – image assoicated with annotation
- **category** (*Category*) – category to label annotation
- **polygons** (*BBox*, list, tuple) – bbox to create annotation from

classmethod from_mask (*mask*, *image=None*, *category=None*)
Creates annotation class from a mask

Parameters

- **image** (*Image*) – image assoicated with annotation
- **category** (*Category*) – category to label annotation
- **mask** (*Mask*, numpy.ndarray, list) – mask to create annotation from

classmethod from_polygons (*polygons*, *image=None*, *category=None*)
Creates annotation from polygons

Accepts following format for lists:

```
# Segmentation Format
[
    [x1, y1, x2, y2, x3, y3,...],
    [x1, y1, x2, y2, x3, y3,...],
    ...
]
```

or

```
# Point Format
[
    [[x1, y1], [x2, y2], [x3, y3],...],
    [[x1, y1], [x2, y2], [x3, y3],...],
    ...
]
```

No sepcificaiton is reqiured between which format is used

Parameters

- **image** (*Image*) – image assoicated with annotation
- **category** (*Category*) – category to label annotation
- **polygons** (*Polygons*, list) – polygons to create annotation from

mask
Mask representation of the annotations

paperjs ()
Export object in PaperJS format

Returns object in format

Return type dict

polygons
Polygons repsentation of the annotations

set_image(*image*)
Sets the annotaiton image information

size
Tuple of width and height

vgg()
Export object in VGG format

voc()
Export object in VOC format

Returns object in format

Return type lxml.element

yolo(*as_string=True*)
Generates YOLO format of annotation (using the bounding box)

Parameters **as_string** (*bool*) – return string (true) or tuple (false) representation

Returns YOLO repersentation of annotation

Return type str, tuple

2.2.2 Bounding Box Object

class imantics.**BBox**(*bbox*, *style=None*)
Bounding Box is an enclosing retangular box for a image marking

INSTANCE_TYPES = (<class 'numpy.ndarray'>, <class 'list'>, <class 'tuple'>)
Value types of *BBox*

MIN_MAX = 'minmax'
Bounding box format style [x1, y1, x2, y2]

WIDTH_HEIGHT = 'widthheight'
Bounding box format style [x1, y1, width, height]

bbox(*style=None*)
Generates tuple repersentation of bounding box

Parameters **style** – stlye to generate bounding box (defaults: MIN_MAX)

Returns tuple of bounding box with specified style

bottom_left
Tops left point of the bounding box:

```
[ ]------[ ]
 |        |
 |        |
[X]------[ ]
```

bottom_right
Tops left point of the bounding box:

```
[ ]------[ ]
 |        |
 |        |
[ ]------[X]
```

classmethod create(*bbox*, *style=None*)
Creates *BBox*

Recommend over the use of __init__.

draw(*image*, *color=None*, *thickness=2*)
Draws a bounding box to the image array of shape (width, height, 3)

This function modifies the image array

Parameters

- **color** (*tuple, list*) – RGB color repersentation
- **thickness** – pixel thickness of box

classmethod empty()

Returns Empty *BBox* object

classmethod from_mask(*mask*)
Creates *BBox* from mask

Parameters **mask** (*Mask*, numpy.ndarray, list) – object to generate bounding box

Returns *BBox* repersentation

classmethod from_polygons(*polygons*)
Creates *BBox* from polygons

Parameters **polygons** (*Polygons*, list) – object to generate bounding box

Returns *BBox* repersentation

mask(*width=None*, *height=None*)
Returns or generates *Mask* representation of bounding box.

Returns Mask representation

Return type *Mask*

max_point
Maximum points of the bounding box (x2, y2)

min_point
Minimum points of the bounding box (x1, y1)

polygons()
Returns or generates *Polygons* representation of bounding box.

Returns Polygon representation

Return type *Polygons*

size
Width and height as a tuple (width, height)

top_left
Tops left point of the bounding box:

```
[X]------[ ]
 |         |
 |         |
[ ]------[ ]
```

top_right
Tops right point of the bounding box:

```
[ ]------[X]
 |         |
 |         |
[ ]------[ ]
```

2.2.3 Category Object

class imantics.**Category**(*name*, *parent=None*, *metadata={}*, *id=0*, *color=None*)

coco(*include=True*)
Export object in COCO format

Returns object in format

Return type dict

export(*style='coco'*)
Exports object into specified style

paperjs()
Export object in PaperJS format

Returns object in format

Return type dict

vgg()
Export object in VGG format

voc()
Export object in VOC format

Returns object in format

Return type lxml.element

yolo()
Export object in YOLO format

Returns object in format

Return type list, tuple

2.2.4 Color Object

class imantics.**Color**(*hls=None*, *rgb=None*, *hex=None*)

classmethod create(*color*)
Creates color class

string - generates color from hex tuple and values between [0, 1] - generates from hls tuple and values between [0, 255] - generates from rgb

Parameters **color** – tuple, list, str

Returns color class

hex
Hex representation of color

hls
HLS representation of color

classmethod random(*h=(0, 1)*, *l=(0.35, 0.7)*, *s=(0.6, 1)*)
Generates a random color

Parameters

- **l** (*tuple*) – range for lightness
- **h** (*tuple*) – range for hue
- **s** (*tuple*) – range for saturation

Returns randomly generated color

Return type *Color*

rgb
RGB representation of color

2.2.5 Dataset Object

class imantics.**Dataset**(*name*, *images=[]*, *id=0*, *metadata={}*)

add(*image*)
Adds image(s) to the current dataset

Parameters **image** (*Image Annotation*, list, typle, path) – list, object or path to add to dataset

coco()
Export object in COCO format

Returns object in format

Return type dict

export(*style='coco'*)
Exports object into specified style

classmethod from_coco(*coco_obj*, *name='COCO Datset'*)
Generates a dataset from a COCO object or python dict

Parameters **coco_obj** (*dict, pycocotools.coco.COCO*) –

Raises **ImportError** – Raised if coco_obj is a *pycocotools.coco.COCO* object and it cannot be imported

iter_annotations()
Generator to iterate over all annotations

iter_categories()
Generator to iterate over all categories

iter_images()
Generator to iterate over all images

paperjs()
Export object in PaperJS format

Returns object in format

Return type dict

split(*ratios*, *random=False*)
Splits dataset images into mutiple sub datasets of the given ratios

If a tuple of (1, 1, 2) was passed in the result would return 3 dataset objects of 25%, 25% and 50% of the images.

```
percents = ratios / ratios.sum()
```

Parameters

- **ratios** (*tuple, list*) – ratios to split dataset into
- **random** – randomize the images before spliting

Returns tuple of datasets with length of the number of ratios

Return type tuple

vgg()
Export object in VGG format

voc()
Export object in VOC format

Returns object in format

Return type lxml.element

yolo()
Export object in YOLO format

Returns object in format

Return type list, tuple

2.2.6 Image Object

class imantics.**Image**(*image_array=None*, *annotations=[]*, *path=''*, *id=0*, *metadata={}*, *dataset=None*, *width=0*, *height=0*)

add(*annotation*, *category=None*)
Adds an annotation, list of annotation, mask, polygon or bbox to current image. If annotation is not a Annotation a category is required List of non-Annotaiton objects will have the same category

Parameters

- **annotation** – annotaiton to add to current image
- **category** – required if annotation is not an Annotation object

coco(*include=True*)
Export object in COCO format

Returns object in format

Return type dict

draw(*bbox=True*, *outline=True*, *mask=True*, *text=True*, *thickness=3*, *alpha=0.5*, *categories=None*, *text_scale=0.5*, *color_by_category=False*)
Draws annotations on top of the image. If no image is loaded, annotations will be applied to a black image array.

Parameters

- **bbox** – Draw bboxes
- **outline** – Draw mask outlines
- **mask** – Draw masks
- **alpha** – opacity of masks (only applies to masks)
- **thickness** – pixel width of lines for outline and bbox
- **color_by_category** – Use the annotations's category to us as color
- **categories** – List of categories to show

Returns Image array with annotations

Return type numpy.ndarray

classmethod empty(*width=0*, *height=0*)
Creates an empty *Image*

export(*style='coco'*)
Exports object into specified style

classmethod from_coco(*coco*, *dataset=None*)
Creates an *Image* from a dict in COCO formatted image

Parameters **coco** (*dict*) – COCO formatted image

Return type *Image*

classmethod from_folder(*directory*)
Creates *Image*'s from all images found in directory

Returns list of *Image*'s

classmethod from_path(*path*)
Returns an array of images if path is a directory Returns an *Image* if path is a file

iter_annotations()
Generator to iterate over all annotations

iter_categories()
Generator to iterate over all categories

paperjs()
Export object in PaperJS format

Returns object in format

Return type dict

vgg()
Export object in VGG format

voc(*pretty=False*)
Export object in VOC format

Returns object in format

Return type lxml.element

yolo()
Export object in YOLO format

Returns object in format

Return type list, tuple

2.2.7 Mask Object

class imantics.**Mask**(*array*)
Mask class

bbox()
Returns or generates *BBox* representation of mask.

Returns Bounding Box representation

Return type *BBox*

contains(*item*)
Checks whether a point (tuple), array or mask is within current mask.

Note: Masks and arrays must be fully contained to return True

Parameters **item** – object to check

Returns bool if item is contained

draw(*image*, *color=None*, *alpha=0.5*)
Draws current mask to the image array of shape (width, height, 3)

This function modifies the image array

Parameters

- **color** (*tuple, list*) – RGB color repersentation
- **alpha** (*float*) – opacity of mask

intersect(*other*)
Intersects the array of the specified mask with this masks's array and returns the result as a new mask.

Parameters **other** (*Mask*, numpy.ndarray) – mask to intersect with

Returns resulting *Mask*

invert()
Inverts current mask

Returns resulting *Mask*

iou(*other*)
Intersect over union value of the specified masks

Parameters **other** (*Mask*, numpy.ndarray) – mask to compute value with

Returns resulting float value

match(*item*, *threshold=0.5*)
Given a overlap threashold determines if masks match

Parameters

- **item** (*Mask*) – item to compare with
- **threshold** – max amount of overlap (percentage)

Returns boolean determining if the items match

polygons()
Returns or generates *Polygons* representation of mask.

Returns Polygons representation

Return type *Polygons*

subtract(*other*)
Subtracts the array of the specified mask from this masks's array and returns the result as a new mask.

Parameters **other** – mask (or numpy array) to subtract

Retrn resulting mask

union(*other*)
Unites the array of the specified mask with this mask's array and returns the result as a new mask.

Parameters **other** (*Mask*, numpy.ndarray) – mask to unite with

Returns resulting *Mask*

2.2.8 Polygons Object

class imantics.**Polygons**(*polygons*)

INSTANCE_TYPES = (<class 'list'>, <class 'tuple'>)
Polygon instance types

bbox()
Returns or generates *BBox* representation of polygons.

Returns Bounding Box representation

Return type *BBox*

draw(*image*, *color=None*, *thickness=3*)
Draws the polygons to the image array of shape (width, height, 3)

This function modifies the image array

Parameters

- **color** (*tuple, list*) – RGB color repersentation
- **thickness** – pixel thickness of box

classmethod from_bbox(*bbox*, *style=None*)
Creates *Polygons* from bounding box

Parameters **bbox** (*BBox*, list, tuple) – object to generate bounding box

Returns *Polygons* repersentation

classmethod from_mask(*mask*)
Creates *Polygons* from mask

Parameters **mask** (*Mask*, numpy.ndarray, list) – object to generate mask

Returns *Polygons* repersentation

mask(*width=None*, *height=None*)
Returns or generates *Mask* representation of polygons.

Returns Mask representation

Return type *Mask*

points
Returns polygon in point format:

```
[
    [[x1, y1], [x2, y2], [x3, y3], ...],
    [[x1, y1], [x2, y2], [x3, y3], ...],
    ...
]
```

segmentation
Returns polygon in segmentation format:

```
[
    [x1, y1, x2, y2, x3, y3, ...],
    [x1, y1, x2, y2, x3, y3, ...],
    ...
]
```

Python Module Index

i

Index

P

R

S

T

U

V

W

Y