polimorfo Documentation

Release 0.10.30

Fabio Fumarola

Jan 31, 2022

Contents:

CHAPTER 1

polimòrfo

Polimòrfo ($\pi o \lambda \mu o \rho o$, comp. di $\pi o \lambda \upsilon$- «poli-» e $\mu o \rho$ «forma») is a dataset loader and converter library for object detection segmentation and classification. The goal of the project is to create a library able to process dataset in format:

- COCO: Common Objects in Context
- Pascal VOC: Visual Object Classes Challenge
- Google Open Images: Object Detection and Segmentation dataset released by Google

and transform these dataset into a common format (COCO).

Moreover, the library offers utilies to handle (load, convert, store and transform) the various type of annotations. This is important when you need to: - convert mask to polygons - store mask in a efficient format - convert mask/poygons into bounding boxes

- Free software: Apache Software License 2.0
- Documentation: https://polimorfo.readthedocs.io.

1.1 Features

1.1.1 TODO

- [X] Coco dataset
- [X] download coco datasets for train and val
- [X] add annotations loader and converter
- [X] add the ability to create dataet from scratch

- [ ] add voc dataset format
- [ ]

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install polimorfo, run this command in your terminal:

```
$ pip install polimorfo
```

This is the preferred method to install polimorfo, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for polimorfo can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/fabiofumarola/polimorfo
```

Or download the tarball:

```
$ curl -OJL https://github.com/fabiofumarola/polimorfo/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use polimorfo in a project:

```
import polimorfo
```

CHAPTER 4

polimorfo

4.1 polimorfo package

4.1.1 Subpackages

4.1.1.1 polimorfo.datasets package

Submodules

polimorfo.datasets.coco module

class polimorfo.datasets.coco.**CocoDataset**(*coco_path: str*, *image_path: str = None*, *verbose: bool = True*)

Bases: object

Process the dataset in COCO format Data Format ——— annotation{

"id": int, "image_id": int, "category_id": int, "segmentation": RLE or [polygon], "area": float, "bbox": [x,y,width,height], "iscrowd": 0 or 1,

} categories[{ "id": int, "name": str, "supercategory": str, }]

add_annotation(*img_id: int, cat_id: int, segmentation: List[List[int]], area: float, bbox: List[T], is_crowd: int, score: float = None*) → int

add a new annotation to the dataset

Args: img_id (int): [description] cat_id (int): [description] segmentation (List[List[int]]): [description] area (float): [description] bbox (List): [description] is_crowd (int): [description] score (float): [optional score of the prediction]

Returns: int: [description]

add_category(*name: str*, *supercategory: str*) → int

add a new category to the dataset

Args: name (str): [description] supercategory (str): [description]

Returns: int: cat id

add_image(*file_name: Union[str, pathlib.Path], height: int, width: int, **kwargs*) → int
Add a new image to the dataset .

Args: file_name (Union[str, Path]): the file name holding the image height (int): the height of the image width (int): the width of the image

Returns: int: [description]

compute_area() → None
compute the area of the annotations

copy()
returns a copy of the given dataset

Returns: [CocoDataset]: a copy of the dataset

count_annotations_per_category(*sort_by='value'*) → Dict[str, int]
the count of annotations per category

Args:s sort_by (str, optional): [description]. Defaults to 'value'.

Returns: list – a list of tuples (category_name, number of annotations)

count_images_per_category()
get the number of images per category Returns:

list – a list of tuples category number of images

crop_image(*img_idx: int, bbox: Tuple[float, float, float, float], dst_path: pathlib.Path*) → str
crop the image id with respect the given bounding box to the specified path

Args: img_idx (int): the id of the image bbox (Tuple[float, float, float, float]): a bounding box with the format [Xmin, Ymin, Xmax, Ymax] dst_path (Path): the path where the image has to be saved

Returns: str: the name of the image

dump(*path=None*, ***kvargs*)
dump the dataset annotations and the images to the given path

Args: path ([type]): the path to save the json and the images

Raises: ValueError: [description]

dumps()
dump the filtered annotations to a json Returns:

object – an object with the dumped annotations

enlarge_box(*bbox*, *height*, *width*, *pxls=10*)
enlarge a given box of pxls pixels

Args: bbox ([type]): a tuple, list of np.arry of shape (4,) height (int): the height of the image width (int): the width of the image pxls (int, optional): the number of pixels to add. Defaults to 10.

Returns: boundingbox: the enlarged bounding box

get_annotations(*img_idx: int*, *category_idxs: List[int] = None*) → List[T]
returns the annotations of the given image

Args: img_idx (int): the image idx category_idxs (List[int]): the list of the category to filter the returned annotations

Returns: List: a list of the annotations in coco format

get_segmentation_mask(*img_idx: int*, *cats_idx: List[int] = None*, *remapping_dict: Dict[int*, *int] = None*, *min_conf: float = 0.5*) → Tuple[PIL.Image.Image, float]
generate a mask and weight for the given image idx

Args: img_idx (int): [the id of the image] cats_idx (List[int], optional): [an optional filter over the classes]. Defaults to None. remapping_dict (Dict[int, int], optional): [description]. Defaults to None. min_conf (float): the min confidence to generate the segment, segments with conf below the threshold are replaced as 255 ignore_index (int): the value used to replace segments with confidence below min_conf

Returns: Tuple[Image.Image, float]: [description]

get_segmentation_mask_multilabel(*img_idx: int*, *cats_idx: List[int] = None*, *remapping_dict: Dict[int*, *int] = None*, *min_conf: float = 0.5*) → Tuple[numpy.ndarray, float]
get a segmentation mask for multilabel task with shape [C, H, W]

Args: img_idx (int): [description] cats_idx (List[int], optional): [description]. Defaults to None. remapping_dict (Dict[int, int], optional): [description]. Defaults to None. min_conf (float, optional): [description]. Defaults to 0.5.

Returns: Tuple[np.np.ndarray, float]: [description]

images_path

keep_categories(*ids: List[int], remove_images: bool = False*)
keep images and annotations only from the selected categories Arguments:

id_categories {list} – the list of the id categories to keep

load_anns(*ann_idxs*)

load_image(*idx*)
load an image from the idx

Args: idx ([int]): the idx of the image

Returns: [Pillow.Image]: []

make_index()

mean_pixels(*sample: int = 1000*) → List[float]
compute the mean of the pixels

Args: sample (int, optional): [description]. Defaults to 1000.

Returns: List[float]: [description]

merge_categories(*cat_to_merge: List[str], new_cat: str*) → None

Merge two or more categories labels to a new single category. Remove from __content the category to be merged and update annotations cat_ids and reindex data with update content.

Args: cat_to_merge (List[str]): categories to be merged new_cat (str): new label to assign to the merged categories

merge_category_ids(*cat_to_merge: Union[List[int], List[str]], new_cat: str*) → None

Merge two or more categories labels to a new single category. Remove from __content the category to be merged and update annotations cat_ids and reindex data with update content.

Args: cat_to_merge (List[int | str]): categories to be merged new_cat (str): new label to assign to the merged categories

move_annotation(*idx: int, bbox: Tuple[float, float, float, float]*) → Dict[KT, VT]
move the bounding box and the segments of the annotation with respect to given bounding box

Args: idx (int): the annotation idx bbox (Tuple[float, float, float, float]): the bounding box

Returns: Dict: a dictioary with the keys iscrowd, bboox, area, segmentation

reindex(*by_image_name=True*)
reindex images and annotations to be zero based and categories one based

remap_categories(*remapping_dict: Dict[int, int]*) → None

remove_annotations(*ids: List[int], remove_images: bool = False*) → None
Remove from the dataset all the annotations ids passes as parameter

Arguments:

img_ann_ids {Dict[int, List[Int]]} – the dictionary of image id annotations ids to remove

remove_categories(*idxs: List[int], remove_images: bool = False*) → None
Remove the categories with the relative annotations

Args: idxs (List[int]): [description]

remove_images(*image_idxs: List[int]*) → None
remove all the images and annotations in the specified list

Arguments: image_idxs {List[int]} – [description]

remove_images_without_annotations()

remove_missing_images()
remove the images missing from images folder

save_idx_class_dict(*path: Union[str, pathlib.Path] = None*) → pathlib.Path
save the idx class dict for the dataset

Args: path (Union[str, Path], optional): [description]. Defaults to None.

Returns: Path: [description]

save_images_and_masks(*path: Union[str, pathlib.Path], cats_idx: List[int] = None, remapping_dict: Dict[int, int] = None, min_conf: float = 0.5, min_num_annotations: int = None, mode: polimorfo.datasets.coco.MaskMode = <MaskMode.MULTICLASS: 1>*) → Tuple[pathlib.Path, pathlib.Path]

Save images and segmentation mask into folders:

- segments
- images
- weights.csv that contains the pairs image_name, weight

children of the specified path

Args: path (Union[str, Path], optional): the path to save the masks. Defaults to None. cats_idx (List[int], optional): [an optional filter over the classes]. Defaults to None. remapping_dict (Dict[int, int], optional): a remapping dictionary for the index to save. Defaults to None. min_conf (float): the min confidence to generate the segment, segments with conf below the threshold are replaced as 255 ignore_index (int): the value used to replace segments with confidence below min_conf min_num_annotations (int, optional): [description]. Defaults to None.

save_segmentation_masks(*path: Union[str, pathlib.Path] = None, cats_idx: List[int] = None, remapping_dict: Dict[int, int] = None, min_conf: float = 0.5, mode: polimorfo.datasets.coco.MaskMode = <MaskMode.MULTICLASS: 1>*) → None
save the segmentation mask for the given dataset

Args: path (Union[str, Path], optional): the path to save the masks. Defaults to None. cats_idx (List[int], optional): [an optional filter over the classes]. Defaults to None. remapping_dict (Dict[int, int], optional): a remapping dictionary for the index to save. Defaults to None. min_conf (float): the min confidence to generate the segment, segments with conf below the threshold are replaced as 255 mode: (MaskMode): the mode to save the mask if multiclass are saved as png else as npy file

show_image(*img_idx: int = None*, *img_name: str = None*, *anns_idx: List[int] = None*, *ax=None*, *title: str = None*, *figsize=(18, 6)*, *colors=None*, *show_boxes=False*, *show_masks=True*, *min_score=0.5*, *min_area: int = 0*, *cats_idx: List[int] = None*, *color_border_only: bool = False*, *line_width: int = 2*, *font_size: int = 10*) → matplotlib.axes._axes.Axes

show an image with its annotations

Args:

img_idx (int, optional): the idx of the image to load (Optional: None) in case the value is not specified take a random id

img_name (str, optional): the name of the image to load anns_idx (List[int], optional): [description]. Defaults to None. ax ([type], optional): [description]. Defaults to None. title (str, optional): [description]. Defaults to None. figsize (tuple, optional): [description]. Defaults to (18, 6). colors ([type], optional): [description]. Defaults to None. show_boxes (bool, optional): [description]. Defaults to False. show_masks (bool, optional): [description]. Defaults to True. min_score (float, optional): [description]. Defaults to 0.5. cats_idx (List, optional): the list of categories to show. Defaults to None to display all the categories color_border_only (bool, optional): if True color only the border of the component. Defaults to False, font_size (int, optional): the font size default is 10

Returns: plt.Axes: [description]

show_images(*idxs_or_num: Union[List[int], int] = None*, *num_cols=4*, *figsize=(32, 32)*, *show_masks=True*, *show_boxes=False*, *min_score: float = 0.5*, *min_area: int = 0*, *cats_idx: List[int] = None*, *color_border_only: bool = False*, *line_width: int = 2*, *font_size: int = 10*, *colors=None*) → matplotlib.figure.Figure

show the images with their annotations

Args:

img_idxs (Union[List[int], int]): a list of image idxs to display or the number of images (Optional: None) If None a random sample of 8 images is taken from the db

num_cols (int, optional): [description]. Defaults to 4. figsize (tuple, optional): [description]. Defaults to (32, 32). show_masks (bool, optional): [description]. Defaults to True. show_boxes (bool, optional): [description]. Defaults to False. min_score (float, optional): [description]. Defaults to 0.5. min_area (int, optional): the min area of the annotations to display, Default to 0

Returns: plt.Figure: [description]

update_images_path(*func*)

update the images path Args:

update_images (UpdateImages): a class with a callable function to change the path

polimorfo.datasets.utils module

Module contents

4.1.1.2 polimorfo.utils package

Submodules

polimorfo.utils.datautils module

polimorfo.utils.imageutils module

polimorfo.utils.maskutils module

polimorfo.utils.maskutils.**mask_to_polygon**(*mask*, *min_score: float = 0.5*, *approx: float = 0.0*, *relative: bool = True*)

generate polygons from masks

Args: mask (np.ndarray): a binary mask min_score (float, optional): [description]. Defaults to 0.5. approx (float, optional): it approximate the polygons to reduce the number of points. Defaults to 0.0 relative (bool, optional): it the value of the approximation is computed on the relative amount of point or with respect to all the points

Returns: [type]: [description]

polimorfo.utils.maskutils.**polygons_to_mask**(*polygons*, *height*, *width*)

convert polygons to mask. Filter all the polygons with less than 4 points

Args: polygons ([type]): [description] height ([type]): [description] width ([type]): [description]

Returns: [type]: a mask of format num_classes, heigth, width

polimorfo.utils.maskutils.**area**(*mask*, *min_score=0.5*)

polimorfo.utils.maskutils.**bbox**(*polygons*, *height*, *width*)

polimorfo.utils.maskutils.**coco_poygons_to_mask**(*segmentations*, *height*, *width*) → numpy.ndarray

polimorfo.utils.visualizeutils module

class polimorfo.utils.visualizeutils.**BoxType**

Bases: enum.Enum

An enumeration.

xywh = 2

xyxy = 1

polimorfo.utils.visualizeutils.**change_color_brightness**(*color: Tuple*, *brightness_factor: float*)

Depending on the brightness_factor, gives a lighter or darker color i.e. a color with less or more saturation than the original color. Args:

color: color of the polygon. Refer to *matplotlib.colors* for a full list of formats that are accepted.

brightness_factor (float): a value in [-1.0, 1.0] range. A lightness factor of 0 will correspond to no change, a factor in [-1.0, 0) range will result in a darker color and a factor in (0, 1.0] range will result in a lighter color.

Returns:

modified_color (tuple[double]): a tuple containing the RGB values of the modified color. Each value in the tuple is in the [0.0, 1.0] range.

polimorfo.utils.visualizeutils.**create_text_labels**(*classes: List[int], scores: List[float], idx_class_dict: Dict[int, str]*)

Args: classes (list[int] or None): scores (list[float] or None): idx_class_dict (Dict[int, str] or None):

Returns: list[str]

polimorfo.utils.visualizeutils.**draw_instances**(*img: Union[PIL.Image.Image, numpy.ndarray], boxes: Union[numpy.ndarray, List[T]], labels: Union[numpy.ndarray, List[T]], scores: Union[numpy.ndarray, List[T]], masks: Union[numpy.ndarray, List[T]], idx_class_dict: Dict[int, str], title: str = '', figsize: Tuple = (16, 8), show_boxes: bool = False, show_masks: bool = True, min_score: float = 0.5, min_area: int = 0, colors: List[T] = None, ax: matplotlib.axes._axes.Axes = None, box_type: polimorfo.utils.visualizeutils.BoxType = <BoxType.xywh: 2>, only_class_idxs: List[int] = None, color_border_only: bool = False, alpha: float = 0.3, line_width: int = 2, font_size: int = 10, show_text: bool = True, *args, **kwargs*)

draw the instances from a object detector or an instance segmentation model

Args: img (np.ndarray): an image with shape (width, height, channels) boxes (np.ndarray): an array of shape (nboxes, 4) labels (np.ndarray): an array of shape (nlabels,) scores (np.ndarray): an array of shape (nscores,) masks (np.ndarray): an array of shape [num_masks, width, height] idx_class_dict (Dict[int, str]): a dictionary that maps class id to class name title (str, optional): [description]. Defaults to ''. figsize (Tuple, optional): [description]. Defaults to (16, 8). show_boxes (bool, optional): [description]. Defaults to False. show_masks (bool, optional): [description]. Defaults to True. min_score (float, optional): [description]. Defaults to 0.5. colors (List, optional): [description]. Defaults to None. ax (plt.Axes, optional): [description]. Defaults to None. box_type (BoxType, optional): [description]. Defaults to BoxType.xywh. only_class_idxs (List[int], optional): [description]. Defaults to None. color_only_border (bool): if true if color only the border (default is False) line_width: (float): the width of the line. Defaults is 2

Returns: [type]: [description]

polimorfo.utils.visualizeutils.**draw_segmentation**(*img: Union[numpy.ndarray, PIL.Image.Image], probs: numpy.ndarray, idx_name_dict: Dict[int, str], min_conf: float, colors: List[T] = None, title: str = '', ax: matplotlib.axes._axes.Axes = None, figsize: Tuple[int, int] = (16, 8), alpha: float = 0.3, fill: bool = True, min_score: float = 0.5*)

draw the result from a segmentation model

Args: img (Union[np.ndarray, Image.Image]): an PIL image or a numpy array probs (np.ndarray): it accepts:

- the logits coming from the model with shape (n_classes, H, W), or
- the mask coming from true annotations with shape (H,W) and containing pixel classification

idx_name_dict (Dict[int, str]): min_conf (float): the min confidence of the mask given as output colors

(List, optional): the colors to diplay categories. Defaults to None. title (str, optional): [description]. Defaults to ‘’. ax (plt.Axes, optional): [description]. Defaults to None. figsize (Tuple[int, int], optional): [description]. Defaults to (16, 8).

Returns: [plt.Axes]: the ax of the given plot

polimorfo.utils.visualizeutils.**draw_segmentation_multilabel**(*img: PIL.Image.Image, probs: numpy.ndarray, idx_name_dict: Dict[int, str], min_conf: float, colors: List[T] = None, title: str = '', ax: matplotlib.axes._axes.Axes = None, figsize: Tuple[int, int] = (16, 8)*)

draw the result from a segmentation model

Args: img (Union[np.ndarray, Image.Image]): an PIL image or a numpy array probs (np.ndarray): a probability tensor of shape (n_classes, H, W) idx_name_dict (Dict[int, str]): min_conf (float): the min confidence of the mask given as output colors (List, optional): the colors to diplay categories. Defaults to None. title (str, optional): [description]. Defaults to ‘’. ax (plt.Axes, optional): [description]. Defaults to None. figsize (Tuple[int, int], optional): [description]. Defaults to (16, 8).

Returns: [plt.Axes]: the ax of the given plot

polimorfo.utils.visualizeutils.**draw_text**(*ax: matplotlib.axes._axes.Axes*, *text: str*, *position: Tuple*, *font_size: float*, *color: str = 'g'*, *horizontal_alignment: str = 'center'*, *rotation: int = 0*)

Args: text (str): class label position (tuple): a tuple of the x and y coordinates to place text on image. font_size (int, optional): font of the text. If not provided, a font size

proportional to the image width is calculated and used.

color: color of the text. Refer to *matplotlib.colors* for full list of formats that are accepted.

horizontal_alignment (str): see *matplotlib.text.Text* rotation: rotation angle in degrees CCW

Returns: output (VisImage): image object with text drawn.

polimorfo.utils.visualizeutils.**generate_colormap**(*nelems: int*, *scaled: bool = False*, *bright: bool = True*)

Module contents

4.1.2 Submodules

4.1.3 polimorfo.cli module

Console script for polimorfo.

polimorfo.cli.**main**()

Console script for polimorfo.

4.1.4 polimorfo.polimorfo module

Main module.

4.1.5 Module contents

Top-level package for polimorfo.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/fabiofumarola/polimorfo/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

polimorfo could always use more documentation, whether as part of the official polimorfo docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/fabiofumarola/polimorfo/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *polimorfo* for local development.

1. Fork the *polimorfo* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/polimorfo.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv polimorfo
$ cd polimorfo/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 polimorfo tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.org/fabiofumarola/polimorfo/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Fabio Fumarola <fabiofumarola@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.2.0 (2020-02-18)

- Add support to process coco dataset

7.2 0.2.1 (2020-02-28)

- add support to download files and archives from the web and google drive

7.3 0.3.0 (2020-10-04)

- addedd support for removing categories and other utilities

7.4 0.4.0 (2020-10-05)

- addedd support to create a dataset from scratch

7.5 0.5.0 (2020-10-06)

- added support to visualize images and annotations
- make image removing optional during annotations and categories deletion

7.6 0.6.0 (2020-10-12)

- added copy dataset
- added split dataset

7.7 0.6.1 (2020-10-12)

- fixed a bug in colors generation for show images

7.8 0.6.2 (2020-10-12)

- update signature for function *def update_images_path(self, func):*

7.9 0.7.0 (2020-10-19)

- add method to dump dataset in format segmentation map

7.10 0.8.0 (2020-10-23)

- fixed bug in maskutils.mask_to_polygons
- add class to transform the predictions from instance and semantic segmentation in coco format
- fixed bug in add_image, add_annotation, add_category
- make load_image and load_images load random images sampled from the dataset

7.11 0.8.1 (2020-10-23)

- fixed bug for tqdm when removing a category and its annotations from the dataset

7.12 0.8.2 (2020-10-23)

- removed the prefix jpg when saving masks
- update draw instance to draw only bounding boxes

7.13 0.8.3 (2020-10-24)

- fixed bug in enum for draw instances

7.14 0.8.4 (2020-10-24)

- add show bounding boxes

7.15 0.8.5 (2020-10-24)

- changed representation for masks from [width, height, labels] to [labels, width, height]

7.16 0.8.6 (2020-10-24)

- added method to crop images
- added method to move annotations with respect a bounding box

7.17 0.8.7 (2020-10-24)

- support fully creation o a new dataset

7.18 0.8.8-11 (2020-10-26)

- fixed vairous bugs

7.19 0.8.12 (2020-10-26)

- fixed bug when the size of the segments is equal to 4

7.20 0.8.13 (2020-10-26)

- fixed bug in json dump to serialize numpy array

7.21 0.8.14 (2020-10-26)

- fixed bug in json dump to serialize numpy types

7.22 0.9.1 (2020-10-28)

- fixed various bugs
- add index for speedup lookup operations

7.23 0.9.2 (2020-10-28)

- add new feature to compute mean average precision and recall per class and global

7.24 0.9.3 (2020-10-28)

- add computation of mean average precision and mean average recall per image

7.25 0.9.4 (2020-10-28)

- fixed bug in score computation

7.26 0.9.36

- fixed bug in mask generation
- feature that allows us to add a single mask per component when saving segmentation results

7.27 0.9.38

- add min confidence when displaying prediction from a segmentation mask model
- now semantic coco accepts only logits to create annotations

7.28 0.9.39

- add new method to remap category idxs

7.29 0.9.48

- add new feature to save images and masks to a folder and filter out images and mask with less than k annotations

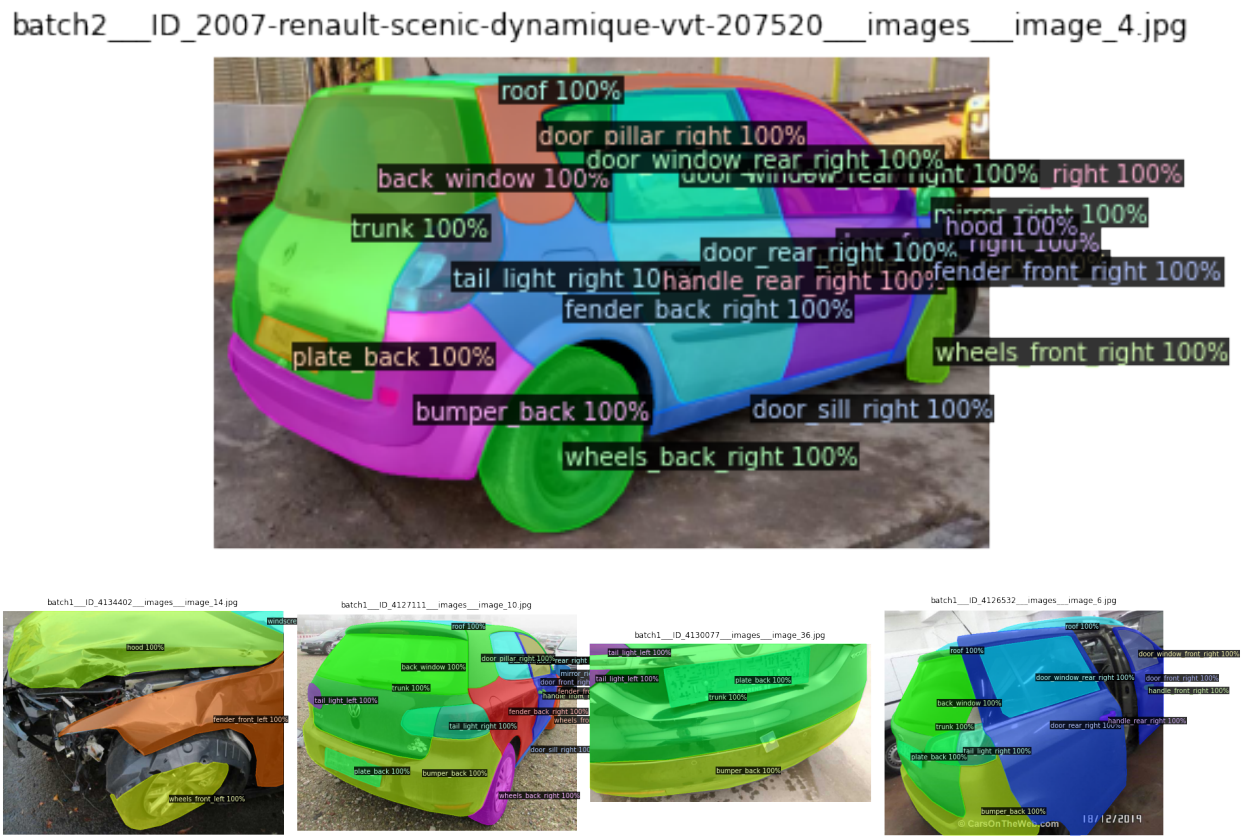
7.30 0.9.52

- the method *get_segmentation_mask* return also the avg score of the image annotations
- the method *save_mask_images* save also a weight files with the avg score for the image

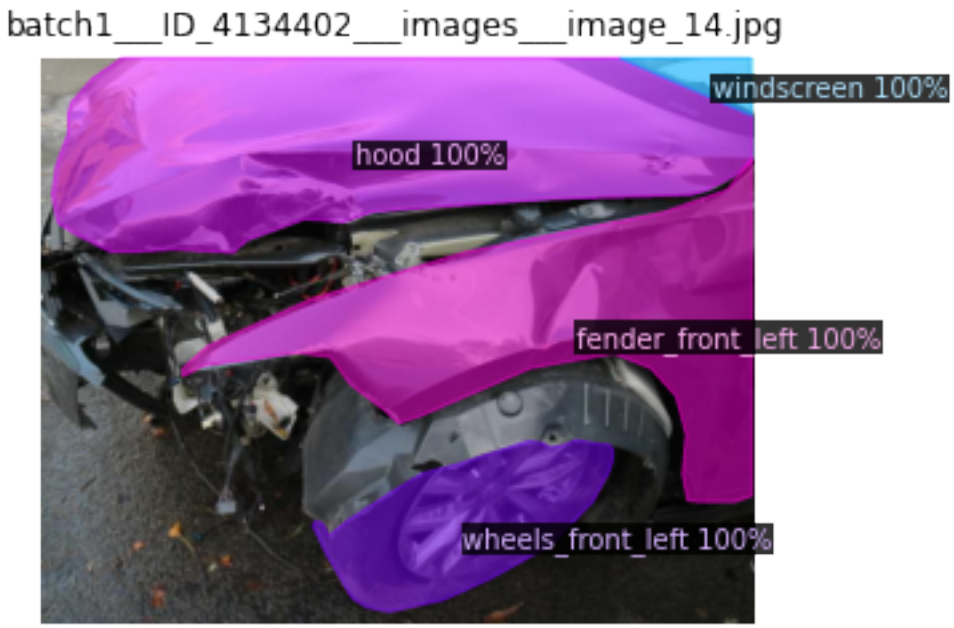
```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

```
load categories: 100%|| 42/42 [00:00<00:00, 141381.03it/s]
load images: 100%|| 8000/8000 [00:00<00:00, 1435238.12it/s]
load annotations: 100%|| 111876/111876 [00:00<00:00, 1191683.23it/s]
```

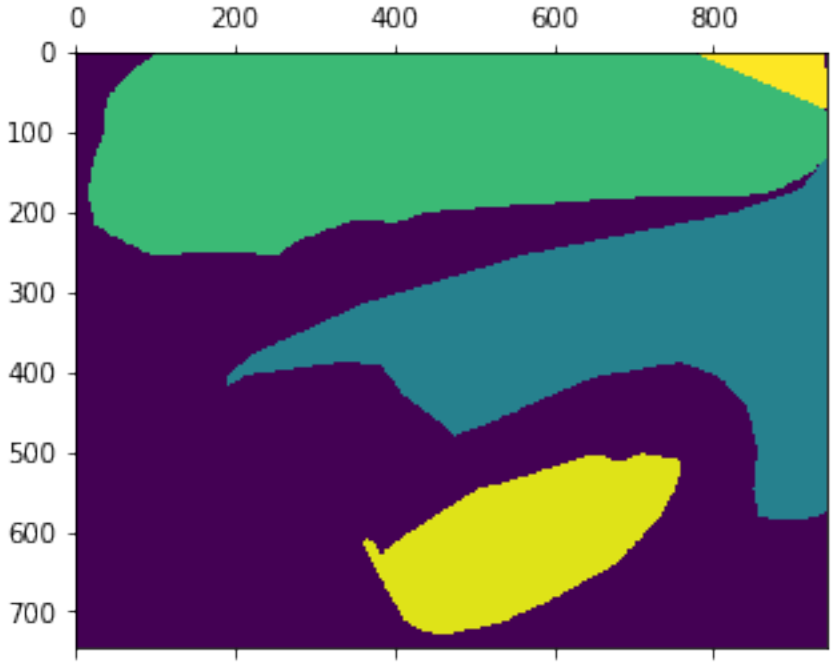
```
<AxesSubplot:title={'center':'batch2___ID_2007-renault-scenic-dynamique-vvt-207520___
↪images___image_4.jpg'}>
```




```
<AxesSubplot:title={'center':'batch1___ID_4134402___images___image_14.jpg'}>
```




```
<matplotlib.image.AxesImage at 0x7ff8892db690>
```




```
{1: {'supercategory': 'thing', 'id': 1, 'name': 'back_window'},
 2: {'supercategory': 'thing', 'id': 2, 'name': 'bumper_back'},
 3: {'supercategory': 'thing', 'id': 3, 'name': 'bumper_front'},
 4: {'supercategory': 'thing', 'id': 4, 'name': 'door_front_left'},
 5: {'supercategory': 'thing', 'id': 5, 'name': 'door_front_right'},
 6: {'supercategory': 'thing', 'id': 6, 'name': 'door_pillar_left'},
 7: {'supercategory': 'thing', 'id': 7, 'name': 'door_pillar_right'},
 8: {'supercategory': 'thing', 'id': 8, 'name': 'door_rear_left'},
```

(continues on next page)

(continued from previous page)

```
 9: {'supercategory': 'thing', 'id': 9, 'name': 'door_rear_right'},
 10: {'supercategory': 'thing', 'id': 10, 'name': 'door_sill_left'},
 11: {'supercategory': 'thing', 'id': 11, 'name': 'door_sill_right'},
 12: {'supercategory': 'thing', 'id': 12, 'name': 'door_window_front_left'},
 13: {'supercategory': 'thing', 'id': 13, 'name': 'door_window_front_right'},
 14: {'supercategory': 'thing', 'id': 14, 'name': 'door_window_rear_left'},
 15: {'supercategory': 'thing', 'id': 15, 'name': 'door_window_rear_right'},
 16: {'supercategory': 'thing', 'id': 16, 'name': 'fender_back_left'},
 17: {'supercategory': 'thing', 'id': 17, 'name': 'fender_back_right'},
 18: {'supercategory': 'thing', 'id': 18, 'name': 'fender_front_left'},
 19: {'supercategory': 'thing', 'id': 19, 'name': 'fender_front_right'},
 20: {'supercategory': 'thing', 'id': 20, 'name': 'fog_light_front_left'},
 21: {'supercategory': 'thing', 'id': 21, 'name': 'fog_light_front_right'},
 22: {'supercategory': 'thing', 'id': 22, 'name': 'handle_front_left'},
 23: {'supercategory': 'thing', 'id': 23, 'name': 'handle_front_right'},
 24: {'supercategory': 'thing', 'id': 24, 'name': 'handle_rear_left'},
 25: {'supercategory': 'thing', 'id': 25, 'name': 'handle_rear_right'},
 26: {'supercategory': 'thing', 'id': 26, 'name': 'headlight_left'},
 27: {'supercategory': 'thing', 'id': 27, 'name': 'headlight_right'},
 28: {'supercategory': 'thing', 'id': 28, 'name': 'hood'},
 29: {'supercategory': 'thing', 'id': 29, 'name': 'mirror_left'},
 30: {'supercategory': 'thing', 'id': 30, 'name': 'mirror_right'},
 31: {'supercategory': 'thing', 'id': 31, 'name': 'plate_back'},
 32: {'supercategory': 'thing', 'id': 32, 'name': 'plate_front'},
 33: {'supercategory': 'thing', 'id': 33, 'name': 'radiator_grill'},
 34: {'supercategory': 'thing', 'id': 34, 'name': 'roof'},
 35: {'supercategory': 'thing', 'id': 35, 'name': 'tail_light_left'},
 36: {'supercategory': 'thing', 'id': 36, 'name': 'tail_light_right'},
 37: {'supercategory': 'thing', 'id': 37, 'name': 'wheels_back_left'},
 38: {'supercategory': 'thing', 'id': 38, 'name': 'wheels_back_right'},
 39: {'supercategory': 'thing', 'id': 39, 'name': 'wheels_front_left'},
 40: {'supercategory': 'thing', 'id': 40, 'name': 'wheels_front_right'},
 41: {'supercategory': 'thing', 'id': 41, 'name': 'windscreen'},
 42: {'supercategory': 'thing', 'id': 42, 'name': 'trunk'}}
```

compute the stats for the images

```
back_window
image [  51273.    194488.25  239112.    374943.   3000000.  ]
cats [   1170.   18240.   28569.   45144. 1107195.]
mean_ratio [0.02281903 0.09378459 0.11947957 0.1204023  0.369065  ]
--------------
bumper_back
image [  51273.  200288.  254694.  399993. 3000000.]
cats [     40.     38345.25   68548.5   118346.25 2496960.  ]
mean_ratio [0.00078014 0.19145056 0.26914062 0.2958708  0.83232   ]
--------------
bumper_front
image [  67032.  204000.  265370.  422928. 3000000.]
cats [     48.   44255.   79600.  139527. 2101760.]
mean_ratio [0.00071608 0.21693627 0.29995855 0.32990722 0.70058667]
--------------
door_front_left
image [  76302.  207332.  265088.  410462. 3000000.]
cats [    276.    16905.5   28531.    60882.  2886408. ]
mean_ratio [0.00361721 0.08153831 0.10762841 0.14832555 0.962136  ]
--------------
```

(continues on next page)

(continued from previous page)

```
door_front_right
image [  67032.    209863.    269952.    415839.75 3000000.  ]
cats [    144.   17670.   30302.   65511. 2972513.]
mean_ratio [0.00214823 0.08419779 0.11224959 0.15753905 0.99083767]
--------------
door_pillar_left
image [  51273.   198678.   252909.   403531.5 3000000. ]
cats [    65.   6860.  15862.  32249. 833940.]
mean_ratio [0.00126772 0.03452823 0.06271821 0.07991693 0.27798   ]
--------------
door_pillar_right
image [  26700.   198369.   258137.5  408011.  3000000. ]
cats [   140.     6644.75  15793.5   32869.5  693248.  ]
mean_ratio [0.00524345 0.03349692 0.06118251 0.08056033 0.23108267]
--------------
door_rear_left
image [  79449.    206988.75  268247.5   423309.5  3000000.  ]
cats [    128.   14620.   27456.   67284. 2724480.]
mean_ratio [0.0016111  0.07063186 0.10235324 0.15894753 0.90816   ]
--------------
door_rear_right
image [  55944.  210255.  274816.  437920. 3000000.]
cats [     85.    15485.    28865.    70983.5 2998500. ]
mean_ratio [0.00151938 0.07364866 0.10503391 0.16209239 0.9995    ]
--------------
door_sill_left
image [  76302.   208703.   268736.   419131.5 3000000. ]
cats [    143.     8476.5   14600.    28554.  1844850. ]
mean_ratio [0.00187413 0.04061513 0.05432841 0.06812659 0.61495   ]
--------------
door_sill_right
image [  67032.   211658.5  271887.   415917.5 3000000. ]
cats [    104.     8994.    14900.    31085.5 1424000. ]
mean_ratio [0.0015515  0.04249298 0.05480218 0.07473958 0.47466667]
--------------
door_window_front_left
image [  76302.   204048.   259246.5  400143.  3000000. ]
cats [   195.   3952.   6683.  12796. 691698.]
mean_ratio [0.00255563 0.01936799 0.02577855 0.03197857 0.230566  ]
--------------
door_window_front_right
image [  67032.    205017.    265696.    411438.25 3000000.  ]
cats [    36.    4125.    6982.   14247.5 811831. ]
mean_ratio [0.00053706 0.02012028 0.02627815 0.03462853 0.27061033]
--------------
door_window_rear_left
image [  76302.   202286.   254940.   398317.5 3000000. ]
cats [     64.     2974.5    5895.5   12414.  1132524. ]
mean_ratio [0.00083877 0.01470443 0.02312505 0.03116609 0.377508  ]
--------------
door_window_rear_right
image [  55944.   203395.5  262656.   414411.5 3000000. ]
cats [   112.   3082.   6318.  13023. 731119.]
mean_ratio [0.002002   0.01515274 0.02405428 0.03142529 0.24370633]
--------------
fender_back_left
image [  51273.   203841.   259063.5  399792.5 3000000. ]
```

(continues on next page)

(continued from previous page)

```
cats [    117.    7542.   26138.   62227. 2363680.]
mean_ratio [0.0022819  0.03699943 0.10089418 0.15564824 0.78789333]
--------------
fender_back_right
image [  51273.   204253.5  263568.   408160.5 3000000. ]
cats [    105.    7185.   25920.   57277. 2524041.]
mean_ratio [0.00204786 0.03517688 0.09834274 0.1403296  0.841347  ]
--------------
fender_front_left
image [  76302.   203586.   259999.5  407153.5 3000000. ]
cats [    154.      5250.5    19468.5    47737.25 1990465.  ]
mean_ratio [0.0020183  0.02579008 0.07487899 0.11724632 0.66348833]
--------------
fender_front_right
image [  67032.    203363.25  261633.    403832.   3000000.  ]
cats [    240.    5460.   20252.   50460. 2764800.]
mean_ratio [0.00358038 0.02684851 0.07740614 0.12495295 0.9216    ]
--------------
fog_light_front_left
image [  67032.   198543.   255108.   390337.5 3000000. ]
cats [   56.   476.   924.  2006. 66676.]
mean_ratio [0.00083542 0.00239747 0.003622   0.00513914 0.02222533]
--------------
fog_light_front_right
image [  67032.   194880.   244872.   370834.5 3000000. ]
cats [    45.    460.    884.   1815. 156792.]
mean_ratio [0.00067132 0.00236043 0.00361005 0.00489437 0.052264  ]
--------------
handle_front_left
image [  76302.   203871.   263266.   404338.5 3000000. ]
cats [   24.   168.   273.   560. 91322.]
mean_ratio [0.00031454 0.00082405 0.00103697 0.00138498 0.03044067]
--------------
handle_front_right
image [  82404.   205897.   266954.   410512.5 3000000. ]
cats [    28.    168.    286.    630. 147026.]
mean_ratio [0.00033979 0.00081594 0.00107135 0.00153467 0.04900867]
--------------
handle_rear_left
image [  79449.  204585.  262104.  421806. 3000000.]
cats [    16.    168.    351.    800. 145408.]
mean_ratio [0.00020139 0.00082117 0.00133916 0.00189661 0.04846933]
--------------
handle_rear_right
image [  55944.   205173.   268584.   416925.5 3000000. ]
cats [    25.    170.    361.    800. 292545.]
mean_ratio [0.00044688 0.00082857 0.00134409 0.00191881 0.097515  ]
--------------
headlight_left
image [  67032.   194493.   249463.5  378389.5 3000000. ]
cats [    39.     3777.5    9676.    20631.75 519042.  ]
mean_ratio [0.00058181 0.01942229 0.03878724 0.05452517 0.173014  ]
--------------
headlight_right
image [  67032.   196300.5  250563.   386647.  3000000. ]
cats [    85.    3790.    9386.5  20452.  562128. ]
mean_ratio [0.00126805 0.01930713 0.03746164 0.05289579 0.187376  ]
```

(continues on next page)

(continued from previous page)

```
--------------
hood
image [  67032.   203663.5  266137.5  430700.  3000000. ]
cats [     25.     30013.5    52536.     89443.75 2128128.  ]
mean_ratio [0.00037296 0.14736809 0.19740172 0.20767065 0.709376  ]
--------------
mirror_left
image [  68142.    196824.    250272.    388976.25 3000000.  ]
cats [    99.     896.    1749.5   3775.5 656363. ]
mean_ratio [0.00145285 0.00455229 0.00699039 0.00970625 0.21878767]
--------------
mirror_right
image [  55944.  199615.  259008.  399696. 3000000.]
cats [     84.     900.    1792.    4085. 1566352.]
mean_ratio [0.0015015  0.00450868 0.00691871 0.01022027 0.52211733]
--------------
plate_back
image [  51273.   194658.   239592.   371353.5 3000000. ]
cats [   297.    4344.5   6844.   10707.  204820. ]
mean_ratio [0.00579252 0.02231863 0.02856523 0.02883237 0.06827333]
--------------
plate_front
image [  67032.    196750.75  251843.5   396708.75 3000000.  ]
cats [   259.   3420.   5925.  10030. 106113.]
mean_ratio [0.00386383 0.0173824  0.02352652 0.02528303 0.035371  ]
--------------
radiator_grill
image [  38781.  197685.  253000.  406638. 3000000.]
cats [   114.     5582.25  12333.5   25880.   658999.  ]
mean_ratio [0.00293958 0.02823811 0.04874901 0.06364383 0.21966633]
--------------
roof
image [  55944.    194882.25  248086.    388943.   3000000.  ]
cats [    192.    4554.    7099.   12211. 1313640.]
mean_ratio [0.003432   0.02336796 0.02861508 0.03139535 0.43788   ]
--------------
tail_light_left
image [  51273.    195515.25  242353.5   381713.25 3000000.  ]
cats [     84.    2168.    7209.   14876. 1091970.]
mean_ratio [0.00163829 0.01108865 0.02974581 0.03897166 0.36399   ]
--------------
tail_light_right
image [  51273.    195984.75  244357.5   387540.   3000000.  ]
cats [    52.    1928.5   7008.   15176.  431288. ]
mean_ratio [0.00101418 0.00984005 0.02867929 0.03915983 0.14376267]
--------------
wheels_back_left
image [  51273.    201319.75  252928.    387589.5  3000000.  ]
cats [    290.    4455.   11625.   27383. 1321811.]
mean_ratio [0.005656   0.02212898 0.0459617  0.07064949 0.44060367]
--------------
wheels_back_right
image [  51273.   203530.5  262065.5  406446.  3000000. ]
cats [    351.     4601.    12096.    26199.5 1427820. ]
mean_ratio [0.00684571 0.02260595 0.0461564  0.06445998 0.47594   ]
--------------
wheels_front_left
```

(continues on next page)

(continued from previous page)

```
image [  76302.   205331.5  261363.   408566.  3000000. ]
cats [   216.     4310.25  12355.5   27655.   939904.  ]
mean_ratio [0.00283086 0.02099166 0.04727333 0.06768796 0.31330133]
--------------
wheels_front_right
image [  67032.   204322.5  263361.5  405653.  3000000. ]
cats [    286.      4456.     12480.     27198.75 1235820.  ]
mean_ratio [0.00426662 0.02180866 0.04738734 0.0670493  0.41194   ]
--------------
windscreen
image [  67032.   201959.5  263344.   418435.  3000000. ]
cats [    462.     18855.25   28211.     46568.5  2441880.  ]
mean_ratio [0.00689223 0.09336154 0.10712604 0.11129208 0.81396   ]
--------------
trunk
image [  51273.    196836.75  243711.5   383033.25 3000000.  ]
cats [     84.    41890.    69419.   112526.5 2125236. ]
mean_ratio [0.00163829 0.21281595 0.28484089 0.29377737 0.708412  ]
--------------
```

```
{'bumper_back': 0.296,
 'bumper_front': 0.33,
 'door_front_left': 0.148,
 'door_front_right': 0.158,
 'door_pillar_left': 0.08,
 'door_pillar_right': 0.081,
 'door_rear_left': 0.159,
 'door_rear_right': 0.162,
 'door_window_front_left': 0.032,
 'door_window_front_right': 0.035,
 'door_window_rear_left': 0.031,
 'door_window_rear_right': 0.031,
 'fender_front_left': 0.117,
 'fender_front_right': 0.125,
 'handle_front_left': 0.001,
 'handle_front_right': 0.002,
 'handle_rear_left': 0.002,
 'handle_rear_right': 0.002,
 'headlight_left': 0.055,
 'headlight_right': 0.053,
 'hood': 0.208,
 'mirror_left': 0.01,
 'mirror_right': 0.01,
 'plate_back': 0.029,
 'plate_front': 0.025,
 'radiator_grill': 0.064,
 'roof': 0.031,
 'wheels_front_left': 0.068,
 'wheels_front_right': 0.067,
 'windscreen': 0.111,
 'back_side': 0.294,
 'window_back': 0.12,
 'sill_left': 0.068,
 'sill_right': 0.075,
 'fender_rear_left': 0.156,
 'fender_rear_right': 0.14,
```

(continues on next page)

(continued from previous page)

```
 'foglight_left': 0.005,
 'foglight_right': 0.005,
 'taillight_left': 0.039,
 'taillight_right': 0.039,
 'wheel_rear_left': 0.071,
 'wheel_rear_right': 0.064}
```

```
{'window_back': 1,
 'bumper_back': 2,
 'bumper_front': 3,
 'door_front_left': 4,
 'door_front_right': 5,
 'pillar_left': 6,
 'pillar_right': 7,
 'door_rear_left': 8,
 'door_rear_right': 9,
 'sill_left': 10,
 'sill_right': 11,
 'door_window_front_left': 12,
 'door_window_front_right': 13,
 'door_window_rear_left': 14,
 'door_window_rear_right': 15,
 'fender_rear_left': 16,
 'fender_rear_right': 17,
 'fender_front_left': 18,
 'fender_front_right': 19,
 'foglight_left': 20,
 'foglight_right': 21,
 'handle_front_left': 22,
 'handle_front_right': 23,
 'handle_rear_left': 24,
 'handle_rear_right': 25,
 'headlight_left': 26,
 'headlight_right': 27,
 'hood': 28,
 'mirror_left': 29,
 'mirror_right': 30,
 'plate_back': 31,
 'plate_front': 32,
 'radiator_grill': 33,
 'roof': 34,
 'taillight_left': 35,
 'taillight_right': 36,
 'wheel_rear_left': 37,
 'wheel_rear_right': 38,
 'wheels_front_left': 39,
 'wheels_front_right': 40,
 'windscreen': 41,
 'back_side': 42}
```

```
load categories: 100%|| 1/1 [00:00<00:00, 2432.89it/s]
load images: 100%|| 998/998 [00:00<00:00, 588819.16it/s]
load annotations: 100%|| 657/657 [00:00<00:00, 571239.16it/s]
reindex images: 998it [00:00, 768904.37it/s]
reindex annotations: 657it [00:00, 593508.02it/s]
load categories: 100%|| 1/1 [00:00<00:00, 3923.58it/s]
```

(continues on next page)

(continued from previous page)

```
load images: 100%|| 998/998 [00:00<00:00, 695912.78it/s]
load annotations: 100%|| 657/657 [00:00<00:00, 774931.87it/s]
reindex images: 998it [00:00, 478171.74it/s]
reindex annotations: 657it [00:00, 413886.71it/s]
```

```
reindex images: 998it [00:00, 630884.01it/s]
reindex annotations: 656it [00:00, 682575.89it/s]
reindex images: 998it [00:00, 300410.18it/s]
reindex annotations: 656it [00:00, 532506.95it/s]
```

```
100%|| 998/998 [00:01<00:00, 600.94it/s]
```

```
0.9984779299847792
```

```
0.9984779299847792
```

```
1.0
```

```
array([1, 2, 3])
```

```
'batch6__2017010400696400__foto0004.jpg'
```

```
/Users/fumarolaf/miniconda3/envs/an/lib/python3.7/site-packages/tqdm/
↪autonotebook/__init__.py:14: TqdmExperimentalWarning: Using tqdm.
↪autonotebook.tqdm in notebook mode. Use tqdm.tqdm instead to force console␣
↪mode (e.g. in jupyter console)
  " (e.g. in jupyter console)", TqdmExperimentalWarning)
```

```
HBox(children=(IntProgress(value=0, description='load images', max=118287,␣
↪style=ProgressStyle(description_wid...
```

```
HBox(children=(IntProgress(value=0, description='load annotations', max=860001,␣
↪style=ProgressStyle(descriptio...
```

```
{1: 'person', 2: 'bicycle', 3: 'car', 4: 'motorcycle', 5: 'airplane', 6: 'bus', 7:
↪'train', 8: 'truck', 9: 'boat', 10: 'traffic light', 11: 'fire hydrant', 13: 'stop␣
↪sign', 14: 'parking meter', 15: 'bench', 16: 'bird', 17: 'cat', 18: 'dog', 19:
↪'horse', 20: 'sheep', 21: 'cow', 22: 'elephant', 23: 'bear', 24: 'zebra', 25:
↪'giraffe', 27: 'backpack', 28: 'umbrella', 31: 'handbag', 32: 'tie', 33: 'suitcase',
↪ 34: 'frisbee', 35: 'skis', 36: 'snowboard', 37: 'sports ball', 38: 'kite', 39:
↪'baseball bat', 40: 'baseball glove', 41: 'skateboard', 42: 'surfboard', 43:
↪'tennis racket', 44: 'bottle', 46: 'wine glass', 47: 'cup', 48: 'fork', 49: 'knife',
↪ 50: 'spoon', 51: 'bowl', 52: 'banana', 53: 'apple', 54: 'sandwich', 55: 'orange',␣
↪56: 'broccoli', 57: 'carrot', 58: 'hot dog', 59: 'pizza', 60: 'donut', 61: 'cake',␣
↪62: 'chair', 63: 'couch', 64: 'potted plant', 65: 'bed', 67: 'dining table', 70:
↪'toilet', 72: 'tv', 73: 'laptop', 74: 'mouse', 75: 'remote', 76: 'keyboard', 77:
↪'cell phone', 78: 'microwave', 79: 'oven', 80: 'toaster', 81: 'sink', 82:
↪'refrigerator', 84: 'book', 85: 'clock', 86: 'vase', 87: 'scissors', 88: 'teddy bear
↪', 89: 'hair drier', 90: 'toothbrush'}
```

```
[('person', 64115), ('chair', 12774), ('car', 12251), ('dining table', 11837), ('cup',
↪ 9189), ('bottle', 8501), ('bowl', 7111), ('handbag', 6841), ('truck', 6127), (
↪'bench', 5570), ('backpack', 5528), ('book', 5332), ('cell phone', 4803), ('sink',␣
↪4678), ('clock', 4659), ('tv', 4561), ('potted plant', 4452), ('couch', 4123), ('dog
↪', 4385), ('knife', 4326), ('sports ball', 4262), ('traffic light', 4139), ('cat',␣
↪4114), ('umbrella', 3968), ('bus', 3952), ('tie', 3810), ('bed', 3682), ('vase',␣
↪3593), ('train', 3588), ('fork', 3555), ('spoon', 3529), ('laptop', 3524), (
↪'motorcycle', 3502), ('surfboard', 3486), ('skateboard', 3476), ('tennis racket',␣
↪3394), ('toilet', 3353), ('bicycle', 3252), ('bird', 3237), ('pizza', 3166), ('skis
↪', 3082), ('remote', 3076), ('boat', 3025), ('airplane', 2986), ('horse', 2941), (
```

(continues on next page)

(continued from previous page)

```
[('person', 262465), ('car', 43867), ('chair', 38491), ('book', 24715), ('bottle',␣
↪24342), ('cup', 20650), ('dining table', 15714), ('bowl', 14358), ('traffic light',␣
↪12884), ('handbag', 12354), ('umbrella', 11431), ('bird', 10806), ('boat', 10759), (
↪'truck', 9973), ('bench', 9838), ('sheep', 9509), ('banana', 9458), ('kite', 9076),␣
↪('motorcycle', 8725), ('backpack', 8720), ('potted plant', 8652), ('cow', 8147), (
↪'wine glass', 7913), ('carrot', 7852), ('knife', 7770), ('broccoli', 7308), ('donut
↪', 7179), ('bicycle', 7113), ('skis', 6646), ('vase', 6613), ('horse', 6587), ('tie
↪', 6496), ('cell phone', 6434), ('orange', 6399), ('cake', 6353), ('sports ball',␣
↪6347), ('clock', 6334), ('suitcase', 6192), ('spoon', 6165), ('surfboard', 6126), (
↪'bus', 6069), ('apple', 5851), ('pizza', 5821), ('tv', 5805), ('couch', 5779), (
↪'remote', 5703), ('sink', 5610), ('skateboard', 5543), ('elephant', 5513), ('dog',␣
↪5508), ('fork', 5479), ('zebra', 5303), ('airplane', 5135), ('giraffe', 5131), (
↪'laptop', 4970), ('tennis racket', 4812), ('teddy bear', 4793), ('cat', 4768), (
↪'train', 4571), ('sandwich', 4373), ('bed', 4192), ('toilet', 4157), ('baseball␣
↪glove', 3747), ('oven', 3334), ('baseball bat', 3276), ('hot dog', 2918), ('keyboard
↪', 2855), ('snowboard', 2685), ('frisbee', 2682), ('refrigerator', 2637), ('mouse',␣
↪2262), ('stop sign', 1983), ('toothbrush', 1954), ('fire hydrant', 1865), (
↪'microwave', 1673), ('scissors', 1481), ('bear', 1294), ('parking meter', 1285), (
↪'toaster', 225), ('hair drier', 198)]
```

```
{3, 4, 89}
```

```
{23, 80, 89}
```

```
dict_keys(['info', 'licenses', 'categories', 'annotations', 'images'])
```

```
images 1366
annotations 1717
```

```
HBox(children=(IntProgress(value=0, description='load images', max=1366,␣
↪style=ProgressStyle(description_width...
```

```
HBox(children=(IntProgress(value=0, description='load annotations', max=1717,␣
↪style=ProgressStyle(description_...
```

```
HBox(children=(IntProgress(value=0, description='download images', max=1366,␣
↪style=ProgressStyle(description_w...
```

```
removed 0 images
```

CHAPTER 8

Filter Coco Dataset

This notebook shows how to create a custom dataset starting from COCO>

```
/Users/fumarolaf/miniconda3/envs/an/lib/python3.7/site-packages/tqdm/
↪autonotebook/__init__.py:14: TqdmExperimentalWarning: Using tqdm.
↪autonotebook.tqdm in notebook mode. Use tqdm.tqdm instead to force console␣
↪mode (e.g. in jupyter console)
  " (e.g. in jupyter console)", TqdmExperimentalWarning)
```

Load the coco dataset. > download the annotations from [coco datasets](http://cocodataset.org/#download

```
HBox(children=(IntProgress(value=0, description='load images', max=118287,␣
↪style=ProgressStyle(description_wid...
```

```
HBox(children=(IntProgress(value=0, description='load annotations', max=860001,␣
↪style=ProgressStyle(descriptio...
```

```
{1: 'person', 2: 'bicycle', 3: 'car', 4: 'motorcycle', 5: 'airplane', 6: 'bus', 7:
↪'train', 8: 'truck', 9: 'boat', 10: 'traffic light', 11: 'fire hydrant', 13: 'stop␣
↪sign', 14: 'parking meter', 15: 'bench', 16: 'bird', 17: 'cat', 18: 'dog', 19:
↪'horse', 20: 'sheep', 21: 'cow', 22: 'elephant', 23: 'bear', 24: 'zebra', 25:
↪'giraffe', 27: 'backpack', 28: 'umbrella', 31: 'handbag', 32: 'tie', 33: 'suitcase',
↪ 34: 'frisbee', 35: 'skis', 36: 'snowboard', 37: 'sports ball', 38: 'kite', 39:
↪'baseball bat', 40: 'baseball glove', 41: 'skateboard', 42: 'surfboard', 43:
↪'tennis racket', 44: 'bottle', 46: 'wine glass', 47: 'cup', 48: 'fork', 49: 'knife',
↪ 50: 'spoon', 51: 'bowl', 52: 'banana', 53: 'apple', 54: 'sandwich', 55: 'orange',␣
↪56: 'broccoli', 57: 'carrot', 58: 'hot dog', 59: 'pizza', 60: 'donut', 61: 'cake',␣
↪62: 'chair', 63: 'couch', 64: 'potted plant', 65: 'bed', 67: 'dining table', 70:
↪'toilet', 72: 'tv', 73: 'laptop', 74: 'mouse', 75: 'remote', 76: 'keyboard', 77:
↪'cell phone', 78: 'microwave', 79: 'oven', 80: 'toaster', 81: 'sink', 82:
↪'refrigerator', 84: 'book', 85: 'clock', 86: 'vase', 87: 'scissors', 88: 'teddy bear
↪', 89: 'hair drier', 90: 'toothbrush'}
```

Show the number of images for category

```
[('person', 64115), ('chair', 12774), ('car', 12251), ('dining table', 11837), ('cup',
↪ 9189), ('bottle', 8501), ('bowl', 7111), ('handbag', 6841), ('truck', 6127), (
↪'bench', 5570), ('backpack', 5528), ('book', 5332), ('cell phone', 4803), ('sink',␣
↪4678), ('clock', 4659), ('tv', 4561), ('potted plant', 4452), ('couch', 4423), ('dog
↪', 4385), ('knife', 4326), ('sports ball', 4262), ('traffic light', 4139), ('cat',␣
↪4114), ('umbrella', 3968), ('bus', 3952), ('tie', 3810), ('bed', 3682), ('vase',␣
↪3593), ('train', 3588), ('fork', 3555), ('spoon', 3529), ('laptop', 3524), (
↪'motorcycle', 3502), ('surfboard', 3486), ('skateboard', 3476), ('tennis racket',␣
↪3394), ('toilet', 3353), ('bicycle', 3252), ('bird', 3237), ('pizza', 3166), ('skis
↪', 3082), ('remote', 3076), ('boat', 3025), ('airplane', 2986), ('horse', 2941), (
↪'cake', 2925), ('oven', 2877), ('baseball glove', 2629), ('giraffe', 2546), ('wine␣
↪glass', 2533), ('baseball bat', 2506), ('suitcase', 2402), ('sandwich', 2365), (
↪'refrigerator', 2360), ('kite', 2261), ('banana', 2243), ('frisbee', 2184), (
↪'elephant', 2143), ('teddy bear', 2140), ('keyboard', 2115), ('cow', 1968), (
↪'broccoli', 1939), ('zebra', 1916), ('mouse', 1876), ('stop sign', 1734), ('fire␣
↪hydrant', 1711), ('orange', 1699), ('carrot', 1683), ('snowboard', 1654), ('apple',␣
↪1586), ('microwave', 1547), ('sheep', 1529), ('donut', 1523), ('hot dog', 1222), (
↪'toothbrush', 1007), ('bear', 960), ('scissors', 947), ('parking meter', 705), (
↪'toaster', 217), ('hair drier', 189)]
```

Show the number of annotations per category

```
[('person', 262465), ('car', 43867), ('chair', 38491), ('book', 24715), ('bottle',␣
↪24342), ('cup', 20650), ('dining table', 15714), ('bowl', 14358), ('traffic light',␣
↪12884), ('handbag', 12354), ('umbrella', 11431), ('bird', 10806), ('boat', 10759), (
↪'truck', 9973), ('bench', 9838), ('sheep', 9509), ('banana', 9458), ('kite', 9076),␣
↪('motorcycle', 8725), ('backpack', 8720), ('potted plant', 8652), ('cow', 8147), (
↪'wine glass', 7913), ('carrot', 7852), ('knife', 7770), ('broccoli', 7308), ('donut
↪', 7179), ('bicycle', 7113), ('skis', 6646), ('vase', 6613), ('horse', 6587), ('tie
↪', 6496), ('cell phone', 6434), ('orange', 6399), ('cake', 6353), ('sports ball',␣
↪6347), ('clock', 6334), ('suitcase', 6192), ('spoon', 6165), ('surfboard', 6126), (
↪'bus', 6069), ('apple', 5851), ('pizza', 5821), ('tv', 5805), ('couch', 5779), (
↪'remote', 5703), ('sink', 5610), ('skateboard', 5543), ('elephant', 5513), ('dog',␣
↪5508), ('fork', 5479), ('zebra', 5303), ('airplane', 5135), ('giraffe', 5131), (
↪'laptop', 4970), ('tennis racket', 4812), ('teddy bear', 4793), ('cat', 4768), (
↪'train', 4571), ('sandwich', 4373), ('bed', 4192), ('toilet', 4157), ('baseball␣
↪glove', 3747), ('oven', 3334), ('baseball bat', 3276), ('hot dog', 2918), ('keyboard
↪', 2855), ('snowboard', 2685), ('frisbee', 2682), ('refrigerator', 2637), ('mouse',␣
↪2262), ('stop sign', 1983), ('toothbrush', 1954), ('fire hydrant', 1865), (
↪'microwave', 1673), ('scissors', 1481), ('bear', 1294), ('parking meter', 1285), (
↪'toaster', 225), ('hair drier', 198)]
```

8.1 Filter the Dataset

we create a daset with only 3 categories: - bear - toaster - hair drier

```
{23, 80, 89}
```

```
dict_keys(['info', 'licenses', 'categories', 'annotations', 'images'])
```

```
images 1366
annotations 1717
```

```
HBox(children=(IntProgress(value=0, description='load images', max=1366,␣
↪style=ProgressStyle(description_width...
```

```
HBox(children=(IntProgress(value=0, description='load annotations', max=1717,␣
↪style=ProgressStyle(description_...
```

We can download the images from the web

```
HBox(children=(IntProgress(value=0, description='download images', max=1366,␣
↪style=ProgressStyle(description_w...
```

```
removed 0 images
```

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index