volumentations Documentation

Release 0.1.7

Alexey Nekrasov

Aug 13, 2020

CONTENTS

Python library for 3d data augmentaiton. Hard fork from alumentations

CHAPTER

ONE

FEATURES

- Augmentations library for point clouds using minimum dependencies;
- Can work wth points and noramls;
- All features of albumentaitons like serializations and core components.

CHAPTER

TWO

PROJECT INFO

- GitHub repository: https://github.com/kumuji/volumentations
- License: MIT

CHAPTER

THREE

INSTALLATION

You can use pip to install volumentations:

```
pip install volumentations
```

If you want to get the latest version of the code before it is released on PyPI you can install the library from GitHub:

```
pip install -U git+https://github.com/kumuji/volumentations
```

CHAPTER

FOUR

DEMO

You can use this Google Colaboratory notebook to adjust image augmentation parameters and see the resulting images.

4.1 API

4.1.1 Core API (volumentations.core)

Composition

class volumentations.core.composition.**Compose**(*transforms*, *additional_targets=None*, *p=1.0*)

Compose transforms and handle all transformations regrading bounding boxes.

Parameters

- **transforms** (*list*) – list of transformations to compose.
- **additional_targets** (*dict*) – Dict with keys - new target name, values - old target name. ex: {'image2': 'image'}
- **p** (*float*) – probability of applying all list of transforms. Default: 1.0.

class volumentations.core.composition.**OneOf**(*transforms*, *p=0.5*)

Select one of transforms to apply.

Parameters

- **transforms** (*list*) – list of transformations to compose.
- **p** (*float*) – probability of applying selected transform. Default: 0.5.

class volumentations.core.composition.**ReplayCompose**(*transforms*, *additional_targets=None*, *p=1.0*, *save_key='replay'*)

Transforms interface

volumentations.core.transforms_interface.**to_tuple**(*param*, *low=None*, *bias=None*)
Convert input argument to min-max tuple

Parameters

- **param** (*scalar, tuple or list of 2+ elements*) – Input value. If value is scalar, return value would be (offset - value, offset + value). If value is tuple, return value would be value + offset (broadcasted).
- **low** – Second element of tuple can be passed as optional argument
- **bias** – An offset factor added to each element

class volumentations.core.transforms_interface.**PointCloudsTransform**(*always_apply=False*, *p=0.5*)
Transform for point clouds.

class volumentations.core.transforms_interface.**NoOp**(*always_apply=False*, *p=0.5*)
Does nothing

Serialization

volumentations.core.serialization.**to_dict**(*transform*, *on_not_implemented_error='raise'*)
Take a transform pipeline and convert it to a serializable representation that uses only standard python data types: dictionaries, lists, strings, integers, and floats.

Parameters **transform** (*object*) – A transform that should be serialized. If the transform doesn't implement the *to_dict* method and *on_not_implemented_error* equals to 'raise' then *NotImplementedError* is raised. If *on_not_implemented_error* equals to 'warn' then *NotImplementedError* will be ignored but no transform parameters will be serialized.

volumentations.core.serialization.**from_dict**(*transform_dict*, *lambda_transforms=None*)

Parameters

- **transform** (*dict*) – A dictionary with serialized transform pipeline.
- **lambda_transforms** (*dict*) – A dictionary that contains lambda transforms, that is instances of the Lambda class. This dictionary is required when you are restoring a pipeline that contains lambda transforms. Keys in that dictionary should be named same as *name* arguments in respective lambda transforms from a serialized pipeline.

volumentations.core.serialization.**save**(*transform*, *filepath*, *data_format='json'*, *on_not_implemented_error='raise'*)
Take a transform pipeline, serialize it and save a serialized version to a file using either json or yaml format.

Parameters

- **transform** (*obj*) – Transform to serialize.
- **filepath** (*str*) – Filepath to write to.
- **data_format** (*str*) – Serialization format. Should be either *json* or 'yaml'.
- **on_not_implemented_error** (*str*) – Parameter that describes what to do if a transform doesn't implement the *to_dict* method. If 'raise' then *NotImplementedError* is raised, if *warn* then the exception will be ignored and no transform arguments will be saved.

volumentations.core.serialization.**load**(*filepath*, *data_format='json'*, *lambda_transforms=None*)
Load a serialized pipeline from a json or yaml file and construct a transform pipeline.

Parameters

- **transform** (*obj*) – Transform to serialize.
- **filepath** (*str*) – Filepath to read from.
- **data_format** (*str*) – Serialization format. Should be either *json* or 'yaml'.
- **lambda_transforms** (*dict*) – A dictionary that contains lambda transforms, that is instances of the Lambda class. This dictionary is required when you are restoring a pipeline that contains lambda transforms. Keys in that dictionary should be named same as *name* arguments in respective lambda transforms from a serialized pipeline.

4.1.2 Augmentations (volumentations.augmentations)

Transforms

class volumentations.augmentations.transforms.**Scale3d**(*scale_limit=(0.1, 0.1, 0.1)*, *bias=(1, 1, 1)*, *always_apply=False*, *p=0.5*)

Scale the input point cloud.

Parameters

- **scale_limit** (*float, float, float*) – maximum scaling of input point cloud. Default: (0.1, 0.1, 0.1).
- **bias** (*list(float, float, float)*) – base scaling that is always applied. List of 3 values to determine the basic scaling. Default: (1, 1, 1).
- **p** (*float*) – probability of applying the transform. Default: 0.5.

Targets: points normals features labels

class volumentations.augmentations.transforms.**RotateAroundAxis3d**(*rotation_limit=1.5707963267948966*, *axis=(0, 0, 1)*, *always_apply=False*, *p=0.5*)

Rotate point cloud around axis on random angle.

Parameters

- **rotation_limit** (*float*) – maximum rotation of the input point cloud. Default: (pi / 2).
- **axis** (*list(float, float, float)*) – axis around which the point cloud is rotated. Default: (0, 0, 1).
- **p** (*float*) – probability of applying the transform. Default: 0.5.

Targets: points normals features labels

class volumentations.augmentations.transforms.**Crop3d**(*x_min=-inf*, *y_min=-inf*, *z_min=-inf*, *x_max=inf*, *y_max=inf*, *z_max=inf*, *always_apply=False*, *p=1.0*)

Crop region from image.

Parameters

- **x_min** (*float*) – Minimum x coordinate.
- **y_min** (*float*) – Minimum y coordinate.
- **z_min** (*float*) – Minimum z coordinate.
- **x_max** (*float*) – Maximum x coordinate.
- **y_max** (*float*) – Maximum y coordinate.
- **z_max** (*float*) – Maximum z coordinate.
- **p** (*float*) – probability of applying the transform. Default: 0.5.

Targets: points normals features labels

class volumentations.augmentations.transforms.**RandomMove3d**(*x_min=-1.0*, *y_min=-1.0*, *z_min=-1.0*, *x_max=1.0*, *y_max=1.0*, *z_max=1.0*, *offset=(0, 0, 0)*, *always_apply=False*, *p=0.5*)

Move point cloud on random offset.

Parameters

- **x_min** (*float*) – Minimum x coordinate. Default: -1.
- **y_min** (*float*) – Minimum y coordinate. Default: -1.
- **z_min** (*float*) – Minimum z coordinate. Default: -1.
- **x_max** (*float*) – Maximum x coordinate. Default: 1.
- **y_max** (*float*) – Maximum y coordinate. Default: 1.
- **z_max** (*float*) – Maximum z coordinate. Default: 1.
- **p** (*float*) – probability of applying the transform. Default: 0.5.

Targets: points normals features labels

class volumentations.augmentations.transforms.**Move3d**(*offset=(0, 0, 0)*, *always_apply=False*, *p=1.0*)

Move point cloud on offset.

Parameters

- **offset** (*float*) – coorinate where to move origin of coordinate frame. Default: 0.
- **p** (*float*) – probability of applying the transform. Default: 0.5.

Targets: points normals features labels

class volumentations.augmentations.transforms.**Center3d**(*offset=(0, 0, 0)*, *always_apply=False*, *p=0.5*)

Move average of point cloud and move it to coordinate (0,0,0).

Parameters **p** (*float*) – probability of applying the transform. Default: 0.5.

Targets: points normals features labels

class volumentations.augmentations.transforms.**RandomDropout3d**(*dropout_ratio=0.2*, *always_apply=False*, *p=0.5*)

Randomly drop points from point cloud.

Parameters

- **dropout_ratio** (*float*) – Percent of points to drop. Default: 0.2.
- **p** (*float*) – probability of applying the transform. Default: 0.5.

Targets: points normals features labels

class volumentations.augmentations.transforms.**Flip3d**(*axis=(0, 0, 1)*, *always_apply=False*, *p=0.5*)

Flip point cloud around axis Implemented as rotation on 180 deg around axis.

Parameters

- **axis** (*list(float, float, float)*) – Axis to flip the point cloud around. Default: 0.2.
- **p** (*float*) – probability of applying the transform. Default: 0.5.

Targets: points normals features labels

Functionals

volumentations.augmentations.functional.**rotate_around_axis**(*points*, *axis*, *angle*)

Return the rotation matrix associated with counterclockwise rotation about the given axis by angle in radians. https://stackoverflow.com/questions/6802577/rotation-of-3d-vector

4.2 Examples

```
import volumentations as V
import numpy as np

volume_aug = V.Compose(
    [
        V.Scale3d(scale_limit=[0.1, 0.1, 0.1], bias=[1, 1, 1]),
        V.RotateAroundAxis3d(axis=[0, 0, 1], rotation_limit=np.pi / 6),
        V.RotateAroundAxis3d(axis=[0, 1, 0], rotation_limit=np.pi / 6),
        V.RotateAroundAxis3d(axis=[1, 0, 0], rotation_limit=np.pi / 6),
        V.RandomDropout3d(dropout_ratio=0.2),
    ]
)
original_point_cloud = np.empty((1000, 3))
augmented_point_cloud = volume_aug(points=original_point_cloud)["points"]
```

For more examples see repository with examples

4.3 Contributing

All development is done on GitHub: https://github.com/kumuji/volumentations

If you find a bug or have a feature request file an issue at https://github.com/kumuji/volumentations/issues

To create a pull request:

1. Fork the repository.
2. Clone it.
3. Install development requirements:

```
python -m pip install poetry
poetry install --dev
```

4. Initialize it from the folder with the repo:

```
pre-commit install
```

4. Make desired changes to the code.
5. Check if your changes passing the tests:

```
nox -rs
```

7. Push code to your forked repo.
8. Create pull request.

PYTHON MODULE INDEX

V

INDEX