TorchSat

Sep 24, 2019

Contents:

TorchSat is an open-source deep learning framework for satellite imagery analysis based on PyTorch.

This project is still **work in progress**. If you want to know more about it, please refer to the Roadmap .

Hightlight

- Support multi-channels(> 3 channels, e.g. 8 channels) images and TIFF file as input.
- Convenient data augmentation method for classification, sementic segmentation and object detection.
- Lots of models for satellite vision tasks, such as ResNet, DenseNet, UNet, PSPNet, SSD, FasterRCNN ...
- Lots of common satellite datasets loader.
- Training script for common satellite vision tasks.

CHAPTER 1

Installtation

1.1 Deepndencies

TorchSat is based on PyTorch, so you should install the PyTorch first. And if you want to use the GPU version, you should install CUDA.

Python package dependencies(seee also requirements.txt): pytorch, torchvision, numpy, pillow, tifffile, six, scipy, opencv

> Note: TorchSat only suport Python 3. We recommend version after Python 3.5(including python 3.5), but wo have not tested any version below Python 3.5

1.2 Install

1.2.1 Install from PyPI or Anconda

- PyPI: `pip3 install torchsat`
- Anconda: ‘‘ ‘‘

1.2.2 Install from source

- Install the latest version

```
git clone https://github.com/sshuair/torchsat.git
cd torchsat
python3 setup.py install
```

- Install the stable version
  1. Visit the release page and download the version you want.

2. Decompress the zip or tar file.
3. Enter the torchsat directory and run this command `python3 setup.py install`.

1.3 For data preparation

[wip]

1.4 Docker

You can pull the docker image from Docker Hub if you want use TorchSat in docker.

1. **pull image**
    - cpu: *docker pull sshuair/torchsat:cpu-latest*
    - gpu: *docker pull sshuair/torchsat:gpu-latest*
2. **run container**
    - cpu: *docker run -ti –name <NAME> sshuair/torchsat:cpu-latest bash*
    - gpu: *docker run -ti –gpu 0,1 –name <NAME> sshuair/torchsat:gpu-latest bash*

This way you can easily use the TorchSat in docker container.

CHAPTER 2

Core Conceptions

Because TorchSat is based on PyTorch, you'd better have some deep learning and PyTorch knowledge to use and modify this project.

Here are some core conceptions you should know.

1. All input image data, Whether it is PNG, JPEG or GeoTIFF, will be converted to NumPy ndarray, and the ndarray dimension is `[height, width]` (single channel image) or `[height, width, channels]` (multi-channel image).

2. After the data is read into NumPy ndarray, there are only three data types, *np.uint8*, *np.uint16*, *np.float*. These three data types are chosen because, in satellite imagery,

- The most common image stored in JPEG or PNG (such as the Google Map Satellite Image) is mostly *8-bit (np.uint8)* data;
- And the *16-bit (np.uint16)* is mostly the original data type of remote sensing satellite imagery, which is also very common;
- The third type *float (np.float)* is considered, because sometimes, we will use remote sensing index (such as NDVI) as features to participate in training. For this data we suppose all input data values range from 0 to 1.

CHAPTER 3

Quickstart

CHAPTER 4

Tutorials

CHAPTER 5

API Reference

5.1 torchsat package

5.1.1 Subpackages

torchsat.datasets package

Submodules

torchsat.datasets.eurosat module

class torchsat.datasets.eurosat.**EuroSAT**(*root*, *mode='RGB'*, *download=False*, ***kwargs*)

Bases: *torchsat.datasets.folder.DatasetFolder*

download()

url_allband = 'http://madm.dfki.de/files/sentinel/EuroSATallBands.zip'

url_rgb = 'http://madm.dfki.de/files/sentinel/EuroSAT.zip'

torchsat.datasets.folder module

class torchsat.datasets.folder.**DatasetFolder**(*root*, *loader*, *extensions*, *classes=None*, *class_to_idx=None*, *transform=None*, *target_transform=None*)

Bases: torch.utils.data.dataset.Dataset

A generic data loader where the samples are arranged in this way:

```
root/class_x/xxx.ext
root/class_x/xxy.ext
root/class_x/xxz.ext
```

(continues on next page)

(continued from previous page)

```
root/class_y/123.ext
root/class_y/nsdf3.ext
root/class_y/asd932_.ext
```

Args:

root (string): Root directory path. loader (callable): A function to load a sample given its path. extensions (list[string]): A list of allowed extensions. classes (callable, optional): List of the class names. class_to_idx (callable, optional): Dict with items (class_name, class_index). transform (callable, optional): A function/transform that takes in

a sample and returns a transformed version. E.g, `transforms.RandomCrop` for images.

target_transform (callable, optional): A function/transform that takes in the target and transforms it.

Attributes: classes (list): List of the class names. class_to_idx (dict): Dict with items (class_name, class_index). samples (list): List of (sample path, class_index) tuples targets (list): The class_index value for each image in the dataset

class torchsat.datasets.folder.**ImageFolder**(*root*, *transform=None*, *target_transform=None*, *loader=<function default_loader>*)

Bases: *torchsat.datasets.folder.DatasetFolder*

A generic data loader where the images are arranged in this way:

```
root/dog/xxx.png
root/dog/xxy.png
root/dog/xxz.png

root/cat/123.png
root/cat/nsdf3.png
root/cat/asd932_.png
```

Args:

root (string): Root directory path. transform (callable, optional): A function/transform that takes in an PIL image

and returns a transformed version. E.g, `transforms.RandomCrop`

target_transform (callable, optional): A function/transform that takes in the target and transforms it.

loader (callable, optional): A function to load an image given its path.

Attributes: classes (list): List of the class names. class_to_idx (dict): Dict with items (class_name, class_index). imgs (list): List of (image path, class_index) tuples

torchsat.datasets.folder.**has_file_allowed_extension**(*filename*, *extensions*)

Checks if a file is an allowed extension.

Args: filename (string): path to a file extensions (iterable of strings): extensions to consider (lowercase)

Returns: bool: True if the filename ends with one of given extensions

`torchsat.datasets.folder.`**`is_image_file`**(*filename*)
Checks if a file is an allowed image extension.

Args: filename (string): path to a file

Returns: bool: True if the filename ends with a known image extension

`torchsat.datasets.folder.`**`make_dataset`**(*dir*, *class_to_idx*, *extensions*)

torchsat.datasets.nwpu_resisc45 module

class `torchsat.datasets.nwpu_resisc45.`**`NWPU_RESISC45`**(*root*, *download=False*, ***kwargs*)
Bases: *torchsat.datasets.folder.DatasetFolder*

`download`()

`url = 'https://sov8mq.dm.files.1drv.com/y4m_Fo6ujI52LiWHDzaRZVtkMIZxF7aqjX2q7KdVr329zV`

torchsat.datasets.patternnet module

class `torchsat.datasets.patternnet.`**`PatternNet`**(*root*, *download=False*, ***kwargs*)
Bases: *torchsat.datasets.folder.DatasetFolder*

`download`()

`url = 'https://doc-0k-9c-docs.googleusercontent.com/docs/securesc/s4mst7k8sdlkn5gslv2v`

torchsat.datasets.sat module

class `torchsat.datasets.sat.`**`SAT`**(*root*, *mode='SAT-4'*, *image_set='train'*, *download=False*, *transform=False*, *target_transform=False*)
Bases: `torch.utils.data.dataset.Dataset`

SAT-4 and SAT-6 datasets

Arguments: data {root} – [description]

Raises: ValueError – [description] ValueError – [description]

Returns: [type] – [description]

`classes_sat4 = {'barren land': 0, 'grassland': 2, 'none': 3, 'trees': 1}`

`classes_sat6 = {'barren land': 1, 'building': 0, 'grassland': 3, 'road': 4, 'trees`

`download`()

torchsat.datasets.utils module

`torchsat.datasets.utils.`**`accimage_loader`**(*path*)

`torchsat.datasets.utils.`**`default_loader`**(*path*)

`torchsat.datasets.utils.`**`download_url`**(*url*, *root*, *filename=None*, *md5=None*)
Download a file from a url and place it in root. Args:

url (str): URL to download file from root (str): Directory to place downloaded file in filename (str, optional): Name to save the file under. If None, use the basename of the URL md5 (str, optional): MD5 checksum of the download. If None, do not check

torchsat.datasets.utils.**gen_bar_updater**()

torchsat.datasets.utils.**pil_loader**(*path*)

torchsat.datasets.utils.**tifffile_loader**(*path*)

Module contents

torchsat.models package

Subpackages

torchsat.models.classification package

Submodules

torchsat.models.classification.densenet module

class torchsat.models.classification.densenet.**DenseNet**(*growth_rate=32*, *block_config=(6, 12, 24, 16)*, *num_init_features=64*, *bn_size=4*, *drop_rate=0*, *num_classes=1000*, *in_channels=3*, *memory_efficient=False*)

Bases: torch.nn.modules.module.Module

Densenet-BC model class, based on "Densely Connected Convolutional Networks" Args:

growth_rate (int) - how many filters to add each layer (*k* in paper) block_config (list of 4 ints) - how many layers in each pooling block num_init_features (int) - the number of filters to learn in the first convolution layer bn_size (int) - multiplicative factor for number of bottle neck layers

(i.e. bn_size * k features in the bottleneck layer)

drop_rate (float) - dropout rate after each dense layer num_classes (int) - number of classification classes memory_efficient (bool) - If True, uses checkpointing. Much more memory efficient,

but slower. Default: *False*. See "paper"

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

torchsat.models.classification.densenet.**densenet121**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

Densenet-121 model from "Densely Connected Convolutional Networks" Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr memory_efficient (bool) - If True, uses checkpointing. Much more memory efficient,

but slower. Default: *False*. See "paper"

torchsat.models.classification.densenet.**densenet169**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

Densenet-169 model from "Densely Connected Convolutional Networks" Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr memory_efficient (bool) - If True, uses checkpointing. Much more memory efficient,

but slower. Default: *False*. See "paper"

torchsat.models.classification.densenet.**densenet201**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

Densenet-201 model from "Densely Connected Convolutional Networks" Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr memory_efficient (bool) - If True, uses checkpointing. Much more memory efficient,

but slower. Default: *False*. See "paper"

torchsat.models.classification.densenet.**densenet161**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

Densenet-161 model from "Densely Connected Convolutional Networks" Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr memory_efficient (bool) - If True, uses checkpointing. Much more memory efficient,

but slower. Default: *False*. See "paper"

torchsat.models.classification.inception module

class torchsat.models.classification.inception.**Inception3**(*num_classes=1000*, *in_channels=3*, *aux_logits=True*, *transform_input=False*)

Bases: torch.nn.modules.module.Module

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks

while the latter silently ignores them.

---

torchsat.models.classification.inception.**inception_v3**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

Inception v3 model architecture from "Rethinking the Inception Architecture for Computer Vision". .. note:

```
**Important**: In contrast to the other models the inception_v3 expects tensors␣
↪with a size of
N x 3 x 299 x 299, so ensure your images are sized accordingly.
```

Args: pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr aux_logits (bool): If True, add an auxiliary branch that can improve training.

Default: *True*

transform_input (bool): If True, preprocesses the input according to the method with which it was trained on ImageNet. Default: *False*

torchsat.models.classification.mobilenet module

class torchsat.models.classification.mobilenet.**MobileNetV2**(*num_classes=1000*, *in_channels=3*, *width_mult=1.0*, *inverted_residual_setting=None*, *round_nearest=8*)

Bases: torch.nn.modules.module.Module

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

torchsat.models.classification.mobilenet.**mobilenet_v2**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

Constructs a MobileNetV2 architecture from "MobileNetV2: Inverted Residuals and Linear Bottlenecks". Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.resnet module

class torchsat.models.classification.resnet.**ResNet**(*block*, *layers*, *num_classes=1000*, *in_channels=3*, *zero_init_residual=False*, *groups=1*, *width_per_group=64*, *replace_stride_with_dilation=None*, *norm_layer=None*)

Bases: torch.nn.modules.module.Module

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

torchsat.models.classification.resnet.**resnet18**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

ResNet-18 model from ‘”Deep Residual Learning for Image Recognition” <https://arxiv.org/pdf/1512.03385.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.resnet.**resnet34**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

ResNet-34 model from ‘”Deep Residual Learning for Image Recognition” <https://arxiv.org/pdf/1512.03385.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.resnet.**resnet50**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

ResNet-50 model from ‘”Deep Residual Learning for Image Recognition” <https://arxiv.org/pdf/1512.03385.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.resnet.**resnet101**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

ResNet-101 model from ‘”Deep Residual Learning for Image Recognition” <https://arxiv.org/pdf/1512.03385.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.resnet.**resnet152**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

ResNet-152 model from ‘”Deep Residual Learning for Image Recognition” <https://arxiv.org/pdf/1512.03385.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.resnet.**resnext50_32x4d**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

ResNeXt-50 32x4d model from “Aggregated Residual Transformation for Deep Neural Networks” Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.resnet.**resnext101_32x8d**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

ResNeXt-101 32x8d model from “Aggregated Residual Transformation for Deep Neural Networks” Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.resnet.**wide_resnet50_2**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

Wide ResNet-50-2 model from “Wide Residual Networks” The model is the same as ResNet except for the bottleneck number of channels which is twice larger in every block. The number of channels in outer 1x1 convolutions is the same, e.g. last block in ResNet-50 has 2048-512-2048 channels, and in Wide ResNet-50-2 has 2048-1024-2048. Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.resnet.**wide_resnet101_2**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

Wide ResNet-101-2 model from “Wide Residual Networks” The model is the same as ResNet except for the bottleneck number of channels which is twice larger in every block. The number of channels in outer 1x1 convolutions is the same, e.g. last block in ResNet-50 has 2048-512-2048 channels, and in Wide ResNet-50-2 has 2048-1024-2048. Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.vgg module

class torchsat.models.classification.vgg.**VGG**(*features*, *num_classes=1000*, *init_weights=True*)

Bases: torch.nn.modules.module.Module

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks

while the latter silently ignores them.

---

torchsat.models.classification.vgg.**vgg11**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

VGG 11-layer model (configuration “A”) from ‘”Very Deep Convolutional Networks For Large-Scale Image Recognition” <https://arxiv.org/pdf/1409.1556.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.vgg.**vgg11_bn**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

VGG 11-layer model (configuration “A”) with batch normalization ‘”Very Deep Convolutional Networks For Large-Scale Image Recognition” <https://arxiv.org/pdf/1409.1556.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.vgg.**vgg13**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

VGG 13-layer model (configuration “B”) ‘”Very Deep Convolutional Networks For Large-Scale Image Recognition” <https://arxiv.org/pdf/1409.1556.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.vgg.**vgg13_bn**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

VGG 13-layer model (configuration “B”) with batch normalization ‘”Very Deep Convolutional Networks For Large-Scale Image Recognition” <https://arxiv.org/pdf/1409.1556.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.vgg.**vgg16**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

VGG 16-layer model (configuration “D”) ‘”Very Deep Convolutional Networks For Large-Scale Image Recognition” <https://arxiv.org/pdf/1409.1556.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.vgg.**vgg16_bn**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

VGG 16-layer model (configuration “D”) with batch normalization ‘”Very Deep Convolutional Networks For Large-Scale Image Recognition” <https://arxiv.org/pdf/1409.1556.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.vgg.**vgg19_bn**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

VGG 19-layer model (configuration ‘E’) with batch normalization ‘”Very Deep Convolutional Networks For Large-Scale Image Recognition” <https://arxiv.org/pdf/1409.1556.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

torchsat.models.classification.vgg.**vgg19**(*num_classes*, *in_channels=3*, *pretrained=False*, *progress=True*, ***kwargs*)

VGG 19-layer model (configuration “E”) ‘”Very Deep Convolutional Networks For Large-Scale Image Recognition” <https://arxiv.org/pdf/1409.1556.pdf>’_ Args:

pretrained (bool): If True, returns a model pre-trained on ImageNet progress (bool): If True, displays a progress bar of the download to stderr

Module contents

torchsat.models.detection package

Submodules

torchsat.models.detection.ssd module

Module contents

torchsat.models.segmentation package

Submodules

torchsat.models.segmentation.pspnet module

torchsat.models.segmentation.unet module

class torchsat.models.segmentation.unet.**UNetResNet**(*encoder_depth*, *num_classes*, *in_channels=3*, *num_filters=32*, *dropout_2d=0.0*, *pretrained=False*, *is_deconv=False*)

Bases: torch.nn.modules.module.Module

PyTorch U-Net model using ResNet(34, 101 or 152) encoder. UNet: https://arxiv.org/abs/1505.04597 ResNet: https://arxiv.org/abs/1512.03385 Proposed by Alexander Buslaev: https://www.linkedin.com/in/al-buslaev/

Args: encoder_depth (int): Depth of a ResNet encoder (34, 101 or 152). num_classes (int): Number of output classes. num_filters (int, optional): Number of filters in the last layer of decoder. Defaults to 32. dropout_2d (float, optional): Probability factor of dropout layer before output layer. Defaults to 0.2. pretrained (bool, optional):

False - no pre-trained weights are being used. True - ResNet encoder is pre-trained on ImageNet. Defaults to False.

is_deconv (bool, optional): False: bilinear interpolation is used in decoder. True: deconvolution is used in decoder. Defaults to False.

Raises: ValueError: [description] NotImplementedError: [description]

Returns: [type]: [description]

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks

while the latter silently ignores them.

torchsat.models.segmentation.unet.**unet34**(*num_classes*, *in_channels=3*, *pretrained=False*, ***kwargs*)

torchsat.models.segmentation.unet.**unet101**(*num_classes*, *in_channels=3*, *pretrained=False*, ***kwargs*)

torchsat.models.segmentation.unet.**unet152**(*num_classes*, *in_channels=3*, *pretrained=False*, ***kwargs*)

Module contents

Submodules

torchsat.models.utils module

Module contents

torchsat.transforms package

Submodules

torchsat.transforms.functional module

torchsat.transforms.functional.**adjust_brightness**(*img*, *value=0*)

torchsat.transforms.functional.**adjust_contrast**(*img*, *factor*)

torchsat.transforms.functional.**adjust_hue**()

torchsat.transforms.functional.**adjust_saturation**()

torchsat.transforms.functional.**bbox_crop**(*bboxes*, *top*, *left*, *height*, *width*)

crop bbox

Arguments: img {ndarray} – image to be croped top {int} – top size left {int} – left size height {int} – croped height width {int} – croped width

torchsat.transforms.functional.**bbox_hflip**(*bboxes*, *img_width*)

horizontal flip the bboxes ^

.....

^

Args: bbox (ndarray): bbox ndarray [box_nums, 4] flip_code (int, optional): [description]. Defaults to 0.

torchsat.transforms.functional.**bbox_pad**(*bboxes*, *padding*)

torchsat.transforms.functional.**bbox_resize**(*bboxes*, *img_size*, *target_size*)

resize the bbox

Args: bboxes (ndarray): bbox ndarray [box_nums, 4] img_size (tuple): the image height and width target_size (int, or tuple): the target bbox size.

Int or Tuple, if tuple the shape should be (height, width)

torchsat.transforms.functional.**bbox_shift**(*bboxes*, *top*, *left*)

torchsat.transforms.functional.**bbox_vflip**(*bboxes*, *img_height*)

vertical flip the bboxes ....

>.....< ..... Args:

bbox (ndarray): bbox ndarray [box_nums, 4] flip_code (int, optional): [description]. Defaults to 0.

torchsat.transforms.functional.**center_crop**(*img*, *output_size*)

crop image

Arguments: img {ndarray} – input image output_size {number or sequence} – the output image size. if sequence, should be [h, w]

Raises: ValueError – the input image is large than original image.

Returns: ndarray image – return croped ndarray image.

torchsat.transforms.functional.**crop**(*img*, *top*, *left*, *height*, *width*)

crop image

Arguments: img {ndarray} – image to be croped top {int} – top size left {int} – left size height {int} – croped height width {int} – croped width

torchsat.transforms.functional.**elastic_transform**(*image*, *alpha*, *sigma*, *alpha_affine*, *interpolation=1*, *border_mode=4*, *random_state=None*, *approximate=False*)

Elastic deformation of images as described in [Simard2003] (with modifications). Based on https://gist.github.com/erniejunior/601cdf56d2b424757de5 .. [Simard2003] Simard, Steinkraus and Platt, "Best Practices for

Convolutional Neural Networks applied to Visual Document Analysis", in Proc. of the International Conference on Document Analysis and Recognition, 2003.

torchsat.transforms.functional.**flip**(*img*, *flip_code*)

torchsat.transforms.functional.**gaussian_blur**(*img*, *kernel_size*)

torchsat.transforms.functional.**hflip**(*img*)

torchsat.transforms.functional.**noise**(*img*, *mode='gaussain'*, *percent=0.02*)

TODO: Not good for uint16 data

torchsat.transforms.functional.**normalize**(*tensor*, *mean*, *std*, *inplace=False*)

Normalize a tensor image with mean and standard deviation.

Note: This transform acts out of place by default, i.e., it does not mutates the input tensor.

See `Normalize` for more details.

Args: tensor (Tensor): Tensor image of size (C, H, W) to be normalized. mean (sequence): Sequence of means for each channel. std (sequence): Sequence of standard deviations for each channel.

Returns: Tensor: Normalized Tensor image.

torchsat.transforms.functional.**pad**(*img*, *padding*, *fill=0*, *padding_mode='constant'*)

torchsat.transforms.functional.**preserve_channel_dim**(*func*)

Preserve dummy channel dim.

torchsat.transforms.functional.**resize**(*img*, *size*, *interpolation=2*)
resize the image TODO: opencv resize 0~1 Arguments:

img {ndarray} – the input ndarray image size {int, iterable} – the target size, if size is intger, width and height will be resized to same otherwise, the size should be tuple (height, width) or list [height, width]

Keyword Arguments: interpolation {Image} – the interpolation method (default: {Image.BILINEAR})

Raises: TypeError – img should be ndarray ValueError – size should be intger or iterable vaiable and length should be 2.

Returns: img – resize ndarray image

torchsat.transforms.functional.**resized_crop**(*img*, *top*, *left*, *height*, *width*, *size*, *interpolation=2*)

torchsat.transforms.functional.**rotate**(*img*, *angle*, *center=None*, *scale=1.0*)

torchsat.transforms.functional.**shift**(*img*, *top*, *left*)

torchsat.transforms.functional.**to_grayscale**(*img*, *output_channels=1*)
convert input ndarray image to gray sacle image.

Arguments: img {ndarray} – the input ndarray image

Keyword Arguments: output_channels {int} – output gray image channel (default: {1})

Returns: ndarray – gray scale ndarray image

torchsat.transforms.functional.**to_pil_image**(*tensor*)

torchsat.transforms.functional.**to_tensor**(*img*)
convert numpy.ndarray to torch tensor.

if the image is uint8 , it will be divided by 255;

if the image is uint16 , it will be divided by 65535;

if the image is float , it will not be divided, we suppose your image range should between [0~1] ;

Arguments: img {numpy.ndarray} – image to be converted to tensor.

torchsat.transforms.functional.**to_tiff_image**(*tensor*)

torchsat.transforms.functional.**vflip**(*img*)

torchsat.transforms.transforms_cls module

class torchsat.transforms.transforms_cls.**Compose**(*transforms*)
Bases: object

Composes serveral classification transform together.

Args: transforms (list of `transform` objects): list of classification transforms to compose.

Example:

```
>>> transforms_cls.Compose([
>>>     transforms_cls.Resize(300),
>>>     transforms_cls.ToTensor()
>>>     ])
```

class torchsat.transforms.transforms_cls.**Lambda**(*lambd*)
Bases: object

Apply a user-defined lambda as function.

Args: lambd (function): Lambda/function to be used for transform.

class torchsat.transforms.transforms_cls.**ToTensor**
Bases: object

onvert numpy.ndarray to torch tensor.

if the image is uint8 , it will be divided by 255; if the image is uint16 , it will be divided by 65535; if the image is float , it will not be divided, we suppose your image range should between [0~1] ;

Args: img {numpy.ndarray} – image to be converted to tensor.

class torchsat.transforms.transforms_cls.**Normalize**(*mean*, *std*, *inplace=False*)
Bases: object

Normalize a tensor image with mean and standard deviation.

Given mean: `(M1,...,Mn)` and std: `(S1,..,Sn)` for `n` channels, this transform will normalize each channel of the input `torch.*Tensor` i.e. `input[channel] = (input[channel] - mean[channel]) / std[channel]` .. note:

```
This transform acts out of place, i.e., it does not mutates the input tensor.
```

Args: tensor (tensor): input torch tensor data. mean (sequence): Sequence of means for each channel. std (sequence): Sequence of standard deviations for each channel. inplace (boolean): inplace apply the transform or not. (default: False)

class torchsat.transforms.transforms_cls.**ToGray**(*output_channels=1*)
Bases: object

Convert the image to grayscale

Args: output_channels (int): number of channels desired for output image. (default: 1)

Returns: [ndarray]: the graysacle version of input - If output_channels=1 : returned single channel image (height, width) - If output_channels>1 : returned multi-channels ndarray image (height, width, channels)

class torchsat.transforms.transforms_cls.**GaussianBlur**(*kernel_size=3*)
Bases: object

Convert the input ndarray image to blurred image by gaussian method.

Args: kernel_size (int): kernel size of gaussian blur method. (default: 3)

Returns: ndarray: the blurred image.

class torchsat.transforms.transforms_cls.**RandomNoise**(*mode='gaussian'*, *percent=0.02*)
Bases: object

Add noise to the input ndarray image. Args:

mode (str): the noise mode, should be one of `gaussian`, `salt`, `pepper`, `s&p`, (default: gaussian).
percent (float): noise percent, only work for `salt`, `pepper`, `s&p` mode. (default: 0.02)

Returns: ndarray: noised ndarray image.

class torchsat.transforms.transforms_cls.**RandomBrightness**(*max_value=0*)
Bases: object

class torchsat.transforms.transforms_cls.**RandomContrast**(*max_factor=0*)
Bases: object

class torchsat.transforms.transforms_cls.**RandomShift**(*max_percent=0.4*)
Bases: object

random shift the ndarray with value or some percent.

Args: max_percent (float): shift percent of the image.

Returns: ndarray: return the shifted ndarray image.

class torchsat.transforms.transforms_cls.**RandomRotation**(*degrees*, *center=None*)
Bases: object

random rotate the ndarray image with the degrees.

Args:

degrees (number or sequence): the rotate degree. If single number, it must be positive. if squeence, it's length must 2 and first number should small than the second one.

Raises: ValueError: If degrees is a single number, it must be positive. ValueError: If degrees is a sequence, it must be of len 2.

Returns: ndarray: return rotated ndarray image.

class torchsat.transforms.transforms_cls.**Resize**(*size*, *interpolation=2*)
Bases: object

resize the image Args:

img {ndarray} : the input ndarray image size {int, iterable} : the target size, if size is intger, width and height will be resized to same otherwise, the size should be tuple (height, width) or list [height, width]

Keyword Arguments: interpolation {Image} : the interpolation method (default: {Image.BILINEAR})

Raises: TypeError : img should be ndarray ValueError : size should be intger or iterable vaiable and length should be 2.

Returns: img (ndarray) : resize ndarray image

class torchsat.transforms.transforms_cls.**Pad**(*padding*, *fill=0*, *padding_mode='constant'*)
Bases: object

Pad the given ndarray image with padding width. Args:

padding [{int, sequence}, padding width] If int, each border same. If sequence length is 2, this is the padding for left/right and top/bottom. If sequence length is 4, this is the padding for left, top, right, bottom.

fill: {int, sequence}: Pixel padding_mode: str or function. contain{'constant','edge','linear_ramp','maximum','mean'

, 'median', 'minimum', 'reflect','symmetric','wrap'} (default: constant)

Examples:

```
>>> transformed_img = Pad(img, 20, mode='reflect')
>>> transformed_img = Pad(img, (10,20), mode='edge')
>>> transformed_img = Pad(img, (10,20,30,40), mode='reflect')
```

class torchsat.transforms.transforms_cls.**CenterCrop**(*out_size*)
Bases: object

crop image

Args: img {ndarray}: input image output_size {number or sequence}: the output image size. if sequence, should be [height, width]

Raises: ValueError: the input image is large than original image.

Returns: ndarray: return croped ndarray image.

class torchsat.transforms.transforms_cls.**RandomCrop**(*size*)
Bases: object

random crop the input ndarray image

Args: size (int, sequence): th output image size, if sequeue size should be [height, width]

Returns: ndarray: return random croped ndarray image.

class torchsat.transforms.transforms_cls.**RandomHorizontalFlip**(*p=0.5*)
Bases: object

Flip the input image on central horizon line.

Args: p (float): probability apply the horizon flip.(default: 0.5)

Returns: ndarray: return the flipped image.

class torchsat.transforms.transforms_cls.**RandomVerticalFlip**(*p=0.5*)
Bases: object

Flip the input image on central vertical line.

Args: p (float): probability apply the vertical flip. (default: 0.5)

Returns: ndarray: return the flipped image.

class torchsat.transforms.transforms_cls.**RandomFlip**(*p=0.5*)
Bases: object

Flip the input image vertical or horizon.

Args: p (float): probability apply flip. (default: 0.5)

Returns: ndarray: return the flipped image.

class torchsat.transforms.transforms_cls.**RandomResizedCrop**(*crop_size*, *target_size*, *interpolation=2*)
Bases: object

[summary]

Args: object ([type]): [description]

Returns: [type]: [description]

class torchsat.transforms.transforms_cls.**ElasticTransform**(*alpha=1*, *sigma=50*, *alpha_affine=50*, *interpolation=1*, *border_mode=4*, *random_state=None*, *approximate=False*)

Bases: object

code modify from https://github.com/albu/albumentations. Elastic deformation of images as described in [Simard2003] (with modifications). Based on https://gist.github.com/erniejunior/601cdf56d2b424757de5 .. [Simard2003] Simard, Steinkraus and Platt, "Best Practices for

> Convolutional Neural Networks applied to Visual Document Analysis", in Proc. of the International Conference on Document Analysis and Recognition, 2003.

Args:

approximate (boolean): Whether to smooth displacement map with fixed kernel size. Enabling this option gives ~2X speedup on large images.

Image types: uint8, uint16 float32

torchsat.transforms.transforms_det module

class torchsat.transforms.transforms_det.**Compose**(*transforms*)

Bases: object

Composes serveral classification transform together.

Args: transforms (list of `transform` objects): list of classification transforms to compose.

Example:

```
>>> transforms_cls.Compose([
>>>     transforms_cls.Resize(300),
>>>     transforms_cls.ToTensor()
>>>     ])
```

class torchsat.transforms.transforms_det.**Lambda**(*lambd*)

Bases: object

Apply a user-defined lambda as function.

Args: lambd (function): Lambda/function to be used for transform.

class torchsat.transforms.transforms_det.**ToTensor**

Bases: object

onvert numpy.ndarray to torch tensor.

> if the image is uint8 , it will be divided by 255; if the image is uint16 , it will be divided by 65535; if the image is float , it will not be divided, we suppose your image range should between [0~1] ;

Args: img {numpy.ndarray} – image to be converted to tensor. bboxes {numpy.ndarray} – target bbox to be converted to tensor. the input should be [box_nums, 4] labels {numpy.ndarray} – target labels to be converted to tensor. the input shape shold be [box_nums]

class torchsat.transforms.transforms_det.**Normalize**(*mean*, *std*, *inplace=False*)

Bases: object

Normalize a tensor image with mean and standard deviation.

Given mean: `(M1,...,Mn)` and std: `(S1,..,Sn)` for `n` channels, this transform will normalize each channel of the input `torch.*Tensor` i.e. `input[channel] = (input[channel] - mean[channel]) / std[channel]` .. note:

```
This transform acts out of place, i.e., it does not mutates the input tensor.
```

Args: tensor (tensor): input torch tensor data. mean (sequence): Sequence of means for each channel. std (sequence): Sequence of standard deviations for each channel. inplace (boolean): inplace apply the transform or not. (default: False)

class torchsat.transforms.transforms_det.**ToGray**(*output_channels=1*)
Bases: object

Convert the image to grayscale

Args: output_channels (int): number of channels desired for output image. (default: 1)

Returns: [ndarray]: the graysacle version of input - If output_channels=1 : returned single channel image (height, width) - If output_channels>1 : returned multi-channels ndarray image (height, width, channels)

class torchsat.transforms.transforms_det.**GaussianBlur**(*kernel_size=3*)
Bases: object

Convert the input ndarray image to blurred image by gaussian method.

Args: kernel_size (int): kernel size of gaussian blur method. (default: 3)

Returns: ndarray: the blurred image.

class torchsat.transforms.transforms_det.**RandomNoise**(*mode='gaussian'*, *percent=0.02*)
Bases: object

Add noise to the input ndarray image. Args:

mode (str): the noise mode, should be one of `gaussian`, `salt`, `pepper`, `s&p`, (default: gaussian).
percent (float): noise percent, only work for `salt`, `pepper`, `s&p` mode. (default: 0.02)

Returns: ndarray: noised ndarray image.

class torchsat.transforms.transforms_det.**RandomBrightness**(*max_value=0*)
Bases: object

class torchsat.transforms.transforms_det.**RandomContrast**(*max_factor=0*)
Bases: object

class torchsat.transforms.transforms_det.**Resize**(*size*, *interpolation=2*)
Bases: object

resize the image Args:

img {ndarray} : the input ndarray image size {int, iterable} : the target size, if size is intger, width and height will be resized to same otherwise, the size should be tuple (height, width) or list [height, width]

Keyword Arguments: interpolation {Image} : the interpolation method (default: {Image.BILINEAR})

Raises: TypeError : img should be ndarray ValueError : size should be intger or iterable vaiable and length should be 2.

Returns: img (ndarray) : resize ndarray image

class torchsat.transforms.transforms_det.**Pad**(*padding*, *fill=0*, *padding_mode='constant'*)

Bases: object

Pad the given ndarray image with padding width. Args:

padding [{int, sequence}, padding width] If int, each border same. If sequence length is 2, this is the padding for left/right and top/bottom. If sequence length is 4, this is the padding for left, top, right, bottom.

fill: {int, sequence}: Pixel padding_mode: str or function. contain{'constant','edge','linear_ramp','maximum','mean'

, 'median', 'minimum', 'reflect','symmetric','wrap'} (default: constant)

Examples:

```
>>> transformed_img = Pad(img, 20, mode='reflect')
>>> transformed_img = Pad(img, (10,20), mode='edge')
>>> transformed_img = Pad(img, (10,20,30,40), mode='reflect')
```

class torchsat.transforms.transforms_det.**CenterCrop**(*out_size*)

Bases: object

crop image

Args: img {ndarray}: input image output_size {number or sequence}: the output image size. if sequence, should be [height, width]

Raises: ValueError: the input image is large than original image.

Returns: ndarray: return croped ndarray image.

class torchsat.transforms.transforms_det.**RandomCrop**(*size*)

Bases: object

random crop the input ndarray image

Args: size (int, sequence): th output image size, if sequeue size should be [height, width]

Returns: ndarray: return random croped ndarray image.

class torchsat.transforms.transforms_det.**RandomHorizontalFlip**(*p=0.5*)

Bases: object

Flip the input image on central horizon line.

Args: p (float): probability apply the horizon flip.(default: 0.5)

Returns: ndarray: return the flipped image.

class torchsat.transforms.transforms_det.**RandomVerticalFlip**(*p=0.5*)

Bases: object

Flip the input image on central vertical line.

Args: p (float): probability apply the vertical flip. (default: 0.5)

Returns: ndarray: return the flipped image.

class torchsat.transforms.transforms_det.**RandomFlip**(*p=0.5*)
Bases: object

Flip the input image vertical or horizon.

Args: p (float): probability apply flip. (default: 0.5)

Returns: ndarray: return the flipped image.

class torchsat.transforms.transforms_det.**RandomResizedCrop**(*crop_size*, *target_size*, *interpolation=2*)
Bases: object

[summary]

Args: object ([type]): [description]

Returns: [type]: [description]

torchsat.transforms.transforms_seg module

class torchsat.transforms.transforms_seg.**Compose**(*transforms*)
Bases: object

class torchsat.transforms.transforms_seg.**Lambda**(*lambd*)
Bases: object

class torchsat.transforms.transforms_seg.**ToTensor**
Bases: object

class torchsat.transforms.transforms_seg.**Normalize**(*mean*, *std*, *inplace=False*)
Bases: object

class torchsat.transforms.transforms_seg.**ToGray**(*output_channels=1*)
Bases: object

Convert the image to grayscale

Args: output_channels (int): number of channels desired for output image. (default: 1)

Returns: [ndarray]: the graysacle version of input - If output_channels=1 : returned single channel image (height, width) - If output_channels>1 : returned multi-channels ndarray image (height, width, channels)

class torchsat.transforms.transforms_seg.**GaussianBlur**(*kernel_size=3*)
Bases: object

class torchsat.transforms.transforms_seg.**RandomNoise**(*mode='gaussian'*, *percent=0.02*)
Bases: object

Add noise to the input ndarray image. Args:

mode (str): the noise mode, should be one of `gaussian`, `salt`, `pepper`, `s&p`, (default: gaussian).
percent (float): noise percent, only work for `salt`, `pepper`, `s&p` mode. (default: 0.02)

Returns: ndarray: noised ndarray image.

class torchsat.transforms.transforms_seg.**RandomBrightness**(*max_value=0*)
Bases: object

class torchsat.transforms.transforms_seg.**RandomContrast**(*max_factor=0*)
Bases: object

class torchsat.transforms.transforms_seg.**RandomShift**(*max_percent=0.4*)
Bases: object

random shift the ndarray with value or some percent.

Args: max_percent (float): shift percent of the image.

Returns: ndarray: return the shifted ndarray image.

class torchsat.transforms.transforms_seg.**RandomRotation**(*degrees*, *center=None*)
Bases: object

random rotate the ndarray image with the degrees.

Args:

degrees (number or sequence): the rotate degree. If single number, it must be positive. if squeence, it's length must 2 and first number should small than the second one.

Raises: ValueError: If degrees is a single number, it must be positive. ValueError: If degrees is a sequence, it must be of len 2.

Returns: ndarray: return rotated ndarray image.

class torchsat.transforms.transforms_seg.**Resize**(*size*, *interpolation=2*)
Bases: object

resize the image Args:

img {ndarray} : the input ndarray image size {int, iterable} : the target size, if size is intger, width and height will be resized to same otherwise, the size should be tuple (height, width) or list [height, width]

Keyword Arguments: interpolation {Image} : the interpolation method (default: {Image.BILINEAR})

Raises: TypeError : img should be ndarray ValueError : size should be intger or iterable vaiable and length should be 2.

Returns: img (ndarray) : resize ndarray image

class torchsat.transforms.transforms_seg.**Pad**(*padding*, *fill=0*, *padding_mode='constant'*)
Bases: object

Pad the given ndarray image with padding width. Args:

padding [{int, sequence}, padding width] If int, each border same. If sequence length is 2, this is the padding for left/right and top/bottom. If sequence length is 4, this is the padding for left, top, right, bottom.

fill: {int, sequence}: Pixel padding_mode: str or function. contain{'constant','edge','linear_ramp','maximum','mean'

, 'median', 'minimum', 'reflect','symmetric','wrap'} (default: constant)

Examples:

```
>>> transformed_img = Pad(img, 20, mode='reflect')
>>> transformed_img = Pad(img, (10,20), mode='edge')
>>> transformed_img = Pad(img, (10,20,30,40), mode='reflect')
```

class torchsat.transforms.transforms_seg.**CenterCrop**(*out_size*)
Bases: object

crop image

Args: img {ndarray}: input image output_size {number or sequence}: the output image size. if sequence, should be [height, width]

Raises: ValueError: the input image is large than original image.

Returns: ndarray: return croped ndarray image.

class torchsat.transforms.transforms_seg.**RandomCrop**(*size*)
Bases: object

random crop the input ndarray image

Args: size (int, sequence): th output image size, if sequeue size should be [height, width]

Returns: ndarray: return random croped ndarray image.

class torchsat.transforms.transforms_seg.**RandomHorizontalFlip**(*p=0.5*)
Bases: object

Flip the input image on central horizon line.

Args: p (float): probability apply the horizon flip.(default: 0.5)

Returns: ndarray: return the flipped image.

class torchsat.transforms.transforms_seg.**RandomVerticalFlip**(*p=0.5*)
Bases: object

Flip the input image on central vertical line.

Args: p (float): probability apply the vertical flip. (default: 0.5)

Returns: ndarray: return the flipped image.

class torchsat.transforms.transforms_seg.**RandomFlip**(*p=0.5*)
Bases: object

Flip the input image vertical or horizon.

Args: p (float): probability apply flip. (default: 0.5)

Returns: ndarray: return the flipped image.

class torchsat.transforms.transforms_seg.**RandomResizedCrop**(*crop_size*, *target_size*, *interpolation=2*)
Bases: object

[summary]

Args: object ([type]): [description]

Returns: [type]: [description]

class torchsat.transforms.transforms_seg.**ElasticTransform**(*alpha=1*, *sigma=50*, *alpha_affine=50*, *interpolation=1*, *border_mode=4*, *random_state=None*, *approximate=False*)
Bases: object

code modify from https://github.com/albu/albumentations. Elastic deformation of images as described in [Simard2003] (with modifications). Based on https://gist.github.com/erniejunior/601cdf56d2b424757de5 .. [Simard2003] Simard, Steinkraus and Platt, "Best Practices for

> Convolutional Neural Networks applied to Visual Document Analysis", in Proc. of the International Conference on Document Analysis and Recognition, 2003.

Args:

approximate (boolean): Whether to smooth displacement map with fixed kernel size. Enabling this option gives ~2X speedup on large images.

Image types: uint8, uint16 float32

Module contents

torchsat.utils package

Submodules

torchsat.utils.metrics module

torchsat.utils.visualizer module

Module contents

5.1.2 Module contents

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

F

G

H

I

L

M

N

P

R

S

T

U

V

W