## Contents:

1 Installation 3  
   1.1 Dependencies ................................................................. 3  
   1.2 Install ........................................................................ 3  

2 Core Conceptions 5  

3 Image Classification 7  
   3.1 Prepare training data ............................................................... 7  
   3.2 Training model .................................................................. 9  
   3.3 Inference new image ................................................................. 10  

4 Semantic Segmentation 11  
   4.1 Prepare training data ............................................................... 11  
   4.2 Training model .................................................................. 12  
   4.3 Inference new image ................................................................. 13  

5 Object Detection 15  

6 Change Detection 17  
   6.1 Prepare the training data ............................................................... 17  
   6.2 train a model .................................................................. 18  
   6.3 We provide the following models: ................................................. 18  

7 Data Augmentation 19  
   7.1 Image classification ................................................................. 19  
   7.2 Object detection .................................................................. 23  
   7.3 Semantic segmentation ............................................................... 26  

8 Useful Tools 29  
   8.1 processing tools .................................................................. 29  
   8.2 train scripts .................................................................. 31  

9 API Reference 33  
   9.1 torchsat package .................................................................. 33  

10 Indices and tables 57  

Python Module Index 59
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.

**Highlight**

- Support multi-channels (> 3 channels, e.g. 8 channels) images and TIFF file as input.
- Convenient data augmentation method for classification, semantic 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

Installation

1.1 Dependencies

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 (see also requirements.txt): pytorch, torchvision, numpy, pillow, tifffile, six, scipy, opencv

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

1.2 Install

1.2.1 Install from PyPI

• PyPI: pip3 install torchsat

1.2.2 Install from source

• Install the latest version

```bash
  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`.

### 1.2.3 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.
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, \texttt{np.uint8}, \texttt{np.uint16}, \texttt{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 (\texttt{np.uint8}) data;
   - And the 16-bit (\texttt{np.uint16}) is mostly the original data type of remote sensing satellite imagery, which is also very common;
   - The third type float (\texttt{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.
This is classification tutorial for satellite image. We will use sentinel-2 TCI data as an example. It cover from training data prepare, train the model, and predict the new files.

### 3.1 Prepare training data

Suppose we got a scene of sentinel-2 satellite TCI image data. You can download from esa scihub. I has got the scene id T51RTQ_20200513T023551_TCI and convert the JPEG2000 to GeoTIFF.

1. patch the large 10980x10980 pixel image to 128x128 pixel image

```
cd tests/classification/
ts make-mask-cls --filepath T51RTQ_20200513T023551_TCI.tif --width 128 --height 128 --outpath ./patched
processing 1/1 file T51RTQ_20200513T023551_TCI.tif ...
14% | 12/85 [00:07] <-00:45, 1.60it/s]
```

You should get the following data:
1. labeling the train data and test data

You can split the data into train data and test data as below. And then labeling those patched image into four classes: water, residential, farmland, forest. Reorganize the catalog of these small images according to different categories and split them to train and validation dataset.

```
<table>
<thead>
<tr>
<th>train</th>
<th>validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>water</td>
<td>water</td>
</tr>
<tr>
<td>T51RTQ_20200513T023551_TCI_1_29.tif</td>
<td></td>
</tr>
<tr>
<td>T51RTQ_20200513T023551_TCI_2_29.tif</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>farmland</td>
<td>farmland</td>
</tr>
<tr>
<td>T51RTQ_20200513T023551_TCI_3_2.tif</td>
<td></td>
</tr>
<tr>
<td>T51RTQ_20200513T023551_TCI_11_1.tif</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>residential</td>
<td>residential</td>
</tr>
<tr>
<td>T51RTQ_20200513T023551_TCI_0_29.tif</td>
<td></td>
</tr>
<tr>
<td>T51RTQ_20200513T023551_TCI_1_37.tif</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>forest</td>
<td>forest</td>
</tr>
<tr>
<td>T51RTQ_20200513T023551_TCI_7_21.tif</td>
<td></td>
</tr>
<tr>
<td>T51RTQ_20200513T023551_TCI_22_45.tif</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```
3.2 Training model

You can use the scripts/train_cls.py to train your classification model. Here is a train examples, for more parameter information, please use python3 train_cls.py help to view.

```
python3 train_cls.py --train-path ./classification/train \  
--val-path ./tests/classification/val \  
--model resnet18 \  
--batch-size 16 \  
--num-classes 4 \  
--device cuda
```

It should start train and the console will print the training logs.

```
Train Epoch: 0 [0/1514 (0%)]  Loss: 1.878198
Train Epoch: 0 [160/1514 (11%)]  Loss: 0.81105
Train Epoch: 0 [320/1514 (21%)]  Loss: 0.774963
Train Epoch: 0 [480/1514 (32%)]  Loss: 0.810051
Train Epoch: 0 [640/1514 (42%)]  Loss: 0.869388
Train Epoch: 0 [800/1514 (53%)]  Loss: 4.763704
Train Epoch: 0 [960/1514 (63%)]  Loss: 0.968885
Train Epoch: 0 [1120/1514 (74%)]  Loss: 4.856205
Train Epoch: 0 [1280/1514 (84%)]  Loss: 1.36379
Train Epoch: 0 [1440/1514 (95%)]  Loss: 0.551179

Test set: Average loss: 16.4018, Accuracy: 68/326 (21%)

Train Epoch: 1 [0/1514 (0%)]  Loss: 2.768502
Train Epoch: 1 [160/1514 (11%)]  Loss: 0.424574
Train Epoch: 1 [320/1514 (21%)]  Loss: 0.572497
Train Epoch: 1 [480/1514 (32%)]  Loss: 1.539173
Train Epoch: 1 [640/1514 (42%)]  Loss: 0.707925
Train Epoch: 1 [800/1514 (53%)]  Loss: 0.545577
Train Epoch: 1 [960/1514 (63%)]  Loss: 0.956915
Train Epoch: 1 [1120/1514 (74%)]  Loss: 0.556552
Train Epoch: 1 [1280/1514 (84%)]  Loss: 0.825140
Train Epoch: 1 [1440/1514 (95%)]  Loss: 0.588212

Test set: Average loss: 0.4656, Accuracy: 254/326 (78%)

Train Epoch: 2 [0/1514 (0%)]  Loss: 0.422114
Train Epoch: 2 [160/1514 (11%)]  Loss: 0.273431
Train Epoch: 2 [320/1514 (21%)]  Loss: 0.505005
....
```

You can also open Tensorboard to view the loss and test accuracy during training.
And we provide lots of classification model, and all of them support multi-channel(>4) tiff image as input.

- **VGG**: vgg11, vgg11_bn, vgg13, vgg13_bn, vgg16, vgg16_bn, vgg19, vgg19_bn, vgg19
- **ResNet**: resnet18, resnet34, resnet50, resnet101, resnet152, resnext50_32x4d,'resnext101_32x8d,wide_resnet50_2,wide_resnet101_2
- **DenseNet**: densenet121, densenet169, densenet201
- **Inception**: inception_v3
- **MobileNet**: mobilenet_v2
- **EfficientNet**: efficientnet_b0, efficientnet_b1, efficientnet_b2, efficientnet_b3,'efficientnet_b4, efficientnet_b5, efficientnet_b6, efficientnet_b7
- **ResNeSt**: resnest50, resnest101, resnest200, resnest269

### 3.3 Inference new image

[TODO]
This tutorial will teach you how to use torchsat to train your semantic segmentation model for your satellite project.

4.1 Prepare training data

If your training data is a large image (such as 10000 x 10000 pixels), you can use torchsat command-line tool to crop it into small pictures for training. There are two cases:

1. If your sample labels are vectors, then you can use ts make_mask_seg to directly generate small samples that can be directly used for training.

2. If your sample label is a picture, then you can use ts make_mask_cls to cut the large satellite image and label image separately.

The final generated train data needs to be organized as follows:

```
train
  image
    train_1.jpg
    train_2.jpg
    ...
  label
    train_1.png
    train_2.png
    ...

val
  image
    val_10.jpg
    val_11.jpg
    ...
  label
    val_10.png
```
4.2 Training model

You can use scripts/train_seg.py to train your model. You can also modify some of them according to your own needs, such as data augmentation, loss function, optimizer, model.

```
usage: test.py [-h] [--train-path TRAIN_PATH] [--val-path VAL_PATH]
               [--extensions EXTENSIONS [EXTENSIONS ...]] [--model MODEL]
               [--pretrained PRETRAINED] [--resume PATH]
               [--num-classes NUM_CLASSES] [--in-channels IN_CHANNELS]
               [--device DEVICE] [-b BATCH_SIZE] [--epochs EPOCHS] [--lr LR]
               [--print-freq PRINT_FREQ] [--ckp-dir CKP_DIR]

TorchSat Segmentation Training Script

optional arguments:
  -h, --help            show this help message and exit
  --train-path TRAIN_PATH
                        train dataset path
  --val-path VAL_PATH   validate dataset path
  --extensions EXTENSIONS [EXTENSIONS ...]
                        the train image extension
  --model MODEL         model name. default, unet34
  --pretrained PRETRAINED
                        use ImageNet pretrained params
  --resume PATH         path to latest checkpoint (default: none)
  --num-classes NUM_CLASSES
                        num of classes
  --in-channels IN_CHANNELS
                        input image channels
  --device DEVICE
  -b BATCH_SIZE, --batch-size BATCH_SIZE
                        batch size
  --epochs EPOCHS
  --lr LR
  --print-freq PRINT_FREQ
                        print frequency
  --ckp-dir CKP_DIR     path to save checkpoint
```

You can also open Tensorboard to view the loss and test accuracy during training.
Currently supported models are:

- **UNet**
  - ResNet Backbone: `unet_restnet18`, `unet_resnet34`, `unet_resnet101`

### 4.3 Inference new image

[TODO]
CHAPTER 5

Object Detection

[TODO]
This is tutorial for satellite change detection.

### 6.1 Prepare the training data

The satellite change detection dataset should be organized by the following struct.

```plaintext
train
  pre
    train_1.png
    train_2.png
    ...
  post
    train_1.png
    train_2.png
    ...
  label
    train_1.png
    train_2.png
    ...

val
  pre
    val_10.png
    val_11.png
    ...
  post
    val_10.png
    val_11.png
    ...
  label
    val_10.png
    ...
```
You can use the `make_mask_cls` generate the pre and post image.

6.2 train a model

6.3 We provide the following models:
This notebook shows the data augmentation for classification, detection, and segmentation.

```python
# matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import tifffile
from torchsat.transforms import transforms_cls, transforms_det, transforms_seg
from torchsat.utils.visualizer import plot_img, plot_bbox, plot_mask
```

### 7.1 Image classification

#### 1. 3 channel image

```python
fp = '../tests/fixtures/different-types/jpeg_3channel_uint8.jpeg'
img = np.array(Image.open(fp))
plot_img(img)
```

<Figure size 720x576 with 0 Axes>
# apply the classification transform
result = transforms_cls.Compose([  
    transforms_cls.GaussianBlur(kernel_size=9),  
    transforms_cls.RandomNoise(),  
    transforms_cls.RandomHorizontalFlip(p=1),  
    transforms_cls.RandomShift(max_percent=0.1),  
    transforms_cls.Resize(400)  
])
plot_img(result)
2. 8 channel uint16 tiff image

```python
fp = '../tests/fixtures/different-types/tiff_8channel_uint16.tif'
img = tifffile.imread(fp)
print('image shape: '.format(img.shape))
plot_img(img, channels=(8,3,1))
```

image shape:

<Figure size 720x576 with 0 Axes>

7.1. Image classification
```python
# apply the classification transform
result = transforms_cls.Compose([
    transforms_cls.GaussianBlur(kernel_size=9),
    transforms_cls.RandomVerticalFlip(p=1),
    transforms_cls.RandomShift(max_percent=0.1),
    transforms_cls.Resize(400),
    transforms_cls.RandomRotation(30),
    # transforms_cls.ElasticTransform()
])(img)
plot_img(result, channels=(8,3,1))
```

Figure size 720x576 with 0 Axes
7.2 Object detection

```
fp_img = '../tests/fixtures/different-types/jpeg_3channel_uint8.jpeg'
img = np.array(Image.open(fp_img))
bboxes = np.array([
    [210, 314, 225, 335],
    [273, 324, 299, 337],
    [275, 378, 302, 391],
    [213, 506, 228, 527],
    [241, 534, 257, 556],
    [31, 627, 49, 640],
    [99, 213, 120, 229],
    [61, 1, 164, 155],
    [1, 59, 44, 138],
    [61, 249, 165, 342],
])
```

(continues on next page)
labels = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3]
classes = ['car', 'building', 'pool']

plot_bbox(img, bboxes, labels=labels, classes=classes)
# apply the object detection transform

```python
result_img, result_bboxes, result_labels = transforms_det.Compose(
    [transfers_det.GaussianBlur(kernel_size=3),
     transfers_det.RandomFlip(p=1),
    })(img, bboxes, labels)
plot_bbox(result_img, result_bboxes, labels=result_labels, classes=classes)
```
7.3 Semantic segmentation

```python
fp_img = '../tests/fixtures/different-types/jpeg_3channel_uint8.jpeg'
fp_mask = '../tests/fixtures/masks/mask_tiff_3channel_uint8.png'
img = np.array(Image.open(fp_img))
mask = np.array(Image.open(fp_mask))
plot_mask(img, mask)
```
# apply the classification transform
result_img, result_mask = transforms_seg.Compose([
    transforms_seg.GaussianBlur(kernel_size=9),
    transforms_seg.RandomVerticalFlip(p=1),
    transforms_seg.RandomShift(max_percent=0.1),
    transforms_seg.RandomRotation(30),
    transforms_seg.Resize((500,500)),
#    transforms_seg.ElasticTransform()
])(img, mask)
plot_mask(result_img, result_mask)

7.3. Semantic segmentation
CHAPTER 8

Useful Tools

Collections of useful tools for satellite deep learning processing.

8.1 processing tools

8.1.1 make_mask_cls

This tool can split the large satellite image (e.g., 10000 x 10000 pixel) to small chips (e.g., 512 x 512 pixel). Both the satellite image and label image can use this tool.

Usage: ts make_mask_cls [OPTIONS]

this tool is to patch the large satellite image to small image.

Options:

|--filepath TEXT  the target satellite image to split. Note: the file should have crs
|--width INTEGER  the width of the patched image, default 512
|--height INTEGER the height of the patched image, default 512
|--drop_last BOOLEAN set to True to drop the last column and row, if the image size is not divisible by the height and width. default True.
|--outpath TEXT the output file path
|--colormap BOOLEAN weather write the colormap to the output patched image. only work for single channel image. default False.
|--extensions TEXT the train image extension, only work for dirctory file path.
|--help Show this message and exit.

example:
8.1.2 make_mask_seg

This tool is to patch the large satellite image to small image and label for segmentation. The input should be projected satellite image and vector file e.g., geojson shpfile.

Usage: ts make_mask_seg [OPTIONS]

this tool **is** to patch the large satellite image to small image **and** label for segmentation.

Options:
- --image_file TEXT the target satellite image to split. Note: the file should have crs
- --label_file TEXT the corresponding label file of the satellite image. vector or raster file. Note the crs should be same as satellite image.
- --field TEXT field to burn
- --width INTEGER the width of the patched image
- --height INTEGER the height of the patched image
- --drop_last BOOLEAN set to True to drop the last column and row, if the image size is not divisible by the height and width.
- --outpath TEXT the output file path
- --help Show this message and exit.

8.1.3 calcuate_mean_std

This tool is for calculating each channel mean and std value of the datasets.

Usage: ts calcuate_mean_std [OPTIONS]

calculuate the datasets mean **and** std value

Options:
- --root PATH root dir of image datasets [required]
- --percent FLOAT percent of images to calculate
- --channels INTEGER datasets image channels
- --maxvalue FLOAT max value of all images default: {255}
- --extension TEXT file suffix to calculate, default ('jpg', 'jpeg', 'png', 'tif', 'tiff')
- --help Show this message and exit.

**example:**

ts calcuate_mean_std --root /tests/classification/val/
100%|| 163/163 [00:02<00:00, 70.53it/s]
scaled mean:[0.36473823 0.40924644 0.41250621]

(continues on next page)
8.2 train scripts

The script can be used for training the model. You can modify the script according to your own project. It is independent of torchsat, you can get it from the scripts directory from torchsat repo.

8.2.1 scripts/train_cls.py


TorchSat Classification Training

optional arguments:
  -h, --help            show this help message and exit
  --train-path TRAIN_PATH
                        train dataset path
  --val-path VAL_PATH   validate dataset path
  --model MODEL         the classification model
  --pretrained PRETRAINED
                        use the ImageNet pretrained model or not
  --resume PATH         path to latest checkpoint (default: none)
  --num-classes NUM_CLASSES
                        num of classes
  --in-channels IN_CHANNELS
                        input image channels
  --device DEVICE
  -b BATCH_SIZE, --batch-size BATCH_SIZE
                        batch size
  --epochs EPOCHS       train epochs
  --lr LR               initial learning rate
  --print-freq PRINT_FREQ
                        print frequency
  --ckp-dir CKP_DIR     path to save checkpoint
CHAPTER 9

API Reference

9.1 torchsat package

9.1.1 Subpackages
torchsat.datasets package

Submodules
torchsat.datasets.eurosat module

```python
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

```python
class torchsat.datasets.folder.ChangeDetectionDataset(root, extentsions='jpg', transforms=None)
    Bases: torch.utils.data.dataset.Dataset

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

```
```
```
TorchSat

```python
Args:
root (string): root dir of train or validate dataset. extensions (tuple or list): extention of training image.
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/xyy.ext
root/class_x/xxz.ext

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>, **kwargs)
Bases: torchsat.datasets.folder.DatasetFolder
A generic data loader where the images are arranged in this way:

```
root/dog/xxx.png
root/dog/xyy.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

```python
class torchsat.datasets.folder.SegmentationDataset (root, extensions='jpg', transforms=None)
```

**Bases:** object

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

```
train
├── train_1.png
├── train_2.png
├── ...
└── label
    ├── train_1.png
    └── train_2.png

val
├── image
│   ├── val_10.png
│   └── val_11.png
└── label
    ├── val_10.png
    └── val_11.png
```

**Args:**
- root (string): root dir of train or validate dataset.
- extensions (tuple or list): extension of training image.

```python
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

```python
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

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

**torchsat.datasets.nwpu_resisc45 module**

```python
class torchsat.datasets.nwpu_resisc45.NWPU_RESISC45 (root, download=False, **kwargs)
```

**Bases:** torchsat.datasets.folder.DatasetFolder

**download** ()

```python
url = 'https://sov8mq.dm.files.1drv.com/y4m_Fo6ujJ52LiWHDzaRZVtkMlZxF7aqjX2q7KdVr329zV1'
```

**torchsat.datasets.patternnet module**

```python
class torchsat.datasets.patternnet.PatternNet (root, download=False, **kwargs)
```

**Bases:** torchsat.datasets.folder.DatasetFolder

**download** ()

```python
url = 'https://doc-0k-9c-docs.googleusercontent.com/docs/securesc/s4mst7k8sd1kn5gslv2v'
```

9.1. torchsat package
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]
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.image_loader(path)
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

```python
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”

- **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 (i.e. \( bn \times k \) features in the bottleneck layer)
- **bn_size** (int) - multiplicative factor for number of bottleneck layers
- **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.

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

Densenet-121 model from “Densely Connected Convolutional Networks”

- **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”

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

Densenet-169 model from “Densely Connected Convolutional Networks”

- **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”

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

Densenet-201 model from “Densely Connected Convolutional Networks”

9.1. torchsat package
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

def 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

def 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)


pretrained (bool): If True, returns a model pre-trained on ImageNet. If in_channels is greater than 3, it will copy the parameters of imagenet to fill in order.
t=\text{progress}\ (\text{bool}): \text{If True, displays a progress bar of the download to stderr}

\text{torcsat.models.classification.resnet.resnet34}\ (\text{num_classes}, \text{in_channels}=3, \text{pretrained}=\text{False}, \text{progress}=\text{True}, **\text{kwargs})


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

\text{torcsat.models.classification.resnet.resnet50}\ (\text{num_classes}, \text{in_channels}=3, \text{pretrained}=\text{False}, \text{progress}=\text{True}, **\text{kwargs})


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

\text{torcsat.models.classification.resnet.resnet101}\ (\text{num_classes}, \text{in_channels}=3, \text{pretrained}=\text{False}, \text{progress}=\text{True}, **\text{kwargs})


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

\text{torcsat.models.classification.resnet.resnet152}\ (\text{num_classes}, \text{in_channels}=3, \text{pretrained}=\text{False}, \text{progress}=\text{True}, **\text{kwargs})


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

\text{torcsat.models.classification.resnet.rexnext50_32x4d}\ (\text{num_classes}, \text{in_channels}=3, \text{pretrained}=\text{False}, \text{progress}=\text{True}, **\text{kwargs})

\text{ResNeXt-50 32x4d model from} "\text{Aggregated Residual Transformation for Deep Neural Networks}" \text{Args:}

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

\text{torcsat.models.classification.resnet.rexnext101_32x8d}\ (\text{num_classes}, \text{in_channels}=3, \text{pretrained}=\text{False}, \text{progress}=\text{True}, **\text{kwargs})

\text{ResNeXt-101 32x8d model from} "\text{Aggregated Residual Transformation for Deep Neural Networks}" \text{Args:}

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

\text{torcsat.models.classification.resnet.wide_resnet50_2}\ (\text{num_classes}, \text{in_channels}=3, \text{pretrained}=\text{False}, \text{progress}=\text{True}, **\text{kwargs})

\text{Wide ResNet-50-2 model from} "\text{Wide Residual Networks}" \text{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.} \text{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

```
def 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)
```


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)
```

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)
```


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

```
class torchsat.models.segmentation.pspnet.PSPNet(num_classes, in_channels=3, backbone='resnet50', pretrained=True, use_aux=True)
```

Bases: torch.nn.modules.module.Module
TorchSat

PSPNet, currently only support 3 channels.

Args: nn ([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.

freeze_bn()

get_backbone_params()

get_decoder_params()

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, pre-trained=False, is_deconv=False)

Bases: torch.nn.modules.module.Module


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. pre-trained (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.unet_resnet34 (num_classes, in_channels=3, pre-trained=False, **kwargs)
torchsat.models.segmentation.unet.unet_resnet101(num_classes, in_channels=3, pre-trained=False, **kwargs)

torchsat.models.segmentation.unet.unet_resnet152(num_classes, in_channels=3, pre-trained=False, **kwargs)

Module contents

Submodules
torchsat.models.utils module
torchsat.models.utils.get_model(name: str, num_classes: int, **kwargs)

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 cropped
top {int} – top size
left {int} – left size
height {int} – cropped height
width {int} – cropped width

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

horizontal flip the bboxes

^  

^

Args:
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:
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)

shift the bbox
Arguments: bboxes {ndarray} – input bboxes, should be num x 4 top {int} – shit to top value, positive mean move down, negative mean move up. left {int} – shit to left value, positive mean move right, negative mean move left.

Returns: [ndarray] – shifted bboxes
torchsat.transforms.functional.bbox_vflip(bboxes, img_height)

vertical flip the bboxes . . . .

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


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)


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 integer, 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 integer or iterable variable 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.rotate_box(bboxes, angle, center_x, center_y)
    rotate_box rotate the bboxes

    **Args:** bboxes (ndarray): the original bboxes, N(numbers) x 4 angle (float): rotate angle should be degree
    center_x (int): rotate center x center_y (int): rotate center y

    **Returns:** ndarray: the rotated bboxes

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

torchsat.transforms.functional.to_grayscale(img, output_channels=1)
    convert input ndarray image to gray scale 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 \([0\sim1]\)

    **Arguments:** img {numpy.ndarray} – image to be converted to tensor. torch.float32

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 several classification transforms together.

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

**Example:**

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

```python
class torchsat.transforms.transforms_cls.Lambda(lambda)
    Bases: object

Apply a user-defined lambda as function.

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

```python
class torchsat.transforms.transforms_cls.ToTensor
    Bases: object

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 be between [0-1];

**Args:** img (numpy.ndarray) – image to be converted to tensor.
```

```python
class torchsat.transforms.transforms_cls.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225], inplace=False)
    Bases: object

Normalize a tensor image with mean and standard deviation.

- Given mean: \((M_1, \ldots, M_n)\) and std: \((S_1, \ldots, S_n)\) 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 mutate 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)
```

```python
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 grayscale 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)
```

```python
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)
```
**TorchSat**

**Returns:** ndarray: the blurred image.

```python
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.
```

```python
class torchsat.transforms.transforms_cls.RandomBrightness:
    max_value=0

Bases: object
```

```python
class torchsat.transforms.transforms_cls.RandomContrast:
    max_factor=0

Bases: object
```

```python
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.
```

```python
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 sequence, 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.
```

```python
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 integer, 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
```

```python
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:

```python
>>> 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.
**TorchSat**

*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


**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 several classification transform together.

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

**Example:**

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

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

Bases: object

Apply a user-defined lambda as function.

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

*class* torchsat.transforms.transforms_det.ToTensor

Bases: object

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];
**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 should be [box_nums]

```python
class torchsat.transforms.transforms_det.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225], inplace=False)
```

Bases: object

Normalize a tensor image with mean and standard deviation.

Given mean: \((M_1, \ldots, M_n)\) and std: \((S_1, \ldots, S_n)\) 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 mutate 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)

```python
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 grayscale 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)

```python
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.

```python
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.

```python
class torchsat.transforms.transforms_det.RandomBrightness(max_value=0)
```

Bases: object

```python
class torchsat.transforms.transforms_det.RandomContrast(max_factor=0)
```

Bases: object

```python
class torchsat.transforms.transforms_det.Resize(size, interpolation=2)
```

Bases: object

resize the image Args:
**TorchSat**

`img {ndarray}`: the input ndarray image size
`{int, iterable}`: the target size, if size is integer, 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 integer or iterable variable and length should be 2.

**Returns:** `img {ndarray}`: resize ndarray image

```python
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:**

```python
>>> 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')
```

```python
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 cropped ndarray image.

```python
class torchsat.transforms.transforms_det.RandomCrop(size)
```

**Bases:** `object`

random crop the input ndarray image

**Args:**
- `size (int, sequence)`: the output image size, if sequence size should be [height, width]

**Returns:** `ndarray`: return random cropped ndarray image.

```python
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(lambda)
    Bases: object

class torchsat.transforms.transforms_seg.ToTensor
    Bases: object

class torchsat.transforms.transforms_seg.Normalize(mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225], 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 grayscale 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 sequence, 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 integer, 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:
```python
>>> 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
    out_size (number or sequence): the output image size. if sequence, should be [height, width]

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

Returns:
    ndarray: return cropped ndarray image.

class `torchsat.transforms.transforms_seg.RandomCrop(size)`

Bases: object
random crop the input ndarray image

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

Returns:
    ndarray: return random cropped ndarray image.

class `torchsat.transforms.transforms_seg.RandomHorizontalFlip(p=0.5)`

Bases: object
Flip the input image on central horizontal line.

Args:
    p (float): probability apply the horizontal 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 vertically or horizontally.

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/601cdf56d2b42475de5 ..

Convolutional Neural Networks applied to Visual Document Analysis”, in Proc. of the International

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

9.1.2 Module contents
CHAPTER 10

Indices and tables

- genindex
- modindex
- search
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>torchsat</td>
<td>56</td>
</tr>
<tr>
<td>torchsat.datasets</td>
<td>36</td>
</tr>
<tr>
<td>torchsat.datasets.eurosat</td>
<td>33</td>
</tr>
<tr>
<td>torchsat.datasets.folder</td>
<td>33</td>
</tr>
<tr>
<td>torchsat.datasets.nwpu_resisc45</td>
<td>35</td>
</tr>
<tr>
<td>torchsat.datasets.patternnet</td>
<td>35</td>
</tr>
<tr>
<td>torchsat.datasets.sat</td>
<td>36</td>
</tr>
<tr>
<td>torchsat.datasets.utils</td>
<td>36</td>
</tr>
<tr>
<td>torchsat.models</td>
<td>44</td>
</tr>
<tr>
<td>torchsat.models.classification</td>
<td>42</td>
</tr>
<tr>
<td>torchsat.models.classification.densenet</td>
<td>37</td>
</tr>
<tr>
<td>torchsat.models.classification.inception</td>
<td>38</td>
</tr>
<tr>
<td>torchsat.models.classification.mobilenet</td>
<td>39</td>
</tr>
<tr>
<td>torchsat.models.classification.resnet</td>
<td>39</td>
</tr>
<tr>
<td>torchsat.models.classification.vgg</td>
<td>41</td>
</tr>
<tr>
<td>torchsat.models.segmentation</td>
<td>44</td>
</tr>
<tr>
<td>torchsat.models.segmentation.pspnet</td>
<td>42</td>
</tr>
<tr>
<td>torchsat.models.segmentation.unet</td>
<td>43</td>
</tr>
<tr>
<td>torchsat.models.utils</td>
<td>44</td>
</tr>
<tr>
<td>torchsat.transforms</td>
<td>56</td>
</tr>
<tr>
<td>torchsat.transforms.functional</td>
<td>44</td>
</tr>
<tr>
<td>torchsat.transforms.transforms_cls</td>
<td>46</td>
</tr>
<tr>
<td>torchsat.transforms.transforms_det</td>
<td>50</td>
</tr>
<tr>
<td>torchsat.transforms.transforms_seg</td>
<td>53</td>
</tr>
<tr>
<td>torchsat.utils</td>
<td>56</td>
</tr>
</tbody>
</table>
Index

A
accimage_loader() (in module torchsat.datasets.utils), 36
adjust_brightness() (in module torchsat.transforms.functional), 44
adjust_contrast() (in module torchsat.transforms.functional), 44
adjust_hue() (in module torchsat.transforms.functional), 44
adjust_saturation() (in module torchsat.transforms.functional), 44

B
bbox_crop() (in module torchsat.transforms.functional), 44
bbox_hflip() (in module torchsat.transforms.functional), 44
bbox_pad() (in module torchsat.transforms.functional), 44
bbox_resize() (in module torchsat.transforms.functional), 44
bbox_shift() (in module torchsat.transforms.functional), 44
bbox_vflip() (in module torchsat.transforms.functional), 45

classes_sat6 (torchsat.datasets.sat.SAT attribute), 36
Compose (class in torchsat.transforms.transforms_cls), 46
Compose (class in torchsat.transforms.transforms_det), 50
Compose (class in torchsat.transforms.transforms_seg), 53
crop() (in module torchsat.transforms.functional), 45

d
DatasetFolder (class in torchsat.datasets.folder), 34
default_loader() (in module torchsat.datasets.utils), 36
DenseNet (class in torchsat.models.classification.densenet), 37
densenet121() (in module torchsat.models.classification.densenet), 37
densenet161() (in module torchsat.models.classification.densenet), 38
densenet169() (in module torchsat.models.classification.densenet), 37
densenet201() (in module torchsat.models.classification.densenet), 37
download() (torchsat.datasets.eurosat.EuroSAT method), 33
download() (torchsat.datasets.nwpu_resisc45.NWPU_RESISC45 method), 35
download() (torchsat.datasets.patternnet.PatternNet method), 35
download() (torchsat.datasets.sat.SAT method), 36
download_url() (in module torchsat.datasets.utils), 36
E
elastic_transform() (in module torchsat.transforms.functional), 45
ElasticTransform (class in torchsat.transforms.transforms_cls), 50
ElasticTransform (class in torchsat.transforms.transforms_seg), 55
EuroSAT (class in torchsat.datasets.eurosat), 33

F
flip() (in module torchsat.transforms.functional), 45
forward() (torchsat.models.classification.densenet.DenseNet method), 37
forward() (torchsat.models.classification.inception.Inception3 method), 38
forward() (torchsat.models.classification.mobilenet.MobileNetV2 method), 39
forward() (torchsat.models.classification.resnet.ResNet method), 39
forward() (torchsat.models.classification.vgg.VGG method), 41
forward() (torchsat.models.segmentation.pspnet.PSPNet method), 43
forward() (torchsat.models.segmentation.unet.UNetResNet method), 43
freeze_bn() (torchsat.models.segmentation.pspnet.PSPNet method), 43

G
gaussian_blur() (in module torchsat.transforms.functional), 45
GaussianBlur (class in torchsat.transforms.transforms_cls), 47
GaussianBlur (class in torchsat.transforms.transforms_det), 51
GaussianBlur (class in torchsat.transforms.transforms_seg), 53
gen_bar_updater() (in module torchsat.datasets.utils), 36
get Backbone_params() (torchsat.models.segmentation.pspnet.PSPNet method), 43
get decoder_params() (torchsat.models.segmentation.pspnet.PSPNet method), 43
get_model() (in module torchsat.models.utils), 44

H
has_file_allowed_extension() (in module torchsat.datasets.folder), 35
hflip() (in module torchsat.transforms.functional), 45

I
image_loader() (in module torchsat.datasets.utils), 36
ImageFolder (class in torchsat.datasets.folder), 34
Inception3 (class in torchsat.models.classification.inception), 38

L
Lambda (class in torchsat.transforms.transforms_cls), 47
Lambda (class in torchsat.transforms.transforms_det), 50
Lambda (class in torchsat.transforms.transforms_seg), 53
make_dataset() (in module torchsat.datasets.folder), 35
mobilenet_v2() (in module torchsat.models.classification.mobilenet), 39
MobileNetV2 (class in torchsat.models.classification.mobilenet), 39

N
noise() (in module torchsat.transforms.functional), 45
Normalize (class in torchsat.transforms.transforms_cls), 47
Normalize (class in torchsat.transforms.transforms_det), 51
Normalize (class in torchsat.transforms.transforms_seg), 53
normalize() (in module torchsat.transforms.functional), 45
NWPU_RESISC45 (class in torchsat.datasets.nwpu_resisc45), 35

P
Pad (class in torchsat.transforms.transforms_cls), 48
Pad (class in torchsat.transforms.transforms_det), 52
Pad (class in torchsat.transforms.transforms_seg), 54
pad() (in module torchsat.transforms.functional), 45
PatternNet (class in torchsat.datasets.patternnet), 35
pil_loader() (in module torchsat.datasets.utils), 36
preserve_channel_dim() (in module torchsat.transforms.functional), 45
PSPNet (class in torchsat.models.classification.pspnet), 42

R
RandomBrightness (class in torchsat.transforms.transforms_cls), 48
RandomBrightness (class in torchsat.transforms.transforms_det), 51
RandomBrightness (class in torchsat.transforms.transforms_seg), 54
TorchSat

RandomContrast (class in torchsat.transforms.transforms_cls), 48
RandomContrast (class in torchsat.transforms.transforms_det), 51
RandomContrast (class in torchsat.transforms.transforms_seg), 54
RandomCrop (class in torchsat.transforms.transforms_cls), 49
RandomCrop (class in torchsat.transforms.transforms_det), 52
RandomCrop (class in torchsat.transforms.transforms_seg), 55
RandomFlip (class in torchsat.transforms.transforms_cls), 49
RandomFlip (class in torchsat.transforms.transforms_det), 53
RandomFlip (class in torchsat.transforms.transforms_seg), 55
RandomHorizontalFlip (class in torchsat.transforms.transforms_cls), 49
RandomHorizontalFlip (class in torchsat.transforms.transforms_det), 52
RandomHorizontalFlip (class in torchsat.transforms.transforms_seg), 55
RandomNoise (class in torchsat.transforms.transforms_cls), 48
RandomNoise (class in torchsat.transforms.transforms_det), 51
RandomNoise (class in torchsat.transforms.transforms_seg), 55
RandomResizedCrop (class in torchsat.transforms.transforms_cls), 49
RandomResizedCrop (class in torchsat.transforms.transforms_det), 53
RandomResizedCrop (class in torchsat.transforms.transforms_seg), 55
RandomRotation (class in torchsat.transforms.transforms_cls), 48
RandomRotation (class in torchsat.transforms.transforms_det), 54
RandomVerticalFlip (class in torchsat.transforms.transforms_cls), 49
RandomVerticalFlip (class in torchsat.transforms.transforms_det), 52
RandomVerticalFlip (class in torchsat.transforms.transforms_seg), 55
Resize (class in torchsat.transforms.transforms_cls), 48
Resize (class in torchsat.transforms.transforms_det), 51

torch- Resize (class in torchsat.transforms.transforms_seg), 54
torch- resize() (in module torchsat.transforms.functional), 46
torch- resized_crop() (in module torchsat.transforms.functional), 46
torch- ResNet (class in torchsat.models.classification.resnet), 39
torch- resnet101() (in module torchsat.models.classification.resnet), 40
torch- resnet152() (in module torchsat.models.classification.resnet), 40
torch- resnet18() (in module torchsat.models.classification.resnet), 40
torch- resnet34() (in module torchsat.models.classification.resnet), 40
torch- resnet50() (in module torchsat.models.classification.resnet), 40
torch- resnext101_32x8d() (in module torchsat.models.classification.resnet), 40
torch- resnext50_32x4d() (in module torchsat.models.classification.resnet), 40
torch- rotate() (in module torchsat.transforms.functional), 46
torch- rotate_box() (in module torchsat.transforms.functional), 46
torch- S
torch- SAT (class in torchsat.datasets.sat), 36
torch- SegmentationDataset (class in torchsat.datasets.folder), 35
torch- shift() (in module torchsat.transforms.functional), 46
torch- T
torch- tifffile_loader() (in module torchsat.datasets.utils), 36
torch- to_grayscale() (in module torchsat.transforms.functional), 46
torch- to_pil_image() (in module torchsat.transforms.functional), 46
torch- to_tensor() (in module torchsat.transforms.functional), 46
torch- to_tiff_image() (in module torchsat.transforms.functional), 46
torch- ToGray (class in torchsat.transforms.transforms_cls), 47
torch- ToGray (class in torchsat.transforms.transforms_det), 51
torch- ToGray (class in torchsat.transforms.transforms_seg), 53
torchsat (module), 56
torchsat.datasets (module), 36
torchsat.datasets.eurosat (module), 33
torchsat.datasets.folder (module), 33
TorchSat

torchsat.datasets.nwpu_resisc45 (module), 35
torchsat.datasets.patternnet (module), 35
torchsat.datasets.sat (module), 36
torchsat.datasets.utils (module), 36
torchsat.models (module), 44
torchsat.models.classification (module), 42
torchsat.models.classification.densenet (module), 37
torchsat.models.classification.inception (module), 38
torchsat.models.classification.mobilnet (module), 39
torchsat.models.classification.resnet (module), 39
torchsat.models.classification.vgg (module), 41
torchsat.models.segmentation (module), 44
torchsat.models.segmentation.pspnet (module), 42
torchsat.models.segmentation.unet (module), 43
torchsat.models.utils (module), 44
torchsat.transforms (module), 56
torchsat.transforms.functional (module), 44
torchsat.transforms.transforms_cls (module), 46
torchsat.transforms.transforms_det (module), 50
torchsat.transforms.transforms_seg (module), 53
torchsat.utils (module), 56
ToTensor (class in torchsat.transforms.transforms_cls), 47
ToTensor (class in torchsat.transforms.transforms_det), 50
ToTensor (class in torchsat.transforms.transforms_seg), 53

V
vflip() (in module torchsat.transforms.functional), 46
VGG (class in torchsat.models.classification.vgg), 41
vgg11() (in module torchsat.models.classification.vgg), 41
vgg11_bn() (in module torchsat.models.classification.vgg), 41
vgg13() (in module torchsat.models.classification.vgg), 41
vgg13_bn() (in module torchsat.models.classification.vgg), 41
vgg16() (in module torchsat.models.classification.vgg), 42
vgg16_bn() (in module torchsat.models.classification.vgg), 42
vgg19() (in module torchsat.models.classification.vgg), 42
vgg19_bn() (in module torchsat.models.classification.vgg), 42

W
wide_resnet101_2() (in module torchsat.models.classification.resnet), 41
wide_resnet50_2() (in module torchsat.models.classification.resnet), 40

U
unet_resnet101() (in module torchsat.models.segmentation.unet), 43
unet_resnet152() (in module torchsat.models.segmentation.unet), 44
unet_resnet34() (in module torchsat.models.segmentation.unet), 43
UNetResNet (class in torchsat.models.segmentation.unet), 43
url (torchsat.datasets.nwpu_resisc45.NWPU_RESISC45 attribute), 35
url (torchsat.datasets.patternnet.PatternNet attribute), 35