Extorch

Release 1.0.4

Junbo Zhao

Apr 12, 2022

USER API

1 extorch.vision **3**
- 1.1 extorch.vision.dataset . . . 3
- 1.2 extorch.vision.transforms . . . 5

2 extorch.utils **11**
- 2.1 extorch.utils.common . . . 11
- 2.2 extorch.utils.data . . . 13
- 2.3 extorch.utils.exception . . . 13
- 2.4 extorch.utils.logger . . . 14
- 2.5 extorch.utils.stats . . . 14
- 2.6 extorch.utils.visual . . . 15

3 extorch.nn **17**
- 3.1 extorch.nn.functional . . . 17
- 3.2 extorch.nn.modules . . . 18
- 3.3 extorch.nn.init . . . 28
- 3.4 extorch.nn.utils . . . 29
- 3.5 extorch.nn.optim . . . 30

4 extorch.adversarial **33**
- 4.1 extorch.adversarial.base . . . 33
- 4.2 extorch.adversarial.gradient_based . . . 34

5 Indices and tables **39**

Python Module Index **41**

Index **43**

Extorch is an useful extension library of PyTorch.

The source code and some minimal working examples can be found on GitHub.

CHAPTER

ONE

EXTORCH.VISION

1.1 extorch.vision.dataset

BaseDataset	Base dataset.
CVDataset	Base dataset for computer vision tasks.
CVClassificationDataset	Base dataset for computer vision classification tasks.
SegmentationDataset	Base dataset for computer vision segmentation tasks.
CIFAR10	
CIFAR100	
MNIST	
FashionMNIST	
SVHN	
TinyImageNet	Tiny ImageNet dataset.
VOCSegmentationDataset	

class extorch.vision.dataset.**BaseDataset**(*data_dir: str*)

Base dataset.

Parameters **data_dir** (*str*) – Base path of the data.

class extorch.vision.dataset.**CVDataset**(*data_dir: str*, *train_transform: torchvision.transforms.transforms.Compose*, *test_transform: torchvision.transforms.transforms.Compose*)

Base dataset for computer vision tasks.

Parameters

- **data_dir** (*str*) – Base path of the data.
- **train_transform** (*transforms.Compose*) – Data transform of the training split.
- **test_transform** (*transforms.Compose*) – Data transform of the test split.

class extorch.vision.dataset.**CVClassificationDataset**(*data_dir: str*, *train_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *test_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *cutout_length: Optional[int] = None*, *cutout_n_holes: Optional[int] = 1*)

Base dataset for computer vision classification tasks.

Parameters

- **data_dir** (*str*) – Base path of the data.
- **train_transform** (*Optional[transforms.Compose]*) – Data transform of the training split. Default: None.
- **test_transform** (*Optional[transforms.Compose]*) – Data transform of the test split. Default: None.
- **cutout_length** (*Optional[int]*) – The length (in pixels) of each square patch in Cutout. If train transform is not specified and cutout_length is not None, we will add Cutout transform at the end. Default: None.
- **cutout_n_holes** (*Optional[int]*) – Number of patches to cut out of each image. Default: 1.

class extorch.vision.dataset.**SegmentationDataset**(*data_dir: str*, *train_transform: torchvision.transforms.transforms.Compose*, *test_transform: torchvision.transforms.transforms.Compose*)

Base dataset for computer vision segmentation tasks.

Parameters

- **data_dir** (*str*) – Base path of the data.
- **train_transform** (*Optional[transforms.Compose]*) – Data transform of the training split. Default: None.
- **test_transform** (*Optional[transforms.Compose]*) – Data transform of the test split. Default: None.

class extorch.vision.dataset.**CIFAR10**(*data_dir: str*, *train_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *test_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *cutout_length: Optional[int] = None*, *cutout_n_holes: Optional[int] = 1*)

class extorch.vision.dataset.**CIFAR100**(*data_dir: str*, *train_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *test_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *cutout_length: Optional[int] = None*, *cutout_n_holes: Optional[int] = 1*)

class extorch.vision.dataset.**MNIST**(*data_dir: str*, *train_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *test_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *cutout_length: Optional[int] = None*, *cutout_n_holes: Optional[int] = 1*)

class extorch.vision.dataset.**FashionMNIST**(*data_dir: str*, *train_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *test_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *cutout_length: Optional[int] = None*, *cutout_n_holes: Optional[int] = 1*)

class extorch.vision.dataset.**SVHN**(*data_dir: str*, *train_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *test_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *cutout_length: Optional[int] = None*, *cutout_n_holes: Optional[int] = 1*)

class extorch.vision.dataset.**TinyImageNet**(*data_dir: str*, *color_jitter: bool = False*, *train_crop_size: int = 64*, *test_crop_size: int = 64*, *train_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *test_transform: Optional[torchvision.transforms.transforms.Compose] = None*, *cutout_length: Optional[int] = None*, *cutout_n_holes: Optional[int] = 1*)

Tiny ImageNet dataset.

Parameters

- **color_jitter** (*bool*) – Whether add color_jitter into default train transform. Default: `False`.
- **train_crop_size** (*int*) – Default: 64.
- **test_crop_size** (*int*) – Default: 64.

class extorch.vision.dataset.**VOCSegmentationDataset**(*data_dir: str*, *year: str = '2012'*, *train_transform: Optional[extorch.vision.transforms.segmentation.SegCompose] = None*, *test_transform: Optional[extorch.vision.transforms.segmentation.SegCompose] = None*)

1.2 extorch.vision.transforms

1.2.1 extorch.vision.transforms.functional

extorch.vision.transforms.functional.**cutout**(*image: torch.Tensor*, *length: int*, *n_holes: int = 1*) → torch.Tensor

Cutout: Randomly mask out one or more patches from an image (Link).

Parameters

- **image** (*Tensor*) – Image of size (C, H, W).
- **length** (*int*) – The length (in pixels) of each square patch.
- **n_holes** (*int*) – Number of patches to cut out of each image. Default: 1.

Returns Image with n_holes of dimension length x length cut out of it.

Return type image (Tensor)

Examples::

```
>>> image = torch.ones((3, 32, 32))
>>> image = cutout(image, 16, 1)
```

1.2.2 extorch.vision.transforms.transforms

AdaptiveRandomCrop	Adaptively randomly crop images with uncertain sizes for a certain size.
AdaptiveCenterCrop	Adaptively center-crop images with uncertain sizes for a certain size.
Cutout	Cutout: Randomly mask out one or more patches from an image (Link).

class extorch.vision.transforms.transforms.**AdaptiveCenterCrop**(*cropped_size: Union[int, Tuple[int, int]]*)

Bases: torch.nn.modules.module.Module

Adaptively center-crop images with uncertain sizes for a certain size.

Parameters **cropped_size** (*Union[int, Tuple[int, int]]*) – The Image size to be cropped out.

forward(*img: torch.Tensor*) → torch.Tensor

Parameters **img** (*Tensor*) – The image to be cropped.

Returns

The cropped image. For example, if the image has size [H, W] and the cropped size if [h, w], size of output will be [H - h, W - w].

Return type img (Tensor)

training: bool

class extorch.vision.transforms.transforms.**AdaptiveRandomCrop**(*cropped_size: Union[int, Tuple[int, int]]*)

Bases: torch.nn.modules.module.Module

Adaptively randomly crop images with uncertain sizes for a certain size.

Parameters **cropped_size** (*Union[int, Tuple[int, int]]*) – The Image size to be cropped out.

forward(*img: torch.Tensor*) → torch.Tensor

Parameters **img** (*Tensor*) – The image to be cropped.

Returns

The cropped image. For example, if the image has size [H, W] and the cropped size if [h, w], size of output will be [H - h, W - w].

Return type img (Tensor)

training: bool

class extorch.vision.transforms.transforms.**Cutout**(*length: int*, *n_holes: int = 1*)

Bases: torch.nn.modules.module.Module

Cutout: Randomly mask out one or more patches from an image (Link).

Parameters

- **length** (*int*) – The length (in pixels) of each square patch.
- **image** (*Tensor*) – Image of size (C, H, W).
- **n_holes** (*int*) – Number of patches to cut out of each image. Default: 1.

Examples::

```
>>> image = torch.ones((3, 32, 32))
>>> Cutout_transform = Cutout(16, 1)
>>> image = Cutout_transform(image)  # Shape: [3, 32, 32]
```

forward(*img: torch.Tensor*) → torch.Tensor

Parameters **img** (*Tensor*) – Image of size (C, H, W).

Returns Image with n_holes of dimension length x length cut out of it.

Return type img (Tensor)

training: bool

1.2.3 extorch.vision.transforms.segmentation

SegCompose	Transform compose for segmentation.
SegRandomHorizontalFlip	Horizontally flip the given image and label randomly with a given probability (Link).
SegNormalize	Normalization for segmentation, where the normalization is only applied on the input image.
SegCenterCrop	Crops the given image at the center.
SegRandomCrop	Random cropping for segmentation.
SegResize	Resize for segmentation.
SegRandomResize	Random resize for segmentation.
SegPILToTensor	PIL to Tensor for segmentation.
SegConvertImageDtype	Convert image dtype for segmentation.

class extorch.vision.transforms.segmentation.**SegCenterCrop**(*size: Union[int, List[int]]*)

Bases: torch.nn.modules.module.Module

Crops the given image at the center.

Parameters **size** (*Union[int, List[int]]*) – Height and width of the crop box.

forward(*image*, *target*)

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.

training: bool

class extorch.vision.transforms.segmentation.**SegCompose**(*transforms*)

Bases: torchvision.transforms.transforms.Compose

Transform compose for segmentation.

class extorch.vision.transforms.segmentation.**SegConvertImageDtype**(*dtype*)

Bases: torch.nn.modules.module.Module

Convert image dtype for segmentation.

forward(*image*, *target*)

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.

training: bool

class extorch.vision.transforms.segmentation.**SegNormalize**(*mean: List[float]*, *std: List[float]*)

Bases: torch.nn.modules.module.Module

Normalization for segmentation, where the normalization is only applied on the input image.

Parameters

- **mean** (*List[float]*) – List of means for each channel.
- **std** (*List[float]*) – List of standard deviations for each channel.

forward(*image*, *target*)

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.

training: bool

class extorch.vision.transforms.segmentation.**SegPILToTensor**

Bases: torch.nn.modules.module.Module

PIL to Tensor for segmentation.

forward(*image*, *target*)

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.

---

training: bool

class extorch.vision.transforms.segmentation.**SegRandomCrop**(*size: Union[int, Tuple[int, int]]*)

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

Random cropping for segmentation. The Cropping is applied on the image and target at the same time.

Parameters **size** (*Union[int, Tuple[int, int]]*) – Desired output size of the crop.

forward(*image*, *target*)

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.

---

training: bool

class extorch.vision.transforms.segmentation.**SegRandomHorizontalFlip**(*p: float = 0.5*)

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

Horizontally flip the given image and label randomly with a given probability (Link).

If the image and label are torch Tensor, they are expected to have [..., H, W] shape, where ... means an arbitrary number of leading dimensions.

Parameters **p** (*float*) – probability of the image and label being flipped. Default: 0.5.

forward(*image*, *target*)

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.

---

training: bool

class extorch.vision.transforms.segmentation.**SegRandomResize**(*min_size: Union[int, Tuple[int, int]]*, *max_size: Optional[Union[T, Tuple[T, T]]] = None*)

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

Random resize for segmentation.

Parameters

- **min_size** (*Union[int, Tuple[int, int]]*) – Desired minimum output size.
- **max_size** (*Optional[Union[int, Tuple[int, int]]]*) – Desired maximum output size. Default: *None*.

forward(*image*, *target*)

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.

---

training: bool

class extorch.vision.transforms.segmentation.**SegResize**(*size: Union[int, Tuple[int, int]]*)

Bases: torch.nn.modules.module.Module

Resize for segmentation.

Parameters **size** (*Union[int, Tuple[int, int]]*) – Desired output size.

forward(*image*, *target*)

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.

---

training: bool

1.2.4 extorch.vision.transforms.detection

DetCompose	Transform compose for detection.

class extorch.vision.transforms.detection.**DetCompose**(*transforms*)

Bases: torchvision.transforms.transforms.Compose

Transform compose for detection.

CHAPTER

TWO

EXTORCH.UTILS

2.1 extorch.utils.common

class extorch.utils.common.**TimeEstimator**(*iter_num: int*)

Bases: object

Remaining time estimator. Estimate the remaining time to finish all iterations based on history.

Parameters **iter_num** (*int*) – Total iteration number.

step() → Tuple[datetime.datetime, datetime.datetime]

Update the estimator and return the estimated remaining time.

Returns The remaining time estimated based on the whole history. nearest_time (datetime): The remaining time estimated based on the last iteration.

Return type overall_time (datetime)

class extorch.utils.common.**abstractclassmethod**(*a_callable*)

Bases: classmethod

extorch.utils.common.**makedir**(*path: str*, *remove: bool = False*, *quiet: bool = False*) → None

Create a leaf directory and all intermediate ones.

Parameters

- **path** (*str*) – The directory to be made.
- **remove** (*bool*) – Sometimes, the targeted directory has existed. If *remove* is false, the targeted directory will not be removed. Default: False.
- **quiet** (*bool*) – When the targeted directory has existed and *remove* == *True*: If *quiet* == *False*, we will ask whether to remove the existing directory. Else, the existing directory will be removed directly. Default: False.

extorch.utils.common.**nullcontext**()

extorch.utils.common.**remove_dirs**(*root: str*, *target: str*) → int

Delete the target dirs under the given root recursively.

Parameters

- **root** (*str*) – Root.
- **target** (*str*) – Name of the target dirs.

Returns Number of the deleted dirs.

Return type delete_num (int)

Examples::

```
>>> # Remove all dirs named "__pycache__" under the current path recursively
>>> delete_num = remove_dirs("./", "__pycache__")
```

extorch.utils.common.**remove_files**(*root: str*, *target: str*) → int

Delete the target files under the given root recursively.

Parameters

- **root** (*str*) – Root.
- **target** (*str*) – Name of the target files.

Returns Number of the deleted files.

Return type delete_num (int)

Examples::

```
>>> # Remove files named "__init__.py" under the current path recursively
>>> delete_num = remove_files("./", "__init__.py")
```

extorch.utils.common.**remove_targets**(*root: str*, *target: str*) → int

Delete the target files and dirs under the given root recursively.

Parameters

- **root** (*str*) – Root.
- **target** (*str*) – Name of the target files or dirs.

Returns Number of the deleted files or dirs.

Return type delete_num (int)

Examples::

```
>>> # Remove dirs or files named "__pycache__" under the current path␣
↪recursively
>>> delete_num = remove_targets("./", "__pycache__")
```

extorch.utils.common.**run_processes**(*commands: List[str]*, ***kwargs*) → List[str]

Run a list of commands one-by-one.

Parameters

- **command** (*List[str]*) – A list of commands to be run.
- **kwargs** – Configurations except 'shell' for *subprocess.check_call*. We note that *shell* must be *True* for this function.

Returns A list of commands that fail to run successfully.

Return type unsuccessful_commands (List[str])

Examples::

```
>>> # Assumes a runnable python file named `test.py` exists
>>> # And assumes no file named `not_exist.py` exists
>>> # Now, we want to run `test.py` and `not_exist.py` one-by-one
>>> commands = ["python test.py", "python not_exist.py"]
>>> unsuccessful_commands = run_processes(commands) # ["python not_exist.py"]
>>> print("Unsuccessful commands: {}".format(unsuccessful_commands))
```

extorch.utils.common.**set_seed**(*seed: int*) → None

Set the seed of the system.

Parameters **seed** (*int*) – The artificial random seed.

extorch.utils.common.**set_trace**() → None

extorch.utils.common.**yaml_dump**(*data*, *file: str*, *mode: str = 'w'*, ***kwargs*) → None

Serialize a Python object into a YAML document.

Parameters

- **data** – The Python object to be serialized.
- **file** (*str*) – Path of the YAML document.
- **mode** (*str*) – Mode to open the document.
- **kwargs** – Other options for opening the document.

extorch.utils.common.**yaml_load**(*file: str*, *mode: str = 'r'*, *Loader=<class 'yaml.loader.FullLoader'>*, ***kwargs*)

Parse the YAML document and produce the corresponding Python object.

Parameters

- **file** (*str*) – Path of the YAML document.
- **mode** (*str*) – Mode to open the document.
- **Loader** – Loader for YAML.
- **kwargs** – Other options for opening the document.

2.2 extorch.utils.data

2.3 extorch.utils.exception

exception extorch.utils.exception.**ExtorchException**

Bases: Exception

exception extorch.utils.exception.**InvalidConfigException**

Bases: *extorch.utils.exception.ExtorchException*

exception extorch.utils.exception.**InvalidValueException**

Bases: *extorch.utils.exception.ExtorchException*

extorch.utils.exception.**expect**(*bool_expr: bool*, *message: str = ''*, *exception_type: Exception = <class 'extorch.utils.exception.ExtorchException'>*)

2.4 extorch.utils.logger

extorch.utils.logger.**getLogger**(*name: str*) → logging.Logger

2.5 extorch.utils.stats

class extorch.utils.stats.**AverageMeter**

Bases: object

is_empty() → bool

reset() → None

update(*val: Union[float, int]*, *n: int = 1*) → None

class extorch.utils.stats.**OrderedStats**

Bases: object

avgs() → Optional[collections.OrderedDict]

items()

update(*stats*, *n: int = 1*) → None

class extorch.utils.stats.**SegConfusionMatrix**(*num_classes: int*)

Bases: object

Confusion matrix as well as metric calculation for image segmentation.

Parameters **num_classes** (*int*) – The number of classes, including the background.

compute()

reset() → None

update(*output: torch.Tensor*, *target: torch.Tensor*) → None

extorch.utils.stats.**accuracy**(*outputs: torch.Tensor*, *targets: torch.Tensor*, *topk: Tuple[int] = (1,)*) → List[torch.Tensor]

extorch.utils.stats.**cal_flops**(*model: torch.nn.modules.module.Module*, *inputs: torch.Tensor*) → float

Calculate FLOPs of the given model.

Parameters

- **model** (*nn.Module*) – The model whose FLOPs is to be calculated.
- **inputs** (*Tensor*) – Example inputs to the model.

Returns FLOPs of the model.

Return type flops (float)

Examples::

```
>>> import torch
>>> from extorch.nn import AuxiliaryHead
>>> module = AuxiliaryHead(3, 10)
>>> input = torch.randn((10, 3, 32, 32))
>>> flops = cal_flops(module, input) / 1.e6 # 32.109868
```

extorch.utils.stats.**get_params**(*model: torch.nn.modules.module.Module*, *only_trainable: bool = False*) → int

Get the parameter number of the model.

Parameters **only_trainable** (*bool*) – If only_trainable is true, only trainable parameters will be counted.

2.6 extorch.utils.visual

extorch.utils.visual.**denormalize**(*image: torch.Tensor*, *mean: List[float]*, *std: List[float]*, *transpose: bool = False*, *detach_numpy: bool = False*) → Union[torch.Tensor, numpy.ndarray]

De-normalize the tensor-like image.

Parameters

- **image** (*Tensor*) – The image to be de-normalized with shape [B, C, H, W] or [C, H, W].
- **mean** (*List[float]*) – Sequence of means for each channel while normalizing the origin image.
- **std** (*List[float]*) – Sequence of standard deviations for each channel while normalizing the origin image.
- **transpose** (*bool*) – Whether transpose the image to [B, H, W, C] or [H, W, C]. Default: *False*.
- **detach_numpy** (*bool*) – If true, return *Tensor.detach().cpu().numpy()*.

Returns The de-normalized image.

Return type image (Union[Tensor, np.ndarray])

Examples

```
>>> image = torch.randn((5, 3, 32, 32)).cuda()  # Shape: [5, 3, 32, 32] (cuda)
>>> mean = [0.5, 0.5, 0.5]
>>> std = [1., 1., 1.]
>>> de_image = denormalize(image, mean, std, True, True)  # Shape: [5, 32, 32, 3]␣
↪(cpu)
```

extorch.utils.visual.**tsne_fit**(*feature: numpy.ndarray*, *n_components: int = 2*, *init: str = 'pca'*, ***kwargs*)

Fit input features into an embedded space and return that transformed output.

Parameters

- **feature** (*np.ndarray*) – The features to be embedded.
- **n_components** (*int*) – Dimension of the embedded space. Default: 2.

- **init** (*str*) – Initialization of embedding. Possible options are "random", "pca", and a numpy array of shape (n_samples, n_components). PCA initialization cannot be used with precomputed distances and is usually more globally stable than random initialization. Default: "pca".
- **kwargs** – Other configurations for TSNE model construction.

Returns The representation in the embedding space.

Return type node_pos (np.ndarray)

Examples::

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> features = np.random.randn(50, 10)
>>> labels = np.random.randint(0, 2, (50, 1))
>>> node_pos = tsne_fit(features, 2, "pca")
>>> plt.figure()
>>> plt.scatter(node_pos[:, 0], node_pos[:, 1], c = labels)
>>> plt.show()
```

CHAPTER

THREE

EXTORCH.NN

3.1 extorch.nn.functional

extorch.nn.functional.**average_logits**(*logits_list: List[torch.Tensor]*) → torch.Tensor

Aggregates logits from different networks in the average manner, and returns the aggregated logits. Used for neural ensemble networks.

Parameters **logits_list** (*List[Tensor]*) – A list of logits from different networks.

Returns The aggregated logits (Tensor).

extorch.nn.functional.**dec_soft_assignment**(*input: torch.Tensor*, *centers: torch.Tensor*, *alpha: float*) → torch.Tensor

Soft assignment used by Deep Embedded Clustering (DEC, `Link`_). Measure the similarity between embedded point and centroid with the Student's t-distribution.

Parameters

- **input** (*Tensor*) – A batch of embedded points. FloatTensor of [batch size, embedding dimension].
- **centers** (*Tensor*) – The cluster centroids. FloatTensor of [cluster_number, embedding dimension].
- **alpha** (*float*) – The degrees of freedom of the Student's tdistribution. Default: 1.0.

Returns The similarity between embedded point and centroid. FloatTensor [batch size, cluster_number].

Return type similarity (Tensor)

Examples::

```
>>> embeddings = torch.ones((2, 10))
>>> centers = torch.zeros((3, 10))
>>> similarity = dec_soft_assignment(embeddings, centers, alpha = 1.)
```

extorch.nn.functional.**mix_data**(*input: torch.Tensor*, *target: torch.Tensor*, *alpha: float = 1.0*)

Mixup data for training neural networks on convex combinations of pairs of examples and their labels (`Link`_).

Parameters

- **input** (*Tensor*) – Input examples.
- **target** (*Tensor*) – Labels of input examples.

- **alpha** (*float*) – Parameter of the beta distribution. Default: 1.0.

Returns Input examples after mixup. mixed_target (Tensor): Labels of mixed inputs. _lambda (float): Parameter sampled from the beta distribution.

Return type mixed_input (Tensor)

extorch.nn.functional.**soft_voting**(*logits_list: List[torch.Tensor]*) → torch.Tensor

Aggregates logits from different sub-networks in the soft-voting manner, and returns the aggregated logits. Used for neural ensemble networks.

Parameters **logits_list** (*List[Tensor]*) – A list of logits from different networks.

Returns The aggregated logits (Tensor).

3.2 extorch.nn.modules

3.2.1 extorch.nn.modules.auxiliary

class extorch.nn.modules.auxiliary.**AuxiliaryHead**(*in_channels: int*, *num_classes: int*)

Bases: torch.nn.modules.module.Module

Auxiliary head for the classification task on CIFAR datasets.

Parameters

- **in_channels** (*int*) – Number of channels in the input feature.
- **num_classes** (*int*) – Number of classes.

Examples::

```
>>> import torch
>>> input = torch.randn((10, 3, 32, 32))
>>> module = AuxiliaryHead(3, 10)
>>> output = module(input)
```

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

training: **bool**

class extorch.nn.modules.auxiliary.**AuxiliaryHeadImageNet**(*in_channels: int*, *num_classes: int*)

Bases: torch.nn.modules.module.Module

Auxiliary head for the classification task on the ImageNet dataset.

Parameters

- **in_channels** (*int*) – Number of channels in the input feature.

- **num_classes** (*int*) – Number of classes.

Examples::

```
>>> import torch
>>> input = torch.randn(10, 5, 32, 32)
>>> module = AuxiliaryHeadImageNet(5, 10)
>>> output = module(input)
```

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

training: bool

3.2.2 extorch.nn.modules.loss

HintonKDLoss	Knowledge distillation loss proposed by Hinton (`Link`_).
CrossEntropyLabelSmooth	
CrossEntropyMixupLoss	CrossEntropyLoss with mixup technique.
DECLoss	Loss used by Deep Embedded Clustering (DEC, `Link`_).

class extorch.nn.modules.loss.**CrossEntropyLabelSmooth**(*epsilon: float*)

Bases: torch.nn.modules.module.Module

forward(*input: torch.Tensor*, *target: torch.Tensor*) → torch.Tensor

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.

---

training: bool

class extorch.nn.modules.loss.**CrossEntropyMixupLoss**(*alpha: float = 1.0*, ***kwargs*)

Bases: torch.nn.modules.module.Module

CrossEntropyLoss with mixup technique.

Parameters

- **alpha** (*float*) – Parameter of the beta distribution. Default: 1.0.

- **kwargs** – Other arguments of torch.nn.CrossEntropyLoss (`Link`_).

forward(*input: torch.Tensor*, *target: torch.Tensor*, *net: torch.nn.modules.module.Module*) → torch.Tensor

Parameters

- **input** (*Tensor*) – Input examples.
- **target** (*Tensor*) – Label of the input examples.
- **net** (*nn.Module*) – Network to calculate the loss.

Returns The loss.

Return type loss (Tensor)

training: bool

class extorch.nn.modules.loss.**DECLoss**(*alpha: float = 1.0*, ***kwargs*)

Bases: torch.nn.modules.module.Module

Loss used by Deep Embedded Clustering (DEC, `Link`_).

Parameters **alpha** (*float*) – The degrees of freedom of the Student's tdistribution. Default: 1.0.

Examples::

```
>>> criterion = DECLoss(alpha = 1.)
>>> embeddings = torch.randn((2, 10))
>>> centers = torch.randn((3, 10))
>>> loss = criterion(embeddings, centers)
```

forward(*input: torch.Tensor*, *centers: torch.Tensor*) → torch.Tensor

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.

---

static target_distribution(*input: torch.Tensor*) → torch.Tensor

training: bool

class extorch.nn.modules.loss.**HintonKDLoss**(*T: float*, *alpha: float*, *reduction: str = 'mean'*, ***kwargs*)

Bases: torch.nn.modules.loss.KLDivLoss

Knowledge distillation loss proposed by Hinton (`Link`_).

$L = (1 - alpha) * L_{CE}(P_s, Y_{gt}) + alpha * T^2 * L_{CE}(P_s, P_t)$

Parameters

- **T** (*float*) – Temperature parameter (>= 1.) used to smooth the softmax output.
- **alpha** (*float*) – Trade-off coefficient between distillation and origin loss.
- **reduction** (*str*) – Specifies the reduction to apply to the output. Default: "mean".
- **kwargs** – Other configurations for nn.CrossEntropyLoss.

Examples::

```
>>> criterion = HintonKDLoss(T = 4., alpha = 0.9)
>>> s_output = torch.randn((5, 10))
>>> t_output = torch.randn((5, 10))
>>> target = torch.ones(5, dtype = torch.long)
>>> loss = criterion(s_output, t_output, target)
```

forward(*s_output: torch.Tensor*, *t_output: torch.Tensor*, *target: torch.Tensor*) → torch.Tensor

Parameters

- **s_output** (*Tensor*) – Student network output.
- **t_output** (*Tensor*) – Teacher network output.
- **target** (*Tensor*) – Hard label of the input.

Returns The calculated loss.

Return type Tensor

reduction: str

3.2.3 extorch.nn.modules.mlp

class extorch.nn.modules.mlp.**MLP**(*dim_in: int*, *dim_out: int*, *hiddens: Union[Tuple[int], List[int]]*, *dropout: float = 0.0*)

Bases: torch.nn.modules.module.Module

Basic muti-layer perception with relu as the activation function.

Parameters

- **dim_in** (*int*) – Input dimension.
- **dim_out** (*int*) – Output dimension.
- **hiddens** (*Union[Tuple[int], List[int]]*) – Hidden dimensions.
- **dropout** (*float*) – Applied dropout rate.

Examples::

```
>>> m = MLP(32, 20, (10, 10, 10), 0.1)
>>> input = torch.randn(2, 32) # shape [2, 32]
>>> output = m(input) # shape [2, 20]
```

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

training: bool

3.2.4 extorch.nn.modules.operation

Identity	

BNReLU	A batch-normalization layer followed by relu.
ReLUBN	A batch-normalization layer following relu.
ConvBNReLU	A convolution followed by batch-normalization and ReLU.
ReLUConvBN	A ReLU followed by convolution and batch-normalization.

class extorch.nn.modules.operation.**BNReLU**(*in_channels: int*, *affine: bool = True*)

Bases: torch.nn.modules.module.Module

A batch-normalization layer followed by relu.

Parameters

- **in_channels** (*int*) – Number of channels in the input image.
- **affine** – A boolean value that when set to True, this module has learnable affine parameters. Default: True.

Examples::

```
>>> m = BNReLU(32, True)
>>> input = torch.randn(2, 32, 10, 3)
>>> output = m(input)
```

forward(*input: torch.Tensor*) → torch.Tensor

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.

training: bool

class extorch.nn.modules.operation.**ConvBN**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[str, int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: bool = True*, *affine: bool = True*, ***kwargs*)

Bases: torch.nn.modules.module.Module

A convolution followed by batch-normalization.

Parameters

- **in_channels** (*int*) – Number of channels in the input image.

- **out_channels** (*int*) – Number of channels produced by the convolution.
- **kernel_size** (*int or tuple*) – Size of the convolving kernel.
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1.
- **padding** (*int, tuple or str, optional*) – Padding added to all four sides of e input. Default: 0.
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1.
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1.
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`.
- **affine** (*bool*) – A boolean value that when set to `True`, the batch-normalization layer has learnable affine parameters. Default: `True`.
- **kwargs** – Other configurations of the convolution.

Examples::

```
>>> m = ConvBN(3, 10, 3, 1)
>>> input = torch.randn(2, 3, 32, 32)
>>> output = m(input)
```

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

training: bool

class extorch.nn.modules.operation.**ConvBNReLU**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[str, int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: bool = True*, *affine: bool = True*, ***kwargs*)

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

A convolution followed by batch-normalization and ReLU.

Parameters

- **in_channels** (*int*) – Number of channels in the input image.
- **out_channels** (*int*) – Number of channels produced by the convolution.
- **kernel_size** (*int or tuple*) – Size of the convolving kernel.
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1.
- **padding** (*int, tuple or str, optional*) – Padding added to all four sides of e input. Default: 0.
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1.

- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1.
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`.
- **affine** (*bool*) – A boolean value that when set to `True`, the batch-normalization layer has learnable affine parameters. Default: `True`.
- **kwargs** – Other configurations of the convolution.

Examples::

```
>>> m = ConvBNReLU(3, 10, 3, 1)
>>> input = torch.randn(2, 3, 32, 32)
>>> output = m(input)
```

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

training: bool

class extorch.nn.modules.operation.**ConvReLU**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[str, int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: Optional[int] = 1*, *bias: Optional[bool] = True*, *inplace: Optional[bool] = False*, ***kwargs*)

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

A convolution followed by a ReLU.

Parameters

- **in_channels** (*int*) – Number of channels in the input image.
- **out_channels** (*int*) – Number of channels produced by the convolution.
- **kernel_size** (*int or tuple*) – Size of the convolving kernel.
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1.
- **padding** (*int, tuple or str, optional*) – Padding added to all four sides of e input. Default: 0.
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1.
- **groups** (*Optional[int]*) – Number of blocked connections from input channels to output channels. Default: 1.
- **bias** (*Optional[bool]*) – If `True`, adds a learnable bias to the output. Default: `True`.
- **inplace** (*Optional[bool]*) – ReLU can optionally do the operation in-place. Default: `False`.

- **kwargs** – Other configurations of the convolution.

Examples::

```
>>> m = ConvReLU(3, 10, 3, 1)
>>> input = torch.randn(2, 3, 32, 32)
>>> output = m(input)
```

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

training: bool

class extorch.nn.modules.operation.**Identity**

Bases: torch.nn.modules.module.Module

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

training: bool

class extorch.nn.modules.operation.**ReLUBN**(*in_channels: int*, *affine: bool = True*)

Bases: torch.nn.modules.module.Module

A batch-normalization layer following relu.

Parameters

- **in_channels** (*int*) – Number of channels in the input image.
- **affine** – A boolean value that when set to `True`, this module has learnable affine parameters. Default: `True`.

Examples::

```
>>> m = ReLUBN(32, True)
>>> input = torch.randn(2, 32, 10, 3)
>>> output = m(input)
```

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

training: bool

class extorch.nn.modules.operation.**ReLUConvBN**(*in_channels: int*, *out_channels: int*, *kernel_size: Union[int, Tuple[int, int]]*, *stride: Union[int, Tuple[int, int]] = 1*, *padding: Union[str, int, Tuple[int, int]] = 0*, *dilation: Union[int, Tuple[int, int]] = 1*, *groups: int = 1*, *bias: bool = True*, *affine: bool = True*, ***kwargs*)

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

A ReLU followed by convolution and batch-normalization.

Parameters

- **in_channels** (*int*) – Number of channels in the input image.
- **out_channels** (*int*) – Number of channels produced by the convolution.
- **kernel_size** (*int or tuple*) – Size of the convolving kernel.
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1.
- **padding** (*int, tuple or str, optional*) – Padding added to all four sides of e input. Default: 0.
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1.
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1.
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`.
- **affine** (*bool*) – A boolean value that when set to `True`, the batch-normalization layer has learnable affine parameters. Default: `True`.
- **kwargs** – Other configurations of the convolution.

Examples::

```
>>> m = ReLUConvBN(3, 10, 3, 1)
>>> input = torch.randn(2, 3, 32, 32)
>>> output = m(input)
```

forward(*input: torch.Tensor*) → torch.Tensor

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.

training: bool

3.2.5 extorch.nn.modules.block

ResNetBasicBlock	ResNet basic block (Link).
ResNetBottleneckBlock	ResNet bottleneck block (Link).

class extorch.nn.modules.block.**ResNetBasicBlock**(*in_channels: int*, *out_channels: int*, *stride: int*, *kernel_size: int = 3*, *affine: bool = True*)

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

ResNet basic block (Link).

Parameters

- **in_channels** (*int*) – Number of channels in the input image.
- **out_channels** (*int*) – Number of channels produced by the convolution.
- **stride** (*int*) – Stride of the convolution.
- **kernel_size** (*int*) – Size of the convolving kernel. Default: 3.
- **affine** (*bool*) – A boolean value that when set to `True`, the batch-normalization layer has learnable affine parameters. Default: `True`.

Examples::

```
>>> m = ResNetBasicBlock(3, 10, 2, 3, True)
>>> input = torch.randn(3, 3, 32, 32)
>>> output = m(input)
```

expansion = 1

forward(*input: torch.Tensor*) → torch.Tensor

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.

training: bool

class extorch.nn.modules.block.**ResNetBottleneckBlock**(*in_channels: int*, *out_channels: int*, *stride: int*, *affine: bool = True*)

Bases: torch.nn.modules.module.Module

ResNet bottleneck block (Link).

Parameters

- **in_channels** (*int*) – Number of channels in the input image.
- **out_channels** (*int*) – Number of channels produced by the convolution.
- **stride** (*int*) – Stride of the convolution.
- **affine** (*bool*) – A boolean value that when set to True, the batch-normalization layer has learnable affine parameters. Default: True.

Examples::

```
>>> m = ResNetBottleneckBlock(10, 10, 2, True)
>>> input = torch.randn(2, 10, 32, 32)
>>> output = m(input)
```

expansion = 4

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

training: bool

3.3 extorch.nn.init

extorch.nn.init.**normal_**(*module: torch.nn.modules.module.Module*, *conv_mean: float = 0.0*, *conv_std: float = 1.0*, *bn_mean: float = 0.0*, *bn_std: float = 1.0*, *linear_mean: float = 0.0*, *linear_std: float = 1.0*) → None

Initialize the module with values drawn from the normal distribution.

Parameters

- **module** (*nn.Module*) – A pytorch module.
- **conv_mean** (*float*) – The mean of the normal distribution for convolution.
- **conv_std** (*float*) – The standard deviation of the normal distribution for convolution.
- **bn_mean** (*float*) – The mean of the normal distribution for batch-normalization.
- **bn_std** (*float*) – The standard deviation of the normal distribution for batch-normalization.
- **linear_mean** (*float*) – The mean of the normal distribution for linear layers.

- **bn_std** – The standard deviation of the normal distribution linear layers.

Examples::

```
>>> import torch.nn as nn
>>> module = nn.Sequential(
        nn.Conv2d(5, 10, 3),
        nn.BatchNorm2d(10),
        nn.ReLU()
    )
>>> module.apply(normal_)
```

3.4 extorch.nn.utils

class extorch.nn.utils.**WrapperModel**(*module: torch.nn.modules.module.Module*, *mean: torch.Tensor*, *std: torch.Tensor*)

Bases: torch.nn.modules.module.Module

A wrapper model for computer vision tasks. Normalize the input before feed-forward.

Parameters

- **module** (*nn.Module*) – A network with input range [0., 1.].
- **mean** (*Tensor*) – The mean value used for input transforms.
- **std** (*Tensor*) – The standard value used for input transforms.

forward(*input: torch.Tensor*) → torch.Tensor

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.

---

training: bool

extorch.nn.utils.**net_device**(*module: torch.nn.modules.module.Module*) → torch.device

Get current device of the network, assuming all weights of the network are on the same device.

Parameters **module** (*nn.Module*) – The network.

Returns The device.

Return type torch.device

Examples::

```
>>> module = nn.Conv2d(3, 3, 3)
>>> device = net_device(module) # "cpu"
```

extorch.nn.utils.**use_params**(*module: torch.nn.modules.module.Module*, *params: collections.OrderedDict*) → None

Replace the parameters in the module with the given ones. And then recover the old parameters.

Parameters

- **module** (*nn.Module*) – The targeted module.
- **params** (*OrderedDict*) – The given parameters.

Examples

```
>>> m = nn.Conv2d(1, 10, 3)
>>> params = m.state_dict()
>>> for p in params.values():
>>>     p.data = torch.zeros_like(p.data)
>>> input = torch.ones((2, 1, 10))
>>> with use_params(m, params):
>>>     output = m(input)
```

3.5 extorch.nn.optim

3.5.1 extorch.nn.optim.lr_scheduler

CosineWithRestarts	Cosine annealing with restarts (Link).

class extorch.nn.optim.lr_scheduler.**CosineWithRestarts**(*optimizer: torch.optim.optimizer.Optimizer*, *T_max: int*, *eta_min: float = 0.0*, *last_epoch: int = - 1*, *factor: float = 1.0*)

Bases: torch.optim.lr_scheduler._LRScheduler

Cosine annealing with restarts (Link).

Parameters

- **optimizer** (*Optimizer*) – The optimizer.
- **t_max** (*int*) – The maximum number of iterations within the first cycle.
- **eta_min** (*float*) – The minimum learning rate. Default: 0.
- **last_epoch** (*int*) – The index of the last epoch. This is used when restarting. Default: -1.
- **factor** (*float*) – The factor by which the cycle length (T_max) increases after each restart. Default: 1.

get_lr() → List[float]

Get updated learning rate.

Returns A list of current learning rates.

Return type lrs (List[float])

Note: We need to check if this is the first time `self.get_lr()` was called, since `torch.optim.lr_scheduler._LRScheduler` will call `self.get_lr()` when first initialized, but the learning rate should remain unchanged for the first epoch.

CHAPTER

FOUR

EXTORCH.ADVERSARIAL

4.1 extorch.adversarial.base

BaseAdversary	Base adversarial adversary.

class extorch.adversarial.base.**BaseAdversary**(*use_eval_mode: bool = False*)

Bases: torch.nn.modules.module.Module

Base adversarial adversary.

Parameters **use_eval_mode** (*bool*) – Whether use eval mode while generating adversarial examples. Default: False.

forward(*net: torch.nn.modules.module.Module*, *input: torch.Tensor*, *target: torch.Tensor*, *output: Optional[torch.Tensor] = None*) → torch.Tensor

Generate adversarial examples.

Parameters

- **net** (*nn.Module*) – The victim network.
- **input** (*Tensor*) – Origin input.
- **target** (*Tensor*) – Label of the input.
- **output** (*Tensor*) – Origin output. Default: None.

Returns The generated adversarial examples.

Return type adv_examples (Tensor)

abstract generate_adv(*net: torch.nn.modules.module.Module*, *input: torch.Tensor*, *target: torch.Tensor*, *output: torch.Tensor*) → torch.Tensor

Adversarial example generation.

Parameters

- **net** (*nn.Module*) – The victim network.
- **input** (*Tensor*) – Origin input.
- **target** (*Tensor*) – Label of the input.
- **output** (*Tensor*) – Origin output.

Returns The generated adversarial examples.

Return type adv_examples (Tensor)

training: bool

4.2 extorch.adversarial.gradient_based

GradientBasedAdversary	Gradient-based adversary.
PGDAdversary	Project Gradient Descent (PGD, `Link`_) adversarial adversary.
CustomizedPGDAdversary	Customized Project Gradient Descent (PGD, `Link`_) adversarial adversary.
FGSMAdversary	Fast Gradient Sign Method (FGSM, `Link`_) adversarial adversary.
MDIFGSMAdversary	Momentum Iterative Fast Gradient Sign Method with Input Diversity Method (M-DI-FGSM, 'Link'_).
MIFGSMAdversary	Momentum Iterative Fast Gradient Sign Method (MI-FGSM, `Link`_).
IFGSMAdversary	Iterative Fast Gradient Sign Method (I-FGSM, `Link`_).
DiversityLayer	Diversity input layer for M-DI-FGSM ('Link'_) adversarial adversary.

class extorch.adversarial.gradient_based.**CustomizedPGDAdversary**(*epsilon: float*, *n_step: int*, *step_size: float*, *rand_init: bool*, *mean: List[float]*, *std: List[float]*, *criterion: Optional[torch.nn.modules.loss._Loss] = CrossEntropyLoss()*, *use_eval_mode: bool = False*)

Bases: *extorch.adversarial.gradient_based.PGDAdversary*

Customized Project Gradient Descent (PGD, `Link`_) adversarial adversary.

generate_adv(*net: torch.nn.modules.module.Module*, *input: torch.Tensor*, *target: torch.Tensor*, *output: torch.Tensor*, *epsilon_c: float*) → torch.Tensor

Parameters **epsilon_c** (*float*) –

training: bool

class extorch.adversarial.gradient_based.**DiversityLayer**(*target: int*, *p: float = 0.5*)

Bases: torch.nn.modules.module.Module

Diversity input layer for M-DI-FGSM ('Link'_) adversarial adversary.

Parameters

- **target** (*int*) – Target reshape size.
- **p** (*float*) – Probability to apply the transformation. Default: 0.5.

forward(*input: torch.Tensor*) → torch.Tensor

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.

training: **bool**

class extorch.adversarial.gradient_based.**FGSMAdversary**(*epsilon: float*, *rand_init: bool*, *mean: List[float]*, *std: List[float]*, *criterion: Optional[torch.nn.modules.loss._Loss] = CrossEntropyLoss()*, *use_eval_mode: bool = False*)

Bases: *extorch.adversarial.gradient_based.PGDAdversary*

Fast Gradient Sign Method (FGSM, **`Link`_**) adversarial adversary.

Parameters

- **epsilon** (*float*) – Maximum distortion of adversarial example compared to origin input.
- **rand_init** (*bool*) – Whether add random perturbation to origin input before formal attack.
- **mean** (*List[float]*) – Sequence of means for each channel while normalizing the origin input.
- **std** (*List[float]*) – Sequence of standard deviations for each channel while normalizing the origin input.
- **criterion** (*Optional[_Loss]*) – Criterion to calculate the loss. Default: `nn.CrossEntropyLoss`.
- **use_eval_mode** (*bool*) – Whether use eval mode of the network while running attack. Default: `False`.

training: **bool**

class extorch.adversarial.gradient_based.**GradientBasedAdversary**(*criterion: Optional[torch.nn.modules.loss._Loss] = CrossEntropyLoss()*, *use_eval_mode: bool = False*)

Bases: *extorch.adversarial.base.BaseAdversary*

Gradient-based adversary.

Parameters

- **criterion** (*Optional[_Loss]*) – Criterion to calculate the loss. Default: `nn.CrossEntropyLoss`.
- **use_eval_mode** (*bool*) – Whether use eval mode of the network while running attack. Default: `False`.

training: **bool**

class extorch.adversarial.gradient_based.**IFGSMAdversary**(*epsilon: float*, *n_step: int*, *rand_init: bool*, *mean: List[float]*, *std: List[float]*, *step_size: Optional[float] = None*, *criterion: Optional[torch.nn.modules.loss._Loss] = CrossEntropyLoss()*, *use_eval_mode: bool = False*)

Bases: *extorch.adversarial.gradient_based.MIFGSMAdversary*

Iterative Fast Gradient Sign Method (I-FGSM, `Link`_).

Parameters

- **epsilon** (*float*) – Maximum distortion of adversarial example compared to origin input.
- **n_step** (*int*) – Number of attack iterations.
- **rand_init** (*bool*) – Whether add random perturbation to origin input before formal attack.
- **mean** (*List[float]*) – Sequence of means for each channel while normalizing the origin input.
- **std** (*List[float]*) – Sequence of standard deviations for each channel while normalizing the origin input.
- **step_size** (*Optional[float]*) – Step size for each attack iteration. If is not specified, `step_size = epsilon / n_step`. Default: `None`.
- **criterion** (*Optional[_Loss]*) – Criterion to calculate the loss. Default: `nn.CrossEntropyLoss`.
- **use_eval_mode** (*bool*) – Whether use eval mode of the network while running attack. Default: `False`.

training: **bool**

class extorch.adversarial.gradient_based.**MDIFGSMAdversary**(*epsilon: float*, *n_step: int*, *momentum: float*, *rand_init: bool*, *mean: List[float]*, *std: List[float]*, *diversity_p: float*, *target_size: int*, *step_size: Optional[float] = None*, *criterion: Optional[torch.nn.modules.loss._Loss] = CrossEntropyLoss()*, *use_eval_mode: bool = False*)

Bases: *extorch.adversarial.gradient_based.PGDAdversary*

Momentum Iterative Fast Gradient Sign Method with Input Diversity Method (M-DI-FGSM, 'Link'_).

Parameters

- **epsilon** (*float*) – Maximum distortion of adversarial example compared to origin input.
- **n_step** (*int*) – Number of attack iterations.
- **momentum** (*float*) – Gradient momentum.
- **rand_init** (*bool*) – Whether add random perturbation to origin input before formal attack.
- **mean** (*List[float]*) – Sequence of means for each channel while normalizing the origin input.
- **std** (*List[float]*) – Sequence of standard deviations for each channel while normalizing the origin input.
- **diversity_p** (*float*) – Probability to apply the input diversity transformation.
- **target_size** (*int*) – Randomly resized shape in the diversity input layer.
- **step_size** (*Optional[float]*) – Step size for each attack iteration. If is not specified, `step_size = epsilon / n_step`. Default: `None`.

- **criterion** (*Optional[_Loss]*) – Criterion to calculate the loss. Default: `nn.CrossEntropyLoss`.
- **use_eval_mode** (*bool*) – Whether use eval mode of the network while running attack. Default: `False`.

generate_adv(*net: torch.nn.modules.module.Module*, *input: torch.Tensor*, *target: torch.Tensor*, *output: torch.Tensor*) → torch.Tensor

Adversarial example generation.

Parameters

- **net** (*nn.Module*) – The victim network.
- **input** (*Tensor*) – Origin input.
- **target** (*Tensor*) – Label of the input.
- **output** (*Tensor*) – Origin output.

Returns The generated adversarial examples.

Return type adv_examples (Tensor)

training: **bool**

class extorch.adversarial.gradient_based.**MIFGSMAdversary**(*epsilon: float*, *n_step: int*, *momentum: float*, *rand_init: bool*, *mean: List[float]*, *std: List[float]*, *step_size: Optional[float] = None*, *criterion: Optional[torch.nn.modules.loss._Loss] = CrossEntropyLoss()*, *use_eval_mode: bool = False*)

Bases: *extorch.adversarial.gradient_based.MDIFGSMAdversary*

Momentum Iterative Fast Gradient Sign Method (MI-FGSM, `Link`_).

Parameters

- **epsilon** (*float*) – Maximum distortion of adversarial example compared to origin input.
- **n_step** (*int*) – Number of attack iterations.
- **momentum** (*float*) – Gradient momentum.
- **rand_init** (*bool*) – Whether add random perturbation to origin input before formal attack.
- **mean** (*List[float]*) – Sequence of means for each channel while normalizing the origin input.
- **std** (*List[float]*) – Sequence of standard deviations for each channel while normalizing the origin input.
- **step_size** (*Optional[float]*) – Step size for each attack iteration. If is not specified, `step_size = epsilon / n_step`. Default: `None`.
- **criterion** (*Optional[_Loss]*) – Criterion to calculate the loss. Default: `nn.CrossEntropyLoss`.
- **use_eval_mode** (*bool*) – Whether use eval mode of the network while running attack. Default: `False`.

training: **bool**

class extorch.adversarial.gradient_based.**PGDAdversary**(*epsilon: float*, *n_step: int*, *step_size: float*, *rand_init: bool*, *mean: List[float]*, *std: List[float]*, *criterion: Optional[torch.nn.modules.loss._Loss] = CrossEntropyLoss()*, *use_eval_mode: bool = False*)

Bases: *extorch.adversarial.gradient_based.GradientBasedAdversary*

Project Gradient Descent (PGD, `Link`_) adversarial adversary.

Parameters

- **epsilon** (*float*) – Maximum distortion of adversarial example compared to origin input.
- **n_step** (*int*) – Number of attack iterations.
- **step_size** (*float*) – Step size for each attack iteration.
- **rand_init** (*bool*) – Whether add random perturbation to origin input before formal attack.
- **mean** (*List[float]*) – Sequence of means for each channel while normalizing the origin input.
- **std** (*List[float]*) – Sequence of standard deviations for each channel while normalizing the origin input.
- **criterion** (*Optional[_Loss]*) – Criterion to calculate the loss. Default: `nn.CrossEntropyLoss`.
- **use_eval_mode** (*bool*) – Whether use eval mode of the network while running attack. Default: `False`.

generate_adv(*net: torch.nn.modules.module.Module*, *input: torch.Tensor*, *target: torch.Tensor*, *output: torch.Tensor*) → torch.Tensor

Adversarial example generation.

Parameters

- **net** (*nn.Module*) – The victim network.
- **input** (*Tensor*) – Origin input.
- **target** (*Tensor*) – Label of the input.
- **output** (*Tensor*) – Origin output.

Returns The generated adversarial examples.

Return type adv_examples (Tensor)

training: bool

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

extorch.adversarial.base, 33
extorch.adversarial.gradient_based, 34
extorch.nn.functional, 17
extorch.nn.init, 28
extorch.nn.modules.auxiliary, 18
extorch.nn.modules.block, 27
extorch.nn.modules.loss, 19
extorch.nn.modules.mlp, 21
extorch.nn.modules.operation, 22
extorch.nn.optim.lr_scheduler, 30
extorch.nn.utils, 29
extorch.utils.common, 11
extorch.utils.exception, 13
extorch.utils.logger, 14
extorch.utils.stats, 14
extorch.utils.visual, 15
extorch.vision.dataset, 3
extorch.vision.transforms.detection, 10
extorch.vision.transforms.functional, 5
extorch.vision.transforms.segmentation, 7
extorch.vision.transforms.transforms, 6

INDEX

A

abstractclassmethod (*class in extorch.utils.common*), 11
accuracy() (*in module extorch.utils.stats*), 14
AdaptiveCenterCrop (*class in extorch.vision.transforms.transforms*), 6
AdaptiveRandomCrop (*class in extorch.vision.transforms.transforms*), 6
AuxiliaryHead (*class in extorch.nn.modules.auxiliary*), 18
AuxiliaryHeadImageNet (*class in extorch.nn.modules.auxiliary*), 18
average_logits() (*in module extorch.nn.functional*), 17
AverageMeter (*class in extorch.utils.stats*), 14
avgs() (*extorch.utils.stats.OrderedStats method*), 14

B

BaseAdversary (*class in extorch.adversarial.base*), 33
BaseDataset (*class in extorch.vision.dataset*), 3
BNReLU (*class in extorch.nn.modules.operation*), 22

C

cal_flops() (*in module extorch.utils.stats*), 14
CIFAR10 (*class in extorch.vision.dataset*), 4
CIFAR100 (*class in extorch.vision.dataset*), 4
compute() (*extorch.utils.stats.SegConfusionMatrix method*), 14
ConvBN (*class in extorch.nn.modules.operation*), 22
ConvBNReLU (*class in extorch.nn.modules.operation*), 23
ConvReLU (*class in extorch.nn.modules.operation*), 24
CosineWithRestarts (*class in extorch.nn.optim.lr_scheduler*), 30
CrossEntropyLabelSmooth (*class in extorch.nn.modules.loss*), 19
CrossEntropyMixupLoss (*class in extorch.nn.modules.loss*), 19
CustomizedPGDAdversary (*class in extorch.adversarial.gradient_based*), 34
Cutout (*class in extorch.vision.transforms.transforms*), 7
cutout() (*in module extorch.vision.transforms.functional*), 5
CVClassificationDataset (*class in extorch.vision.dataset*), 3
CVDataset (*class in extorch.vision.dataset*), 3

D

dec_soft_assignment() (*in module extorch.nn.functional*), 17
DECLoss (*class in extorch.nn.modules.loss*), 20
denormalize() (*in module extorch.utils.visual*), 15
DetCompose (*class in extorch.vision.transforms.detection*), 10
DiversityLayer (*class in extorch.adversarial.gradient_based*), 34

E

expansion (*extorch.nn.modules.block.ResNetBasicBlock attribute*), 27
expansion (*extorch.nn.modules.block.ResNetBottleneckBlock attribute*), 28
expect() (*in module extorch.utils.exception*), 13
extorch.adversarial.base
    module, 33
extorch.adversarial.gradient_based
    module, 34
extorch.nn.functional
    module, 17
extorch.nn.init
    module, 28
extorch.nn.modules.auxiliary
    module, 18
extorch.nn.modules.block
    module, 27
extorch.nn.modules.loss
    module, 19
extorch.nn.modules.mlp
    module, 21
extorch.nn.modules.operation
    module, 22
extorch.nn.optim.lr_scheduler
    module, 30
extorch.nn.utils
    module, 29

extorch.utils.common
    module, 11
extorch.utils.exception
    module, 13
extorch.utils.logger
    module, 14
extorch.utils.stats
    module, 14
extorch.utils.visual
    module, 15
extorch.vision.dataset
    module, 3
extorch.vision.transforms.detection
    module, 10
extorch.vision.transforms.functional
    module, 5
extorch.vision.transforms.segmentation
    module, 7
extorch.vision.transforms.transforms
    module, 6
ExtorchException, 13

F

FashionMNIST (*class in extorch.vision.dataset*), 5
FGSMAdversary (*class in extorch.adversarial.gradient_based*), 35
forward() (*extorch.adversarial.base.BaseAdversary method*), 33
forward() (*extorch.adversarial.gradient_based.DiversityLayer method*), 34
forward() (*extorch.nn.modules.auxiliary.AuxiliaryHead method*), 18
forward() (*extorch.nn.modules.auxiliary.AuxiliaryHeadImageNet method*), 19
forward() (*extorch.nn.modules.block.ResNetBasicBlock method*), 27
forward() (*extorch.nn.modules.block.ResNetBottleneckBlock method*), 28
forward() (*extorch.nn.modules.loss.CrossEntropyLabelSmooth method*), 19
forward() (*extorch.nn.modules.loss.CrossEntropyMixupLoss method*), 20
forward() (*extorch.nn.modules.loss.DECLoss method*), 20
forward() (*extorch.nn.modules.loss.HintonKDLoss method*), 21
forward() (*extorch.nn.modules.mlp.MLP method*), 21
forward() (*extorch.nn.modules.operation.BNReLU method*), 22
forward() (*extorch.nn.modules.operation.ConvBN method*), 23
forward() (*extorch.nn.modules.operation.ConvBNReLU method*), 24
forward() (*extorch.nn.modules.operation.ConvReLU method*), 25
forward() (*extorch.nn.modules.operation.Identity method*), 25
forward() (*extorch.nn.modules.operation.ReLUBN method*), 25
forward() (*extorch.nn.modules.operation.ReLUConvBN method*), 26
forward() (*extorch.nn.utils.WrapperModel method*), 29
forward() (*extorch.vision.transforms.segmentation.SegCenterCrop method*), 7
forward() (*extorch.vision.transforms.segmentation.SegConvertImageDtyp method*), 8
forward() (*extorch.vision.transforms.segmentation.SegNormalize method*), 8
forward() (*extorch.vision.transforms.segmentation.SegPILToTensor method*), 8
forward() (*extorch.vision.transforms.segmentation.SegRandomCrop method*), 9
forward() (*extorch.vision.transforms.segmentation.SegRandomHorizontal method*), 9
forward() (*extorch.vision.transforms.segmentation.SegRandomResize method*), 10
forward() (*extorch.vision.transforms.segmentation.SegResize method*), 10
forward() (*extorch.vision.transforms.transforms.AdaptiveCenterCrop method*), 6
forward() (*extorch.vision.transforms.transforms.AdaptiveRandomCrop method*), 6
forward() (*extorch.vision.transforms.transforms.Cutout method*), 7

G

generate_adv() (*extorch.adversarial.base.BaseAdversary method*), 33
generate_adv() (*extorch.adversarial.gradient_based.CustomizedPGDAd method*), 34
generate_adv() (*extorch.adversarial.gradient_based.MDIFGSMAdversa method*), 37
generate_adv() (*extorch.adversarial.gradient_based.PGDAdversary method*), 38
get_lr() (*extorch.nn.optim.lr_scheduler.CosineWithRestarts method*), 30
get_params() (*in module extorch.utils.stats*), 15
getLogger() (*in module extorch.utils.logger*), 14
GradientBasedAdversary (*class in extorch.adversarial.gradient_based*), 35

H

HintonKDLoss (*class in extorch.nn.modules.loss*), 20

I

Identity (*class in extorch.nn.modules.operation*), 25

IFGSMAdversary (*class in extorch.adversarial.gradient_based*), 35
InvalidConfigException, 13
InvalidValueException, 13
is_empty() (*extorch.utils.stats.AverageMeter method*), 14
items() (*extorch.utils.stats.OrderedStats method*), 14

M

makedir() (*in module extorch.utils.common*), 11
MDIFGSMAdversary (*class in extorch.adversarial.gradient_based*), 36
MIFGSMAdversary (*class in extorch.adversarial.gradient_based*), 37
mix_data() (*in module extorch.nn.functional*), 17
MLP (*class in extorch.nn.modules.mlp*), 21
MNIST (*class in extorch.vision.dataset*), 4
module
    extorch.adversarial.base, 33
    extorch.adversarial.gradient_based, 34
    extorch.nn.functional, 17
    extorch.nn.init, 28
    extorch.nn.modules.auxiliary, 18
    extorch.nn.modules.block, 27
    extorch.nn.modules.loss, 19
    extorch.nn.modules.mlp, 21
    extorch.nn.modules.operation, 22
    extorch.nn.optim.lr_scheduler, 30
    extorch.nn.utils, 29
    extorch.utils.common, 11
    extorch.utils.exception, 13
    extorch.utils.logger, 14
    extorch.utils.stats, 14
    extorch.utils.visual, 15
    extorch.vision.dataset, 3
    extorch.vision.transforms.detection, 10
    extorch.vision.transforms.functional, 5
    extorch.vision.transforms.segmentation, 7
    extorch.vision.transforms.transforms, 6

N

net_device() (*in module extorch.nn.utils*), 29
normal_() (*in module extorch.nn.init*), 28
nullcontext() (*in module extorch.utils.common*), 11

O

OrderedStats (*class in extorch.utils.stats*), 14

P

PGDAdversary (*class in extorch.adversarial.gradient_based*), 37

R

reduction (*extorch.nn.modules.loss.HintonKDLoss attribute*), 21
ReLUBN (*class in extorch.nn.modules.operation*), 25
ReLUConvBN (*class in extorch.nn.modules.operation*), 26
remove_dirs() (*in module extorch.utils.common*), 11
remove_files() (*in module extorch.utils.common*), 12
remove_targets() (*in module extorch.utils.common*), 12
reset() (*extorch.utils.stats.AverageMeter method*), 14
reset() (*extorch.utils.stats.SegConfusionMatrix method*), 14
ResNetBasicBlock (*class in extorch.nn.modules.block*), 27
ResNetBottleneckBlock (*class in extorch.nn.modules.block*), 27
run_processes() (*in module extorch.utils.common*), 12

S

SegCenterCrop (*class in extorch.vision.transforms.segmentation*), 7
SegCompose (*class in extorch.vision.transforms.segmentation*), 8
SegConfusionMatrix (*class in extorch.utils.stats*), 14
SegConvertImageDtype (*class in extorch.vision.transforms.segmentation*), 8
SegmentationDataset (*class in extorch.vision.dataset*), 4
SegNormalize (*class in extorch.vision.transforms.segmentation*), 8
SegPILToTensor (*class in extorch.vision.transforms.segmentation*), 8
SegRandomCrop (*class in extorch.vision.transforms.segmentation*), 9
SegRandomHorizontalFlip (*class in extorch.vision.transforms.segmentation*), 9
SegRandomResize (*class in extorch.vision.transforms.segmentation*), 9
SegResize (*class in extorch.vision.transforms.segmentation*), 10
set_seed() (*in module extorch.utils.common*), 13
set_trace() (*in module extorch.utils.common*), 13
soft_voting() (*in module extorch.nn.functional*), 18
step() (*extorch.utils.common.TimeEstimator method*), 11
SVHN (*class in extorch.vision.dataset*), 5

T

target_distribution() (*extorch.nn.modules.loss.DECLoss static method*), 20
TimeEstimator (*class in extorch.utils.common*), 11
TinyImageNet (*class in extorch.vision.dataset*), 5

training (*extorch.adversarial.base.BaseAdversary attribute*), 33
training (*extorch.adversarial.gradient_based.CustomizedPGDAdversary attribute*), 34
training (*extorch.adversarial.gradient_based.DiversityLayer attribute*), 35
training (*extorch.adversarial.gradient_based.FGSMAdversary attribute*), 35
training (*extorch.adversarial.gradient_based.GradientBaseAdversary attribute*), 35
training (*extorch.adversarial.gradient_based.IFGSMAdversary attribute*), 36
training (*extorch.adversarial.gradient_based.MDIFGSMAdversary attribute*), 37
training (*extorch.adversarial.gradient_based.MIFGSMAdversary attribute*), 37
training (*extorch.adversarial.gradient_based.PGDAdversary attribute*), 38
training (*extorch.nn.modules.auxiliary.AuxiliaryHead attribute*), 18
training (*extorch.nn.modules.auxiliary.AuxiliaryHeadImageNet attribute*), 19
training (*extorch.nn.modules.block.ResNetBasicBlock attribute*), 27
training (*extorch.nn.modules.block.ResNetBottleneckBlock attribute*), 28
training (*extorch.nn.modules.loss.CrossEntropyLabelSmooth attribute*), 19
training (*extorch.nn.modules.loss.CrossEntropyMixupLoss attribute*), 20
training (*extorch.nn.modules.loss.DECLoss attribute*), 20
training (*extorch.nn.modules.mlp.MLP attribute*), 21
training (*extorch.nn.modules.operation.BNReLU attribute*), 22
training (*extorch.nn.modules.operation.ConvBN attribute*), 23
training (*extorch.nn.modules.operation.ConvBNReLU attribute*), 24
training (*extorch.nn.modules.operation.ConvReLU attribute*), 25
training (*extorch.nn.modules.operation.Identity attribute*), 25
training (*extorch.nn.modules.operation.ReLUBN attribute*), 26
training (*extorch.nn.modules.operation.ReLUConvBN attribute*), 27
training (*extorch.nn.utils.WrapperModel attribute*), 29
training (*extorch.vision.transforms.segmentation.SegCenterCrop attribute*), 8
training (*extorch.vision.transforms.segmentation.SegConvertImageDtype attribute*), 8
training (*extorch.vision.transforms.segmentation.SegNormalize attribute*), 8
training (*extorch.vision.transforms.segmentation.SegPILToTensor attribute*), 9
training (*extorch.vision.transforms.segmentation.SegRandomCrop attribute*), 9
training (*extorch.vision.transforms.segmentation.SegRandomHorizontalF attribute*), 9
training (*extorch.vision.transforms.segmentation.SegRandomResize attribute*), 10
training (*extorch.vision.transforms.segmentation.SegResize attribute*), 10
training (*extorch.vision.transforms.transforms.AdaptiveCenterCrop attribute*), 6
training (*extorch.vision.transforms.transforms.AdaptiveRandomCrop attribute*), 6
training (*extorch.vision.transforms.transforms.Cutout attribute*), 7
tsne_fit() (*in module extorch.utils.visual*), 15

U

update() (*extorch.utils.stats.AverageMeter method*), 14
update() (*extorch.utils.stats.OrderedStats method*), 14
update() (*extorch.utils.stats.SegConfusionMatrix method*), 14
use_params() (*in module extorch.nn.utils*), 29

V

VOCSegmentationDataset (*class in extorch.vision.dataset*), 5

W

WrapperModel (*class in extorch.nn.utils*), 29

Y

yaml_dump() (*in module extorch.utils.common*), 13
yaml_load() (*in module extorch.utils.common*), 13