PyMIC
Release 0.1

HiLab

Sep 30, 2022
# GETTING STARTED

## 1 Installation

## 2 Usage

2.1 Quick Start .............................................. 5
2.2 Fully Supervised Learning .................................. 8
2.3 Semi-Supervised Learning .................................. 14
2.4 Weakly-Supervised Learning ................................. 16
2.5 Noisy Label Learning ...................................... 18

## 3 API

3.1 pymic.io package ........................................ 21
3.2 pymic.layer package ...................................... 23
3.3 pymic.loss package ....................................... 27
3.4 pymic.net package ........................................ 41
3.5 pymic.net_run package .................................... 66
3.6 pymic.net_run_nll package ................................. 72
3.7 pymic.net_run_ssl package ................................. 76
3.8 pymic.net_run_wsl package ................................. 81
3.9 pymic.transform package ................................. 84
3.10 pymic.util package ....................................... 92

## 4 Citation

99

Python Module Index .............................. 101

Index .............................................. 103
PyMIC is a pytorch-based toolkit for medical image computing with annotation-efficient deep learning. PyMIC is developed to support learning with imperfect labels, including semi-supervised and weakly supervised learning, and learning with noisy annotations.

Check out the Installation section for install PyMIC, and go to the Usage section for understanding modules for the segmentation pipeline designed in PyMIC. Please follow PyMIC_examples to quickly start with using PyMIC.

**Note:** This project is under active development. It will be updated later.
Install PyMIC using pip (e.g., within a Python virtual environment):

```
pip install PYMIC
```

Alternatively, you can download or clone the code from GitHub and install PyMIC by

```
git clone https://github.com/HiLab-git/PyMIC
cd PyMIC
python setup.py install
```

PyMIC requires Python 3.6 (or higher) and depends on the following packages:

- pandas
- h5py
- NumPy
- scikit-image
- SciPy
- SimpleITK
This usage gives details of how to use PyMIC. Beginners can easily start with training a deep learning model with configure files. When you are more familiar with the PyMIC pipeline, you can define your customized modules and reuse the remaining parts of the pipeline, with minimal workload.

2.1 Quick Start

2.1.1 Train and Test

PyMIC accepts a configuration file for running. For example, to train a network for segmentation with full supervision, run the following command:

```
pymic_run train myconfig.cfg
```

After training, run the following command for testing:

```
pymic_run test myconfig.cfg
```

Tip: We provide several examples in PyMIC_examples. Please run these examples to quickly start with using PyMIC.

2.1.2 Configuration File

PyMIC uses configuration files to specify the setting and parameters of a deep learning pipeline, so that users can reuse the code and minimize their workload. Users can use configuration files to config almost all the components involved, such as dataset, network structure, loss function, optimizer, learning rate scheduler and post processing methods, etc.

Note: Generally, the configuration file have four sections: dataset, network, training and testing.

The following is an example configuration file used for segmentation of lung from radiograph, which can be find in PyMIC_examples/segmentation/JSRT.

```
[dataset]
# tensor type (float or double)
tensor_type = float
task_type = seg
root_dir = ../../PyMIC_data/JSRT
```

(continues on next page)
```python
train_csv = config/jsrt_train.csv
valid_csv = config/jsrt_valid.csv
test_csv = config/jsrt_test.csv
train_batch_size = 4

# data transforms
train_transform = [NormalizeWithMeanStd, RandomCrop, LabelConvert, LabelToProbability]
valid_transform = [NormalizeWithMeanStd, LabelConvert, LabelToProbability]
test_transform = [NormalizeWithMeanStd]

NormalizeWithMeanStd_channels = [0]
RandomCrop_output_size = [240, 240]

LabelConvert_source_list = [0, 255]
LabelConvert_target_list = [0, 1]

[network]
net_type = UNet2D
# Parameters for UNet2D
class_num = 2
in_chns = 1
feature_chns = [16, 32, 64, 128, 256]
dropout = [0, 0, 0.3, 0.4, 0.5]
bilinear = False
deep_supervise = False

[training]
# list of gpus
gpus = [0]
loss_type = DiceLoss

# for optimizers
optimizer = Adam
learning_rate = 1e-3
momentum = 0.9
weight_decay = 1e-5

# for lr scheduler (MultiStepLR)
lr_scheduler = MultiStepLR
lr_gamma = 0.5
lr_milestones = [2000, 4000, 6000]

ckpt_save_dir = model/unet_dice_loss
ckpt_prefix = unet

# start iter
iter_start = 0
iter_max = 8000
iter_valid = 200
iter_save = 8000
```

(continues on next page)
# list of gpus

gpus = [0]

# checkpoint mode can be [0-latest, 1-best, 2-specified]
ckpt_mode = 0
output_dir = result

# convert the label of prediction output
label_source = [0, 1]
label_target = [0, 255]

## 2.1.3 Evaluation

To evaluate a model’s prediction results compared with the ground truth, use the pymic_eval_seg and pymic_eval_cls commands for segmentation and classification tasks, respectively. Both of them accept a configuration file to specify the evaluation metrics, predicted results, ground truth and other information.

For example, for segmentation tasks, run:

```
pymic_eval_seg evaluation.cfg
```

The configuration file is like (an example from PYMIC_examples/seg_ssl/ACDC):

```ini
[evaluation]
metric = dice
label_list = [1,2,3]
organ_name = heart

ground_truth_folder_root = ../../PyMIC_data/ACDC/preprocess
segmentation_folder_root = result/unet2d_em
evaluation_image_pair = config/data/image_test_gt_seg.csv
```

See `pymic.util.evaluation_seg.evaluation` for details of the configuration required.

For classification tasks, run:

```
pymic_eval_cls evaluation.cfg
```

The configuration file is like (an example from PYMIC_examples/classification/CHNCXR):

```ini
[evaluation]
metric_list = [accuracy, auc]
ground_truth_csv = config/cxr_test.csv
predict_csv = result/resnet18.csv
predict_prob_csv = result/resnet18_prob.csv
```

See `pymic.util.evaluation_cls.main` for details of the configuration required.
2.2 Fully Supervised Learning

2.2.1 SegmentationAgent

`pymic.net_run.agent_seg.SegmentationAgent` is the general class used for training and inference of deep learning models. You just need to specify a configuration file to initialize an instance of that class. An example code to use it is:

```python
from pymic.util.parse_config import *
from pymic.net_run.agent_seg import SegmentationAgent

config_name = "a_config_file.cfg"
config = parse_config(config_name)
config = synchronize_config(config)
stage = "train"  # or "test"
agent = SegmentationAgent(config, stage)
agent.run()
```

The above code will use the dataset, network and loss function, etc specified in the configuration file for running.

Tip: If you use the built-in modules such as UNet and Dice + CrossEntropy loss for segmentation, you don’t need to write the above code. Just just use the `pymic_run` command.

2.2.2 Dataset

PyMIC provides two types of datasets for loading images from disk to memory: `NiftyDataset` and `H5DataSet`. `NiftyDataset` is designed for 2D and 3D images in common formats such as png, jpeg, bmp and nii.gz. `H5DataSet` is used for hdf5 data that are more efficient to load.

To use `NiftyDataset`, users need to specify the root path of the dataset and the csv file storing the image and label file names. The configurations include the following items:

- `tensor_type`: data type for tensors. Should be `float` or `double`.
- `task_type`: should be `seg` for segmentation tasks.
- `root_dir` (string): the root dir of dataset.
- `modal_num` (int, default is 1): modalities number. For images with N modalities, each modality should be saved in an independent file.
- `train_csv` (string): the path of csv file for training set.
- `valid_csv` (string): the path of csv file for validation set.
- `test_csv` (string): the path of csv file for testing set.
- `train_batch_size` (int): the batch size for training set.
- `valid_batch_size` (int, optional): the batch size for validation set. The default value is set as `train_batch_size`.
- `test_batch_size` (int, optional): the batch size for testing set. The default value is 1.

The csv file should have at least two columns (fields), one for `image` and the other for `label`. If the input image have multiple modalities with each modality saved in a single file, then the csv file should have N + 1 columns, where the first
N columns are for the N modalities, and the last column is for the label. The following is an example for configuration of dataset.

```
[dataset]
# tensor type (float or double)
tensor_type = float
task_type = seg
root_dir = ../../PyMIC_data/JSRT
train_csv = config/jsrt_train.csv
valid_csv = config/jsrt_valid.csv
test_csv = config/jsrt_test.csv
train_batch_size = 4
```

To use your own dataset, you can define a dataset as a child class of `NiftyDataset`, `H5DataSet`, or `torch.utils.data.Dataset`, and use `SegmentationAgent.set_datasets()` to set the customized datasets. For example:

```python
from torch.utils.data import Dataset
from pymic.net_run.agent_seg import SegmentationAgent

class MyDataset(Dataset):
    ...
    # define your custom dataset here

trainset, valset, testset = MyDataset(...), MyDataset(...), MyDataset(...)
agent = SegmentationAgent(config, stage)
agent.set_datasets(trainset, valset, testset)
agent.run()
```

### 2.2.3 Transforms

Several transforms are defined in PyMIC to preprocess or augment the data before sending it to the network. The `TransformDict` in `pymic.transform.trans_dict` lists all the built-in transforms supported in PyMIC.

In the configuration file, users can specify the transforms required for training, validation and testing data, respectively. The parameters of each transform class should also be provided, such as following:

```
# data transforms
train_transform = [Pad, RandomRotate, RandomCrop, RandomFlip, NormalizeWithMeanStd,
                   GammaCorrection, GaussianNoise, LabelToProbability]
valid_transform = [NormalizeWithMeanStd, Pad, LabelToProbability]
test_transform = [NormalizeWithMeanStd, Pad]

# the inverse transform will be enabled during testing
Pad_output_size = [8, 256, 256]
Pad_ceil_mode = False
Pad_inverse = True
RandomRotate_angle_range_d = [-90, 90]
RandomRotate_angle_range_h = None
RandomRotate_angle_range_w = None
RandomCrop_output_size = [6, 192, 192]
RandomCrop_foreground_focus = False
```

(continues on next page)
For spatial transforms, you can specify whether an inverse transform is enabled or not. Setting the inverse flag as True will transform the prediction output inversely during testing, such as `Pad_inverse = True` shown above. If you want to make images with different shapes to have the same shape before testing, then the corresponding transform’s inverse flag can be set as True, so that the prediction output will be transformed back to the original image space. This is also useful for test time augmentation.

You can also define your own transform operations. To integrate your customized transform to the PyMIC pipeline, just add it to the `TransformDict`, and you can also specify the parameters via a configuration file for the customized transform. The following is some example code for this:

```python
from pymic.transform.trans_dict import TransformDict
from pymic.transform.abstract_transform import AbstractTransform
from pymic.net_run.agent_seg import SegmentationAgent

# customized transform
class MyTransform(AbstractTransform):
    def __init__(self, params):
        super(MyTransform, self).__init__(params)
        ...
    def __call__(self, sample):
        ...
    def inverse_transform_for_prediction(self, sample):
        ...

my_trans_dict = TransformDict
my_trans_dict["MyTransform"] = MyTransform
agent = SegmentationAgent(config, stage)
agent.set_transform_dict(my_trans_dict)
agent.run()
```
2.2.4 Networks

The configuration file has a network section to specify the network’s type and hyper-parameters. For example, the following is a configuration for using 2DUNet:

```yaml
[network]
net_type = UNet2D
# Parameters for UNet2D
class_num = 2
in_chns = 1
feature_chns = [16, 32, 64, 128, 256]
dropout = [0, 0, 0.3, 0.4, 0.5]
bilinear = False
deep_supervise = False
```

The SegNetDict in `pymic.net.net_dict_seg` lists all the built-in network structures currently implemented in PyMIC.

You can also define your own networks. To integrate your customized network to the PyMIC pipeline, just call set_network() of SegmentationAgent. The following is some example code for this:

```python
import torch.nn as nn
from pymic.net_run.agent_seg import SegmentationAgent

# customized network
class MyNetwork(nn.Module):
    def __init__(self, params):
        super(MyNetwork, self).__init__()
    ...
    def forward(self, x):
        ...

net = MyNetwork(params)
agent = SegmentationAgent(config, stage)
agent.set_network(net)
agent.run()
```

2.2.5 Loss Functions

The setting of loss function is in the training section of the configuration file, where the loss function name and hyper-parameters should be provided. The SegLossDict in `pymic.loss.loss_dict_seg` lists all the built-in loss functions currently implemented in PyMIC.

The following is an example of the setting of loss:

```yaml
loss_type = DiceLoss
loss_softmax = True
```

Note that PyMIC supports using a combination of loss functions. Just set loss_type as a list of loss functions, and use loss_weight to specify the weight of each loss, such as the following:

```yaml
loss_type = [DiceLoss, CrossEntropyLoss]
loss_weight = [0.5, 0.5]
```
You can also define your own loss functions. To integrate your customized loss function to the PyMIC pipeline, just add it to the SegLossDict, and you can also specify the parameters via a configuration file for the customized loss. The following is some example code for this:

```python
from pymic.loss.loss_dict_seg import SegLossDict
from pymic.net_run.agent_seg import SegmentationAgent

# customized loss
class MyLoss(nn.Module):
    def __init__(self, params = None):
        super(MyLoss, self).__init__()
        ...

    def forward(self, loss_input_dict):
        ...

my_loss_dict = SegLossDict
my_loss_dict["MyLoss"] = MyLoss
agent = SegmentationAgent(config, stage)
agent.set_loss_dict(my_loss_dict)
agent.run()
```

### 2.2.6 Training Options

In addition to the loss function, users can specify several training options in the training section of the configuration file.

#### Iterations

For training iterations, the following parameters need to be specified in the configuration file:

- **iter_start**: the start iteration, by default is 0. None zero value means the iteration where a pre-trained model stopped for continuing with the training.
- **iter_max**: the maximal allowed iteration for training.
- **iter_valid**: if the value is K, it means evaluating the performance on the validation set for every K steps.
- **iter_save**: The iterations for saving the model. If the value is k, it means the model will be saved every k iterations. It can also be a list of integer numbers, which specifies the iterations to save the model.
- **early_stop_patience**: if the value is k, it means the training will stop when the performance on the validation set does not improve for k iterations.

#### Optimizer

For optimizer, users need to set optimizer, learning_rate, momentum and weight_decay. The built-in optimizers include SGD, Adam, SparseAdam, Adadelta, Adagrad, Adamax, ASGD, LBFGS, RMSprop and Rprop that are implemented in torch.optim.

You can also use customized optimizers via SegmentationAgent.set_optimizer().
Learning Rate Scheduler

The current built-in learning rate schedulers are ReduceLROnPlateau and MultiStepLR, which can be specified in lr_scheduler in the configuration file.

Parameters related to ReduceLROnPlateau include lr_gamma. Parameters related to MultiStepLR include lr_gamma and lr_milestones.

You can also use customized lr schedulers via SegmentationAgent.set_scheduler().

Other Options

Other options for training include:

- **gpus**: a list of GPU index for training the model. If the length is larger than one (such as [0, 1]), it means the model will be trained on multiple GPUs parallelly.
- **deterministic** (bool, default is True): set the training deterministic or not.
- **random_seed** (int, optional): the random seed customized by user. Default value is 1.
- **ckpt_save_dir**: the path to the folder for saving the trained models.
- **ckpt_prefix**: the prefix of the name to save the checkpoints.

2.2.7 Inference Options

There are several options for inference after training the model. You can also select the GPUs for testing, enable sliding window inference or inference with test-time augmentation, etc. The following is a list of options available for inference:

- **gpus**: a list of GPU index. Actually, only the first GPU in the list is used.
- **evaluation_mode** (bool, default is True): set the model to evaluation mode or not.
- **test_time_dropout** (bool, default is False): use test-time dropout or not.
- **ckpt_mode** (int): which checkpoint is used. 0–the last checkpoint; 1–the checkpoint with the best performance on the validation set; 2–a specified checkpoint.
- **ckpt_name** (string, optional): the full path to the checkpoint if ckpt_mode = 2.
- **post_process** (string, default is None): the post process method after inference. The current available post processing is PostKeepLargestComponent. Users can also specify customized post process methods via SegmentationAgent.set_postprocessor().
- **sliding_window_enable** (bool, default is False): use sliding window for inference or not.
- **sliding_window_size** (optional): a list for sliding window size when sliding_window_enable = True.
- **sliding_window_stride** (optional): a list for sliding window stride when sliding_window_enable = True.
- **tta_mode** (int, default is 0): the mode for Test Time Augmentation (TTA). 0–not using TTA; 1–using TTA based on horizontal and vertical flipping.
- **output_dir** (string): the dir to save the prediction output.
- **ignore_dir** (bool, default is True): if the input image name has a /, it will be replaced with _ in the output file name.
- **save_probability** (bool, default is False): save the output probability for each class.
- **label_source** (list, default is None): a list of label to be converted after prediction. For example, label_source = [0, 1] and label_target = [0, 255] will convert label value from 1 to 255.
- label_target (list, default is None): a list of label after conversion. Use this with label_source.
- filename_replace_source (string, default is None): the substring in the filename will be replaced with a new substring specified by filename_replace_target.
- filename_replace_target (string, default is None): work with filename_replace_source.

2.3 Semi-Supervised Learning

2.3.1 pymic_ssl

pymic_ssl is the command for using built-in semi-supervised methods for training. Similarly to pymic_run, it should be followed by two parameters, specifying the stage and configuration file, respectively. The training and testing commands are:

```
pymic_ssl train myconfig_ssl.cfg
pymic_ssl test myconfig_ssl.cfg
```

Tip: If the SSL method only involves one network, either pymic_ssl or pymic_run can be used for inference. Their difference only exists in the training stage.

2.3.2 SSL Configurations

In the configuration file for pymic_ssl, in addition to those used in fully supervised learning, there are some items specificated for semi-supervised learning.

Users should provide values for the following items in dataset section of the configuration file:

- train_csv_unlab (string): the csv file for unlabeled dataset. Note that train_csv is only used for labeled dataset.
- train_batch_size_unlab (int): the batch size for unlabeled dataset. Note that train_batch_size means the batch size for the labeled dataset.
- train_transform_unlab (list): a list of transforms used for unlabeled data.

The following is an example of the dataset section for semi-supervised learning:

```python
... root_dir =../../PyMIC_data/ACDC/preprocess/
train_csv = config/data/image_train_r10_lab.csv
train_csv_unlab = config/data/image_train_r10_unlab.csv
valid_csv = config/data/image_valid.csv
test_csv = config/data/image_test.csv
train_batch_size = 4
train_batch_size_unlab = 4

# data transforms
train_transform = [Pad, RandomRotate, RandomCrop, RandomFlip, NormalizeWithMeanStd,
GammaCorrection, GaussianNoise, LabelToProbability]
train_transform_unlab = [Pad, RandomRotate, RandomCrop, RandomFlip, NormalizeWithMeanStd,
(continues on next page)```
In addition, there is a semi_supervised_learning section that is specifically designed for SSL methods. In that section, users need to specify the ssl_method and configurations related to the SSL method. For example, the corresponding configuration for CPS is:

```python
[semi_supervised_learning]
ssl_method = CPS
regularize_w = 0.1
rampup_start = 1000
rampup_end = 20000
...
```

Note: The configuration items vary with different SSL methods. Please refer to the API of each built-in SSL method for details of the corresponding configuration.

### 2.3.3 Built-in SSL Methods

`pymic.net_run.ssl.ssl_abstract.SSLSegAgent` is the abstract class used for semi-supervised learning. The built-in SSL methods are child classes of SSLSegAgent. The available SSL methods implemented in PyMIC are listed in `pymic.net_run.ssl.ssl_main.SSLMethodDict`, and they are:

- **EntropyMinimization**: (NeurIPS 2005) Using entropy minimization to regularize unannotated samples.
- **MeanTeacher**: (NeurIPS 2017) Use self-ensembling mean teacher to supervise the student model on unannotated samples.
- **UAMT**: (MICCAI 2019) Uncertainty aware mean teacher.
- **CCT**: (CVPR 2020) Cross-consistency training.
- **CPS**: (CVPR 2021) Cross-pseudo supervision.
- **URPC**: (MIA 2022) Uncertainty rectified pyramid consistency.

### 2.3.4 Customized SSL Methods

PyMIC also supports customizing SSL methods by inheriting the SSLSegAgent class. You may only need to rewrite the `training()` method and reuse most part of the existing pipeline, such as data loading, validation and inference methods. For example:

```python
from pymic.net_run.ssl.ssl_abstract import SSLSegAgent
class MySSLMethod(SSLSegAgent):
    def __init__(self, config, stage = 'train'):
        super(MySSLMethod, self).__init__(config, stage)
        ...
```

(continues on next page)
You may need to check the source code of built-in SSL methods to be more familiar with how to implement your own SSL method.

2.4 Weakly-Supervised Learning

2.4.1 pymic_wsl

pymic_wsl is the command for using built-in weakly-supervised methods for training. Similarly to pymic_run, it should be followed by two parameters, specifying the stage and configuration file, respectively. The training and testing commands are:

```
pymic_wsl train myconfig_wsl.cfg
pymic_wsl test myconfig_wsl.cfg
```

Tip: If the WSL method only involves one network, either pymic_wsl or pymic_run can be used for inference. Their difference only exists in the training stage.

Note: Currently, the weakly supervised methods supported by PyMIC are only for learning from partial annotations, such scribble-based annotation. Learning from image-level or point annotations may involve several training stages and will be considered in the future.

2.4.2 WSL Configurations

In the configuration file for pymic_wsl, in addition to those used in fully supervised learning, there are some items specificated for weakly-supervised learning.

First, in the train_transform list, a special transform named PartialLabelToProbability should be used to transform partial labels into a one-hot probability map and a weighting map of pixels (i.e., the weight of a pixel is 1 if labeled and 0 otherwise). The partial cross entropy loss on labeled pixels is actually implemented by a weighted cross entropy loss. The loss setting is loss_type = CrossEntropyLoss.

Second, there is a weakly_supervised_learning section that is specifically designed for WSL methods. In that section, users need to specify the wsl_method and configurations related to the WSL method. For example, the corresponding configuration for GatedCRF is:

```
[dataset]
...  
root_dir = ../../../PyMIC_data/ACDC/preprocess
train_csv = config/data/image_train.csv
valid_csv = config/data/image_valid.csv
test_csv = config/data/image_test.csv
```

(continues on next page)
train_batch_size = 4

# data transforms
train_transform = [Pad, RandomCrop, RandomFlip, NormalizeWithMeanStd,
                  PartialLabelToProbability]
valid_transform = [NormalizeWithMeanStd, Pad, LabelToProbability]
test_transform = [NormalizeWithMeanStd, Pad]
...

[network]
...

[training]
...
loss_type = CrossEntropyLoss
...

[weakly_supervised_learning]
wsli_method = GatedCRF
regularize_w = 0.1
rampup_start = 2000
rampup_end = 15000
GatedCRFLoss_W0 = 1.0
GatedCRFLoss_XY0 = 5
GatedCRFLoss_rgb = 0.1
GatedCRFLoss_W1 = 1.0
GatedCRFLoss_XY1 = 3
GatedCRFLoss_Radius = 5

[testing]
...

Note: The configuration items vary with different WSL methods. Please refer to the API of each built-in WSL method for details of the corresponding configuration.

2.4.3 Built-in WSL Methods

pymic.net_run_wsl.wsl_abstract.WSLSegAgent is the abstract class used for weakly-supervised learning. The built-in WSL methods are child classes of WSLSegAgent. The available WSL methods implemented in PyMIC are listed in pymic.net_run_wsl.wsl_main.WSLMethodDict, and they are:

- TotalVariation: (arXiv 2022) Use Total Variation to regularize unannotated pixels.
- USTM: (PR 2022) Adapt USTM with transform-consistency regularization.
- DMPLS: (MICCAI 2022) Dynamically mixed pseudo label supervision
2.4.4 Customized WSL Methods

PyMIC also supports customizing WSL methods by inheriting the `WSLSegAgent` class. You may only need to rewrite the `training()` method and reuse most part of the existing pipeline, such as data loading, validation and inference methods. For example:

```python
from pymic.net_run_wsl.wsl_abstract import WSLSegAgent
class MyWSLMethod(WSLSegAgent):
    def __init__(self, config, stage = 'train'):
        super(MyWSLMethod, self).__init__(config, stage)
        ...

    def training(self):
        ...

agent = MyWSLMethod(config, stage)
agent.run()
```

You may need to check the source code of built-in WSL methods to be more familiar with how to implement your own WSL method.

2.5 Noisy Label Learning

2.5.1 pymic_nll

`pymic_nll` is the command for using built-in NLL methods for training. Similarly to `pymic_run`, it should be followed by two parameters, specifying the stage and configuration file, respectively. The training and testing commands are:

```
pymic_nll train myconfig_nll.cfg
pymic_nll test myconfig_nll.cfg
```

*Tip:* If the NLL method only involves one network, either `pymic_nll` or `pymic_run` can be used for inference. Their difference only exists in the training stage.

*Note:* Some NLL methods only use noise-robust loss functions without complex training process, and just combining the standard `SegmentationAgent` with such loss function works for training. `pymic_run` instead of `pymic_nll` should be used for these methods.
2.5.2 NLL Configurations

In the configuration file for `pymic_nll`, in addition to those used in standard fully supervised learning, there is a `noisy_label_learning` section that is specifically designed for NLL methods. In that section, users need to specify the `nll_method` and configurations related to the NLL method. For example, the corresponding configuration for CoTeaching is:

```plaintext
[dataset]
...

[network]
...

[training]
...

[noisy_label_learning]
nll_method = CoTeaching
coteaching_select_ratio = 0.8
rampup_start = 1000
rampup_end = 8000

[testing]
...
```

**Note:** The configuration items vary with different NLL methods. Please refer to the API of each built-in NLL method for details of the corresponding configuration.

2.5.3 Built-in NLL Methods

Some NLL methods only use noise-robust loss functions. They are used with `pymic_run` for training. Just set `loss_type` to one of them in the configuration file, similarly to the fully supervised learning.

- **GCELoss:** (NeurIPS 2018) Generalized cross entropy loss.
- **MAELoss:** (AAAI 2017) Mean Absolute Error loss.
- **NR DiceLoss:** (TMI 2020) Noise-robust Dice loss.

The other NLL methods are implemented in child classes of `pymic.net_run_nll.nll_abstract.NLLSegAgent`, and they are:

- **CLSLR:** (MICCAI 2020) Confident learning with spatial label smoothing regularization.
- **CoTeaching:** (NeurIPS 2018) Co-teaching between two networks for learning from noisy labels.
- **TriNet:** (MICCAI 2020) Tri-network combined with sample selection.
- **DAST:** (JBHI 2022) Divergence-aware selective training.
2.5.4 Customized NLL Methods

PyMIC also supports customizing NLL methods by inheriting the `NLLSegAgent` class. You may only need to rewrite the `training()` method and reuse most part of the existing pipeline, such as data loading, validation and inference methods. For example:

```python
from pymic.net_run_nll.nll_abstract import NLLSegAgent

class MyNLLMethod(NLLSegAgent):
    def __init__(self, config, stage = 'train'):
        super(MyNLLMethod, self).__init__(config, stage)
        ...

    def training(self):
        ...

agent = MyNLLMethod(config, stage)
agent.run()
```

You may need to check the source code of built-in NLL methods to be more familiar with how to implement your own NLL method.

In addition, if you want to design a new noise-robust loss function, just follow *Fully Supervised Learning* to implement and use the customized loss.
3.1 pymic.io package

3.1.1 Submodules

3.1.2 pymic.io.h5_dataset module

3.1.3 pymic.io.image_read_write module

pymic.io.image_read_write.load_image_as_nd_array(image_name)
Load an image and return a 4D array with shape [C, D, H, W], or 3D array with shape [C, H, W].

Parameters
- filename (str) The input file name

Returns
A dictionary storing data array, origin, spacing and direction.

pymic.io.image_read_write.load_nifty_volume_as_4d_array(filename)
Read a nifty image and return a dictionary storing data array, origin, spacing and direction.

output['data_array'] 4D array with shape [C, D, H, W];
output['spacing'] A list of spacing in z, y, x axis;
output['direction'] A 3x3 matrix for direction.

Parameters
- filename (str) The input file name

Returns
A dictionary storing data array, origin, spacing and direction.

pymic.io.image_read_write.load_rgb_image_as_3d_array(filename)
Read an RGB image and return a dictionary storing data array, origin, spacing and direction.

output['data_array'] 3D array with shape [D, H, W];
output['spacing'] a list of spacing in z, y, x axis;
output['direction'] a 3x3 matrix for direction.

Parameters
- filename (str) The input file name

Returns
A dictionary storing data array, origin, spacing and direction.
Rotate the axis of a 3D volume to LPS

**Parameters**

- `filename_or_image_dict` – (str) Filename of the nifty file (str) or image dictionary returned by load_nifty_volume_as_4d_array. If supplied with the former, the flipped image data will be saved to override the original file. If supplied with the later, only flipped image data will be returned.
- `origin` – (list/tuple) The origin of the image.
- `direction` – (list or tuple) The direction of the image.

**Returns**

A dictionary for image data and meta info, with `data_array`, `origin`, `direction` and `spacing`.

Save a numpy array as nifty image

**Parameters**

- `data` – (numpy.ndarray) A numpy array with shape [Depth, Height, Width].
- `image_name` – (str) The output file name.
- `reference_name` – (str) File name of the reference image of which meta information is used.

Save a numpy array as rgb image.

**Parameters**

- `image_name` – (str) The output file name.

Save a 3D or 2D numpy array as medical image or RGB image.

**Parameters**

- `reference_name` – (str) File name of the reference image of which meta information is used.

### 3.1.4 pymic.io.nifty_dataset module

**class** `pymic.io.nifty_dataset.ClassificationDataset(root_dir, csv_file, modal_num=1, class_num=2, with_label=False, transform=None)`

**Bases:** `NiftyDataset`


**Parameters**

- `root_dir` – (str) Directory with all the images.
• csv_file – (str) Path to the csv file with image names.
• modal_num – (int) Number of modalities.
• class_num – (int) Class number of the classification task.
• with_label – (bool) Load the data with segmentation ground truth or not.
• transform – (list) List of transforms to be applied on a sample. The built-in transforms can be listed in `pymic.transform.trans_dict`.

```python
class pymic.io.nifty_dataset.NiftyDataset(root_dir, csv_file, modal_num=1, with_label=False, transform=None)
```

Bases: Dataset


Parameters

• root_dir – (str) Directory with all the images.
• csv_file – (str) Path to the csv file with image names.
• modal_num – (int) Number of modalities.
• with_label – (bool) Load the data with segmentation ground truth or not.
• transform – (list) List of transforms to be applied on a sample. The built-in transforms can be listed in `pymic.transform.trans_dict`.

3.1.5 Module contents

3.2 pymic.layer package

3.2.1 Submodules

3.2.2 pymic.layer.activation module

```python
pymic.layer.activation.get_acti_func(acti_func, params)
```

3.2.3 pymic.layer.convolution module

```python
class pymic.layer.convolution.ConvolutionLayer(in_channels, out_channels, kernel_size, dim=3, stride=1, padding=0, dilation=1, conv_group=1, bias=True, norm_type='batch_norm', norm_group=1, acti_func=None)
```

Bases: Module

A compose layer with the following components: convolution -> (batch_norm / layer_norm / group_norm / instance_norm) -> (activation) -> (dropout) Batch norm and activation are optional.

Parameters

• in_channels – (int) The input channel number.
• out_channels – (int) The output channel number.
• **kernel_size** – The size of convolution kernel. It can be either a single int or a tuple of two or three ints.

• **dim** – (int) The dimension of convolution (2 or 3).

• **stride** – (int) The stride of convolution.

• **padding** – (int) Padding size.

• **dilation** – (int) Dilation rate.

• **conv_group** – (int) The group number of convolution.

• **bias** – (bool) Add bias or not for convolution.

• **norm_type** – (str or None) Normalization type, can be `batch_norm`, `group_norm`.

• **norm_group** – (int) The number of group for group normalization.

• **acti_func** – (str or None) Activation function.

```python
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
training: bool
class pymic.layer.convolution.DepthSeperableConvolutionLayer(in_channels, out_channels, kernel_size, dim=3, stride=1, padding=0, dilation=1, conv_group=1, bias=True, norm_type='batch_norm', norm_group=1, acti_func=None)
```

Bases: `Module`

Depth separable convolution with the following components: 1x1 conv -> group conv -> (batch_norm / layer_norm / group_norm / instance_norm) -> (activation) -> (dropout) Batch norm and activation are optional.

**Parameters**

• **in_channels** – (int) The input channel number.

• **out_channels** – (int) The output channel number.

• **kernel_size** – The size of convolution kernel. It can be either a single int or a tuple of two or three ints.

• **dim** – (int) The dimension of convolution (2 or 3).

• **stride** – (int) The stride of convolution.

• **padding** – (int) Padding size.

• **dilation** – (int) Dilation rate.

• **conv_group** – (int) The group number of convolution.

• **bias** – (bool) Add bias or not for convolution.
- **norm_type** – (str or None) Normalization type, can be `batch_norm`, ‘group_norm’.
- **norm_group** – (int) The number of group for group normalization.
- **acti_func** – (str or None) Activation function.

```python
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
training: bool
```

### 3.2.4 `pymic.layer.deconvolution` module

```python
class pymic.layer.deconvolution.DeconvolutionLayer(in_channels, out_channels, kernel_size, dim=3, 
    stride=1, padding=0, output_padding=0, 
    dilation=1, groups=1, bias=True, 
    batch_norm=True, acti_func=None)
```

Bases: `Module`

A compose layer with the following components: deconvolution -> (batch_norm / layer_norm / group_norm / instance_norm) -> (activation) -> (dropout) Batch norm and activation are optional.

**Parameters**

- **in_channels** – (int) The input channel number.
- **out_channels** – (int) The output channel number.
- **kernel_size** – The size of convolution kernel. It can be either a single int or a tupe of two or three ints.
- **dim** – (int) The dimension of convolution (2 or 3).
- **stride** – (int) The stride of convolution.
- **padding** – (int) Padding size.
- **dilation** – (int) Dilation rate.
- **groups** – (int) The group number of convolution.
- **bias** – (bool) Add bias or not for convolution.
- **batch_norm** – (bool) Use batch norm or not.
- **acti_func** – (str or None) Activation function.

```python
forward(x)
```

Defines the computation performed at every call.

Should be overridden by all subclasses.
training:  bool

class pymic.layer.deconvolution.DepthSeparableDeconvolutionLayer(in_channels, out_channels, kernel_size, dim=3, stride=1, padding=0, output_padding=0, dilation=1, groups=1, bias=True, batch_norm=True, acti_func=None)

Bases: Module

Depth separable deconvolution with the following components: 1x1 conv -> deconv -> (batch_norm / layer_norm / group_norm / instance_norm) -> (activation) -> (dropout) Batch norm and activation are optional.

Parameters

- in_channels – (int) The input channel number.
- out_channels – (int) The output channel number.
- kernel_size – The size of convolution kernel. It can be either a single int or a tuple of two or three ints.
- dim – (int) The dimension of convolution (2 or 3).
- stride – (int) The stride of convolution.
- padding – (int) Padding size for input.
- output_padding – (int) Padding size for output.
- dilation – (int) Dilation rate.
- groups – (int) The group number of convolution.
- bias – (bool) Add bias or not for convolution.
- batch_norm – (bool) Use batch norm or not.
- acti_func – (str or None) Activation function.

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.

training:  bool
3.2.5 `pymic.layer.space2channel` module

```python
class pymic.layer.space2channel.ChannelToSpace3D
    Bases: Module
    Channel to space transform for 3D input.
    forward(x)
        Defines the computation performed at every call.
        Should be overridden by all subclasses.

    training: bool
```

```python
class pymic.layer.space2channel.SpaceToChannel3D
    Bases: Module
    Space to channel transform for 3D input.
    forward(x)
        Defines the computation performed at every call.
        Should be overridden by all subclasses.

    training: bool
```

3.2.6 Module contents

3.3 `pymic.loss` package

3.3.1 Subpackages

`pymic.loss.cls` package

Submodules

`pymic.loss.cls.basic` module

```python
class pymic.loss.cls.basic.AbstractClassificationLoss(params=None)
    Bases: Module
    Abstract Classification Loss.
```
forward(loss_input_dict)

The arguments should be written in the loss_input_dict dictionary, and it has the following fields.

Parameters

• prediction – A prediction with shape of [N, C] where C is the class number.
• ground_truth – The corresponding ground truth, with shape of [N, 1].

Note that prediction is the digit output of a network, before using softmax.

training: bool

class pymic.loss.cls.basic.CrossEntropyLoss(params=None)

Bases: AbstractClassificationLoss

Standard Softmax-based CE loss.

forward(loss_input_dict)

The arguments should be written in the loss_input_dict dictionary, and it has the following fields.

Parameters

• prediction – A prediction with shape of [N, C] where C is the class number.
• ground_truth – The corresponding ground truth, with shape of [N, 1].

Note that prediction is the digit output of a network, before using softmax.

training: bool

class pymic.loss.cls.basic.L1Loss(params=None)

Bases: AbstractClassificationLoss

L1 (MAE) loss for classification.

forward(loss_input_dict)

The arguments should be written in the loss_input_dict dictionary, and it has the following fields.

Parameters

• prediction – A prediction with shape of [N, C] where C is the class number.
• ground_truth – The corresponding ground truth, with shape of [N, 1].

Note that prediction is the digit output of a network, before using softmax.

training: bool

class pymic.loss.cls.basic.MSELoss(params=None)

Bases: AbstractClassificationLoss

Mean Square Error loss for classification.

forward(loss_input_dict)

The arguments should be written in the loss_input_dict dictionary, and it has the following fields.

Parameters

• prediction – A prediction with shape of [N, C] where C is the class number.
• ground_truth – The corresponding ground truth, with shape of [N, 1].

Note that prediction is the digit output of a network, before using softmax.
**training:**  bool

class pymic.loss.cls.basic.NLLLoss(params=None)
    Bases: AbstractClassificationLoss
    The negative log likelihood loss for classification.

    forward(loss_input_dict)
      The arguments should be written in the `loss_input_dict` dictionary, and it has the following fields.

      Parameters
      • **prediction** – A prediction with shape of [N, C] where C is the class number.
      • **ground_truth** – The corresponding ground truth, with shape of [N, 1].

      Note that `prediction` is the digit output of a network, before using softmax.

**training:**  bool

class pymic.loss.cls.basic.SigmoidCELoss(params=None)
    Bases: AbstractClassificationLoss
    Sigmoid-based CE loss.

    forward(loss_input_dict)
      The arguments should be written in the `loss_input_dict` dictionary, and it has the following fields.

      Parameters
      • **prediction** – A prediction with shape of [N, C] where C is the class number.
      • **ground_truth** – The corresponding ground truth, with shape of [N, 1].

      Note that `prediction` is the digit output of a network, before using softmax.

pymic.loss.cls.util module

pymic.loss.cls.util.get_soft_label(input_tensor, num_class, data_type='float')
    Convert a label tensor to one-hot soft label.

    Parameters
    • **input_tensor** – Tensor with shape of [B, 1].
    • **output_tensor** – Tensor with shape of [B, num_class].
    • **num_class** – (int) Class number.
    • **data_type** – (str) `float` or `double`. 
Module contents

pymic.loss.seg package

Submodules

pymic.loss.seg.abstract module

class pymic.loss.seg.abstract.AbstractSegLoss(params=None)
    Bases: Module

    Abstract class for loss function of segmentation tasks. The parameters should be written in the params dictionary, and it has the following fields:

    Parameters
    - loss_softmax – (optional, bool) Apply softmax to the prediction of network or not. Default is True.

    forward(loss_input_dict)

    Forward pass for calculating the loss. The arguments should be written in the loss_input_dict dictionary, and it has the following fields:

    Parameters
    - pixel_weight – (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.

    Returns
    - Loss function value.

    training: bool

pymic.loss.seg.ce module

class pymic.loss.seg.ce.CrossEntropyLoss(params=None)
    Bases: AbstractSegLoss

    Cross entropy loss for segmentation tasks.

    The parameters should be written in the params dictionary, and it has the following fields:

    Parameters
    - loss_softmax – (optional, bool) Apply softmax to the prediction of network or not. Default is True.

    forward(loss_input_dict)

    Forward pass for calculating the loss. The arguments should be written in the loss_input_dict dictionary, and it has the following fields:

    Parameters


• **pixel_weight** – (optional) Pixel-wise weight map, with the shape of $[N, 1, D, H, W]$ or $[N, 1, H, W]$. Default is None.

**Returns**

Loss function value.

```python
training: bool
class pymic.loss.seg.ce.GeneralizedCELoss(params)
    Bases: AbstractSegLoss
    Generalized cross entropy loss to deal with noisy labels.

    The parameters should be written in the `params` dictionary, and it has the following fields:

    **Parameters**

    • **loss_softmax** – (bool) Apply softmax to the prediction of network or not.

    • **loss_gce_q** – (float): hyper-parameter in the range of $(0, 1)$.

    • **loss_with_pixel_weight** – (optional, bool): Use pixel weighting or not.

    • **loss_class_weight** – (optional, list or none): If not none, a list of weight for each class.

    **forward(loss_input_dict)**
    Forward pass for calculating the loss. The arguments should be written in the `loss_input_dict` dictionary, and it has the following fields:

    **Parameters**


    • **pixel_weight** – (optional) Pixel-wise weight map, with the shape of $[N, 1, D, H, W]$ or $[N, 1, H, W]$. Default is None.

    **Returns**

    Loss function value.

    training: bool
```
**pymic.loss.seg.combined module**

class pymic.loss.seg.combined.CombinedLoss(params, loss_dict)

Bases: AbstractSegLoss

A combination of a list of loss functions. Parameters should be saved in the `params` dictionary.

Parameters

- `loss_softmax` – (optional, bool) Apply softmax to the prediction of network or not. Default is True.
- `loss_type` – (list) A list of loss function name.
- `loss_weight` – (list) A list of weights for each loss function.
- `loss_dict` – (dictionary) A dictionary of available loss functions.

**forward(loss_input_dict)**

Forward pass for calculating the loss. The arguments should be written in the `loss_input_dict` dictionary, and it has the following fields:

Parameters

- `pixel_weight` – (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.

Returns

Loss function value.

**training: bool**

**pymic.loss.seg.deep_sup module**

class pymic.loss.seg.deep_sup.DeepSuperviseLoss(params)

Bases: AbstractSegLoss

Combine deep supervision with a basic loss function. Arguments should be provided in the `params` dictionary, and it has the following fields:

Parameters

- `loss_softmax` – (optional, bool) Apply softmax to the prediction of network or not. Default is True.
- `deep_supervise_weight` – (list) A list of weight for each deep supervision scale.
- `base_loss` – (nn.Module) The basic function used for each scale.

**forward(loss_input_dict)**

Forward pass for calculating the loss. The arguments should be written in the `loss_input_dict` dictionary, and it has the following fields:

Parameters

- `pixel_weight` – (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.
• **prediction** – (tensor) Prediction of a network, with the shape of [N, C, D, H, W] or [N, C, H, W].

• **ground_truth** – (tensor) Ground truth, with the shape of [N, C, D, H, W] or [N, C, H, W].

• **pixel_weight** – (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.

**Returns**
Loss function value.

**training**: bool

### pymic.loss.seg.dice module

**class** `pymic.loss.seg.dice.DiceLoss(params=None)`

**Bases:** `AbstractSegLoss`

Dice loss for segmentation tasks. The parameters should be written in the `params` dictionary, and it has the following fields:

**Parameters**
- `loss_softmax` – (bool) Apply softmax to the prediction of network or not.

**forward**(loss_input_dict)
Forward pass for calculating the loss. The arguments should be written in the `loss_input_dict` dictionary, and it has the following fields:

**Parameters**
- **prediction** – (tensor) Prediction of a network, with the shape of [N, C, D, H, W] or [N, C, H, W].

- **ground_truth** – (tensor) Ground truth, with the shape of [N, C, D, H, W] or [N, C, H, W].

- **pixel_weight** – (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.

**Returns**
Loss function value.

**training**: bool

**class** `pymic.loss.seg.dice.FocalDiceLoss(params=None)`

**Bases:** `AbstractSegLoss`

Focal Dice according to the following paper:
- Pei Wang and Albert C. S. Chung, Focal Dice Loss and Image Dilation for Brain Tumor Segmentation, 2018.

The parameters should be written in the `params` dictionary, and it has the following fields:

**Parameters**
- `loss_softmax` – (bool) Apply softmax to the prediction of network or not.

- `FocalDiceLoss_beta` – (float) The hyper-parameter to set (>=1.0).
forward(loss_input_dict)

Forward pass for calculating the loss. The arguments should be written in the loss_input_dict dictionary, and it has the following fields:

Parameters

- **prediction** – (tensor) Prediction of a network, with the shape of [N, C, D, H, W] or [N, C, H, W].
- **ground_truth** – (tensor) Ground truth, with the shape of [N, C, D, H, W] or [N, C, H, W].
- **pixel_weight** – (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.

Returns
Loss function value.

training: bool

class pymic.loss.seg.dice.NoiseRobustDiceLoss(params)

Bases: AbstractSegLoss

Noise-robust Dice loss according to the following paper.


The parameters should be written in the params dictionary, and it has the following fields:

Parameters

- **loss_softmax** – (bool) Apply softmax to the prediction of network or not.
- **NoiseRobustDiceLoss_gamma** – (float) The hyper-parameter gamma to set (1, 2).

forward(loss_input_dict)

Forward pass for calculating the loss. The arguments should be written in the loss_input_dict dictionary, and it has the following fields:

Parameters

- **prediction** – (tensor) Prediction of a network, with the shape of [N, C, D, H, W] or [N, C, H, W].
- **ground_truth** – (tensor) Ground truth, with the shape of [N, C, D, H, W] or [N, C, H, W].
- **pixel_weight** – (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.

Returns
Loss function value.

training: bool
**pymic.loss.seg.exp_log module**

**class** `pymic.loss.seg.exp_log.ExpLogLoss(params)`

**Bases:** `AbstractSegLoss`

The exponential logarithmic loss in this paper:


The arguments should be written in the `params` dictionary, and it has the following fields:

**Parameters**

- `loss_softmax` – (bool) Apply softmax to the prediction of network or not.
- `ExpLogLoss_w_dice` – (float) Weight of ExpLog Dice loss in the range of [0, 1].
- `ExpLogLoss_gamma` – (float) Hyper-parameter gamma.

**forward(loss_input_dict)**

Forward pass for calculating the loss. The arguments should be written in the `loss_input_dict` dictionary, and it has the following fields:

**Parameters**

- `pixel_weight` – (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.

**Returns**

Loss function value.

**training:** `bool`

**pymic.loss.seg.gatedcrf module**

The code is adapted from the original implementation at Github.

**class** `pymic.loss.seg.gatedcrf.GatedCRFLoss`  
**Bases:** `Module`

Gated CRF Loss for Weakly Supervised Semantic Image Segmentation. This loss function promotes consistent label assignment guided by input features, such as RGBXY.


**forward(y_hat_softmax, kernels_desc, kernels_radius, sample, height_input, width_input, mask_src=None, mask_dst=None, compatibility=None, custom_modality_downsamplers=None, out_kernels_vis=False)**

Performs the forward pass of the loss.

**Parameters**

- `y_hat_softmax` – A tensor of predicted per-pixel class probabilities of size NxCxHxW
• **kernels_desc** – A list of dictionaries, each describing one Gaussian kernel composition from modalities. The final kernel is a weighted sum of individual kernels. Following example is a composition of RGBXY and XY kernels: kernels_desc: [{'weight': 0.9,'xy': 6,'rgb': 0.1},{'weight': 0.1,'xy': 6}]

• **kernels_radius** – Defines size of bounding box region around each pixel in which the kernel is constructed.

• **sample** – A dictionary with modalities (except ‘xy’) used in kernels_desc parameter. Each of the provided modalities is allowed to be larger than the shape of y_hat_softmax, in such case downsampling will be invoked. Default downsampling method is area resize; this can be overridden by setting custom_modality_downsamplers parameter.

• **height_input (width_input,)** – Dimensions of the full scale resolution of modalities

• **mask_src** – (optional) Source mask.

• **mask_dst** – (optional) Destination mask.

• **compatibility** – (optional) Classes compatibility matrix, defaults to Potts model.

• **custom_modality_downsamplers** – A dictionary of modality downsampling functions.

• **out_kernels_vis** – Whether to return a tensor with kernels visualized with some step.

Returns
Loss function value.

```
training: bool
```

**pymic.loss.seg.mse module**

```python

class pymic.loss.seg.mse.MAEloss(params=None)

    Bases: AbstractSegLoss

    Mean Absolute Loss for segmentation tasks. The arguments should be written in the params dictionary, and it has the following fields:

    Parameters
    loss_softmax – (bool) Apply softmax to the prediction of network or not.

    forward(loss_input_dict)

    Forward pass for calculating the loss. The arguments should be written in the loss_input_dict dictionary, and it has the following fields:

    Parameters
    • prediction – (tensor) Prediction of a network, with the shape of [N, C, D, H, W] or [N, C, H, W].
    • ground_truth – (tensor) Ground truth, with the shape of [N, C, D, H, W] or [N, C, H, W].
    • pixel_weight – (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.

    Returns
    Loss function value.

    training: bool
```

36 Chapter 3. API
class pymic.loss.seg.mse.MSELoss(params=None)
    Bases: AbstractSegLoss

    Mean Square Loss for segmentation tasks. The parameters should be written in the params dictionary, and it has the following fields:

    Parameters
    loss_softmax -- (bool) Apply softmax to the prediction of network or not.

    forward(loss_input_dict)
    Forward pass for calculating the loss. The arguments should be written in the loss_input_dict dictionary, and it has the following fields:

    Parameters
    • prediction -- (tensor) Prediction of a network, with the shape of [N, C, D, H, W] or [N, C, H, W].
    • ground_truth -- (tensor) Ground truth, with the shape of [N, C, D, H, W] or [N, C, H, W].
    • pixel_weight -- (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.

    Returns
    Loss function value.

    training: bool

pymic.loss.seg.mumford_shah module

class pymic.loss.seg.mumford_shah.MumfordShahLoss(params=None)
    Bases: Module

    Implementation of Mumford Shah Loss for weakly supervised learning.


    The original implementation is available at Github. Currently only 2D version is supported.

    The parameters should be written in the params dictionary, and it has the following fields:

    Parameters
    • loss_softmax -- (bool) Apply softmax to the prediction of network or not.
    • MumfordShahLoss_penalty -- (optional, str) l1 or l2. Default is l1.
    • MumfordShahLoss_lambda -- (optional, float) Hyper-parameter lambda, default is 1.0.

    forward(loss_input_dict)
    Forward pass for calculating the loss. The arguments should be written in the loss_input_dict dictionary, and it has the following fields:

    Parameters
    • prediction -- (tensor) Prediction of a network, with the shape of [N, C, D, H, W] or [N, C, H, W].
    • image -- (tensor) Image, with the shape of [N, C, D, H, W] or [N, C, H, W].
Returns
Loss function value.

get_gradient_loss(pred, penalty='l2')

get_levelset_loss(output, target)
Get the level set loss value.

Parameters
• output – (tensor) softmax output of a network.
• target – (tensor) the input image.

Returns
the level set loss.

training: bool

pymic.loss.seg.slsr module

class pymic.loss.seg.slsr.SLSRLoss(params=None)
Bases: AbstractSegLoss

Spatial Label Smoothing Regularization (SLSR) loss for learning from noisy annotations. This loss requires pixel weighting, please make sure that a pixel_weight field is provided for the csv file of the training images.

The pixel weight here is actually the confidence mask, i.e., if the value is one, it means the label of corresponding pixel is noisy and should be smoothed.


The arguments should be written in the params dictionary, and it has the following fields:

Parameters
• loss_softmax – (bool) Apply softmax to the prediction of network or not.
• slsrloss_epsilon – (optional, float) Hyper-parameter epsilon. Default is 0.25.

forward(loss_input_dict)
Forward pass for calculating the loss. The arguments should be written in the loss_input_dict dictionary, and it has the following fields:

Parameters
• prediction – (tensor) Prediction of a network, with the shape of [N, C, D, H, W] or [N, C, H, W].
• ground_truth – (tensor) Ground truth, with the shape of [N, C, D, H, W] or [N, C, H, W].
• pixel_weight – (optional) Pixel-wise weight map, with the shape of [N, 1, D, H, W] or [N, 1, H, W]. Default is None.

Returns
Loss function value.

training: bool
**pymic.loss.seg.ssl module**

**class pymic.loss.seg.ssl.EntropyLoss**(params=None)

Bases: Module

Entropy Minimization for segmentation tasks. The parameters should be written in the `params` dictionary, and it has the following fields:

**Parameters**

- **loss_softmax** – (bool) Apply softmax to the prediction of network or not.

**forward**(loss_input_dict)

Forward pass for calculating the loss. The arguments should be written in the `loss_input_dict` dictionary, and it has the following fields:

**Parameters**

- **prediction** – (tensor) Prediction of a network, with the shape of [N, C, D, H, W] or [N, C, H, W].

**Returns**

Loss function value.

**training**: bool

**class pymic.loss.seg.ssl.TotalVariationLoss**(params=None)

Bases: Module

Total Variation Loss for segmentation tasks. The parameters should be written in the `params` dictionary, and it has the following fields:

**Parameters**

- **loss_softmax** – (bool) Apply softmax to the prediction of network or not.

**forward**(loss_input_dict)

Forward pass for calculating the loss. The arguments should be written in the `loss_input_dict` dictionary, and it has the following fields:

**Parameters**

- **prediction** – (tensor) Prediction of a network, with the shape of [N, C, D, H, W] or [N, C, H, W].

**Returns**

Loss function value.

**training**: bool

**pymic.loss.seg.util module**

**pymic.loss.seg.util.get_classwise_dice**(predict, soft_y, pix_w=None)

Get dice scores for each class in predict (after softmax) and soft_y.

**Parameters**

- **predict** – (tensor) Prediction of a segmentation network after softmax.
- **soft_y** – (tensor) The one-hot segmentation ground truth.
- **pix_w** – (optional, tensor) The pixel weight map. Default is None.
Returns
Dice score for each class.

```
pymic.loss.seg.util.get_soft_label(input_tensor, num_class, data_type='float')
```
Convert a label tensor to one-hot label for segmentation tasks.

Parameters
- **input_tensor** – (tensor) Tensor with shape [B, 1, D, H, W] or [B, 1, H, W].
- **num_class** – (int) The class number.
- **data_type** – (optional, str) Type of data, float (default) or double.

Returns
A tensor with shape [B, num_class, D, H, W] or [B, num_class, H, W]

```
pymic.loss.seg.util.reshape_prediction_and_ground_truth(predict, soft_y)
```
Reshape input variables two 2D.

Parameters

Returns
Two output tensors with shape [voxel_n, C] that correspond to the two inputs.

```
pymic.loss.seg.util.reshape_tensor_to_2D(x)
```

Module contents

### 3.3.2 Submodules

#### 3.3.3 pymic.loss.loss_dict_cls module

Built-in loss functions for classification.

- **CrossEntropyLoss** `pymic.loss.cls.basic.CrossEntropyLoss`
- **SigmoidCELoss** `pymic.loss.cls.basic.SigmoidCELoss`
- **L1Loss** `pymic.loss.cls.basic.L1Loss`
- **MSELoss** `pymic.loss.cls.basic.MSELoss`
- **NLLLoss** `pymic.loss.cls.basic.NLLLoss`
3.3.4 pymic.loss.loss_dict_seg module

Built-in loss functions for segmentation. The following are for fully supervised learning, or learning from noisy labels:

- CrossEntropyLoss  \texttt{pymic.loss.seg.ce.CrossEntropyLoss}
- GeneralizedCELoss \texttt{pymic.loss.seg.ce.GeneralizedCELoss}
- DiceLoss \texttt{pymic.loss.seg.dice.DiceLoss}
- FocalDiceLoss \texttt{pymic.loss.seg.dice.FocalDiceLoss}
- NoiseRobustDiceLoss \texttt{pymic.loss.seg.dice.NoiseRobustDiceLoss}
- ExpLogLoss \texttt{pymic.loss.seg.exp_log.ExpLogLoss}
- MAELoss \texttt{pymic.loss.seg.mse.MAEloss}
- MSELoss \texttt{pymic.loss.seg.mse.MSELoss}
- SLSRLoss \texttt{pymic.loss.seg.slsr.SLSRLoss}

The following are for semi-supervised or weakly supervised learning:

- EntropyLoss \texttt{pymic.loss.seg.ssl.EntropyLoss}
- GatedCRFLoss: \texttt{pymic.loss.seg.gatedcrf.GatedCRFLoss}
- MumfordShahLoss \texttt{pymic.loss.seg.mumford_shah.MumfordShahLoss}
- TotalVariationLoss \texttt{pymic.loss.seg.ssl.TotalVariationLoss}

3.3.5 Module contents

3.4 pymic.net package

3.4.1 Subpackages

\texttt{pymic.net.cls} package

Submodules

\texttt{pymic.net.cls.torch_pretrained_net} module

\texttt{class pymic.net.cls.torch_pretrained_net.BuiltInNet(params)}

\texttt{Bases: Module}

Built-in Network in Pytorch for classification. Parameters should be set in the \texttt{params} dictionary that contains the following fields:

Parameters

- \texttt{input_chns} – (int) Input channel number, default is 3.
- \texttt{pretrain} – (bool) Using pretrained model or not, default is True.
- \texttt{update_mode} – (str) The strategy for updating layers: “all” means updating all the layers, and “last” (by default) means updating the last layer, as well as the first layer when \texttt{input_chns} is not 3.
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
get_parameters_to_update()
```

```python
class pymic.net.cls.torch_pretrained_net.MobileNetV2(params)
```

Bases: `BuiltInNet`

MobileNetV2 for classification. Parameters should be set in the `params` dictionary that contains the following fields:

**Parameters**

- `input_chns` – (int) Input channel number, default is 3.
- `pretrain` – (bool) Using pretrained model or not, default is True.
- `update_mode` – (str) The strategy for updating layers: “all” means updating all the layers, and “last” (by default) means updating the last layer, as well as the first layer when `input_chns` is not 3.

```python
get_parameters_to_update()
```

```python
class pymic.net.cls.torch_pretrained_net.ResNet18(params)
```

Bases: `BuiltInNet`

ResNet18 for classification. Parameters should be set in the `params` dictionary that contains the following fields:

**Parameters**

- `input_chns` – (int) Input channel number, default is 3.
- `pretrain` – (bool) Using pretrained model or not, default is True.
- `update_mode` – (str) The strategy for updating layers: “all” means updating all the layers, and “last” (by default) means updating the last layer, as well as the first layer when `input_chns` is not 3.

```python
get_parameters_to_update()
```

```python
class pymic.net.cls.torch_pretrained_net.VGG16(params)
```

Bases: `BuiltInNet`

VGG16 for classification. Parameters should be set in the `params` dictionary that contains the following fields:

**Parameters**

- `input_chns` – (int) Input channel number, default is 3.
- `pretrain` – (bool) Using pretrained model or not, default is True.
• **update_mode** – (str) The strategy for updating layers: “all” means updating all the layers, and “last” (by default) means updating the last layer, as well as the first layer when `input_chns` is not 3.

    get_parameters_to_update()

    **training:** bool

### Module contents

**pymic.net.net2d package**

**Submodules**

**pymic.net.net2d.cople_net module**

**class pymic.net.net2d.cople_net.ASPPBlock**(in_channels, out_channels_list, kernel_size_list, dilation_list)

    Bases: Module
    
    ASPP block.

    **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.

    **training:** bool

**class pymic.net.net2d.cople_net.COPLENet**(params)

    Bases: Module

    Implementation of of COPLEnet for COVID-19 pneumonia lesion segmentation from CT images.


    Parameters are given in the `params` dictionary, and should include the following fields:

    **Parameters**

    • **in_chns** – (int) Input channel number.

    • **feature_chns** – (list) Feature channel for each resolution level. The length should be 5, such as [16, 32, 64, 128, 256].

    • **dropout** – (list) The dropout ratio for each resolution level. The length should be the same as that of `feature_chns`.

    • **class_num** – (int) The class number for segmentation task.

    • **bilinear** – (bool) Using bilinear for up-sampling or not. If False, deconvolution will be used for up-sampling.
PyMIC, Release 0.1

```python
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
training: bool
class pymic.net.net2d.cople_net.ConvBNActBlock(in_channels, out_channels, dropout_p)
Bases: Module
Two convolution layers with batch norm, leaky relu, dropout and SE block.

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
training: bool
class pymic.net.net2d.cople_net.ConvLayer(in_channels, out_channels, kernel_size=1)
Bases: Module
A combination of Conv2d, BatchNorm2d and LeakyReLU.

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
training: bool
class pymic.net.net2d.cople_net.DownBlock(in_channels, out_channels, dropout_p)
Bases: Module
Downsampling by a concatenation of max-pool and avg-pool, followed by ConvBNActBlock.

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
training: bool
class pymic.net.net2d.cople_net.SEBlock(in_channels, r)
Bases: Module
A Modified Squeeze-and-Excitation block for spatial attention.
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
training: bool
class pymic.net.net2d.cople_net.UpBlock(in_channels1, in_channels2, out_channels, bilinear=True, dropout_p=0.5)
Bases: Module
Upssampling followed by ConvBNActBlock.
forward(x1, x2)
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
training: bool

pymic.net.net2d.scse2d module

2D implementation of:
1. Channel Squeeze and Excitation
2. Spatial Squeeze and Excitation
3. Concurrent Spatial and Channel Squeeze & Excitation

Original file is on Github.
```
class pymic.net.net2d.scse2d.ChannelSELayer(num_channels, reduction_ratio=2)
    Bases: Module
    Re-implementation of Squeeze-and-Excitation (SE) block.
    

    Parameters
    - num_channels – Number of input channels
    - reduction_ratio – By how much should the num_channels should be reduced.

    forward(input_tensor)

    Parameters
    - input_tensor – X, shape = (batch_size, num_channels, H, W)

    Returns
    output tensor

    training: bool

class pymic.net.net2d.scse2d.ChannelSpatialSELayer(num_channels, reduction_ratio=2)
    Bases: Module
    Re-implementation of concurrent spatial and channel squeeze & excitation.
    
    - Reference: Roy et al., Concurrent Spatial and Channel Squeeze & Excitation in Fully Convolutional Net-
      works, MICCAI 2018.

    Parameters
    - num_channels – Number of input channels.
    - reduction_ratio – By how much should the num_channels should be reduced.

    forward(input_tensor)

    Parameters
    - input_tensor – X, shape = (batch_size, num_channels, H, W)

    Returns
    output_tensor

    training: bool

class pymic.net.net2d.scse2d.SELayer(value)
    Bases: Enum

    Enum restricting the type of SE Blockes available. So that type checking can be adding when adding these
    blockes to a neural network:
    
    ```python
    if self.se_block_type == se.SELayer.CSE.value:
        self.SELayer = se.ChannelSpatialSELayer(params['num_filters'])
    elif self.se_block_type == se.SELayer.SSE.value:
        self.SELayer = se.SpatialSELayer(params['num_filters'])
    elif self.se_block_type == se.SELayer.CSSE.value:
        self.SELayer = se.ChannelSpatialSELayer(params['num_filters'])
    ```
CSE = 'CSE'
CSSE = 'CSSE'
NONE = 'NONE'
SSE = 'SSE'

class pymic.net.net2d.scse2d.SpatialSELayer(num_channels)
    Bases: Module
    Re-implementation of SE block – squeezing spatially and exciting channel-wise.
    • Reference: Roy et al., Concurrent Spatial and Channel Squeeze & Excitation in Fully Convolutional Networks, MICCAI 2018.

    Parameters
    num_channels – Number of input channels.

    forward(input_tensor, weights=None)
    
    Parameters
    • weights – weights for few shot learning
    • input_tensor – X, shape = (batch_size, num_channels, H, W)

    Returns
    output_tensor

    training: bool

pymic.net.net2d.unet2d module

class pymic.net.net2d.unet2d.ConvBlock(in_channels, out_channels, dropout_p)
    Bases: Module
    Two convolution layers with batch norm and leaky relu. Dropout is used between the two convolution layers.

    Parameters
    • in_channels – (int) Input channel number.
    • out_channels – (int) Output channel number.
    • dropout_p – (int) Dropout probability.

    forward(x)
    
    Defines the computation performed at every call.
    Should be overridden by all subclasses.

    training: bool
class pymic.net.net2d.unet2d.Decoder(params)
    Bases: Module
    Decoder of 2D UNet.

    Parameters are given in the params dictionary, and should include the following fields:

    Parameters
    - `in_chns` – (int) Input channel number.
    - `feature_chns` – (list) Feature channel for each resolution level. The length should be 4 or 5, such as [16, 32, 64, 128, 256].
    - `dropout` – (list) The dropout ratio for each resolution level. The length should be the same as that of `feature_chns`.
    - `class_num` – (int) The class number for segmentation task.
    - `bilinear` – (bool) Using bilinear for up-sampling or not. If False, deconvolution will be used for up-sampling.

    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.

training: bool

class pymic.net.net2d.unet2d.DownBlock(in_channels, out_channels, dropout_p)
    Bases: Module
    Downsampling followed by ConvBlock

    Parameters
    - `in_channels` – (int) Input channel number.
    - `out_channels` – (int) Output channel number.
    - `dropout_p` – (int) Dropout probability.

    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.

training: bool
class pymic.net.net2d.unet2d.Encoder(params)

Bases: Module

Encoder of 2D UNet.

Parameters are given in the params dictionary, and should include the following fields:

**Parameters**

- **in_chns** – (int) Input channel number.
- **feature_chns** – (list) Feature channel for each resolution level. The length should be 4 or 5, such as [16, 32, 64, 128, 256].
- **dropout** – (list) The dropout ratio for each resolution level. The length should be the same as that of feature_chns.

**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.

training: bool

class pymic.net.net2d.unet2d.UNet2D(params)

Bases: Module

An implementation of 2D U-Net.


Note that there are some modifications from the original paper, such as the use of batch normalization, dropout, leaky relu and deep supervision.

Parameters are given in the params dictionary, and should include the following fields:

**Parameters**

- **in_chns** – (int) Input channel number.
- **feature_chns** – (list) Feature channel for each resolution level. The length should be 4 or 5, such as [16, 32, 64, 128, 256].
- **dropout** – (list) The dropout ratio for each resolution level. The length should be the same as that of feature_chns.
- **class_num** – (int) The class number for segmentation task.
- **bilinear** – (bool) Using bilinear for up-sampling or not. If False, deconvolution will be used for up-sampling.
- **deep_supervise** – (bool) Using deep supervision for training or not.

**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.

training: bool
class pymic.net.net2d.unet2d.UpBlock(in_channels1, in_channels2, out_channels, dropout_p, bilinear=True):
    bases: Module
    Upsampling followed by ConvBlock
    Parameters
    • in_channels1 – (int) Channel number of high-level features.
    • in_channels2 – (int) Channel number of low-level features.
    • out_channels – (int) Output channel number.
    • dropout_p – (int) Dropout probability.
    • bilinear – (bool) Use bilinear for up-sampling (by default). If False, deconvolution is used for up-sampling.
    forward(x1, x2)
    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 pymic.net.net2d.unet2d_attention module
class pymic.net.net2d.unet2d_attention.AttentionGateBlock(chns_l, chns_h):
    bases: Module
    forward(x_l, x_h)
    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 pymic.net.net2d.unet2d_attention.AttentionUNet2D(params):
    bases: UNet2D
training: bool

class pymic.net.net2d.unet2d_attention.UpBlockWithAttention(in_channels1, in_channels2, out_channels, dropout_p, bilinear=True)

Bases: Module
Upsampling followed by ConvBlock

forward(x1, x2)
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

pymic.net.net2d.unet2d_cct module

class pymic.net.net2d.unet2d_cct.AuxiliaryDecoder(params, aux_type)
Bases: Module
An Auxiliary Decoder. aux_type should be one of {DropOut, FeatureDrop, FeatureNoise and VAT}. Other parameters for the decoder are given in the params dictionary, see pymic.net.net2d.unet2d.Decoder for details. In addition, the following fields are needed for pertubation:

Parameters

- Uniform_range – (float) The range of noise. Only needed when aux_type = 'FeatureNoise'.
- VAT_it – (float) The iteration number of VAT. Only needed when aux_type = 'VAT'.
- VAT_xi – (float) The hyper-parameter xi of VAT. Only needed when aux_type = 'VAT'.
- VAT_eps – (float) The hyper-parameter eps of VAT. Only needed when aux_type = 'VAT'.

feature_based_noise(x)
feature_drop(x)

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.

training: bool
class pymic.net.net2d.unet2d_cct.UNet2D_CCT(params)

Bases: Module

An modification the U-Net with auxiliary decoders according to the CCT paper.


Code adapted from Github.

Parameter for the network backbone are given in the params dictionary, see `pymic.net.net2d.unet2d.UNet2D` for details. In addition, the following fields are needed for pertubation in the auxiliary decoders:

Parameters

- CCT_aux_decoders – (list) A list of auxiliary decoder types. Supported values are `{DropOut, FeatureDrop, FeatureNoise and VAT}`.

The parameters for different types of auxiliary decoders should also be given in the params dictionary, see `pymic.net.net2d.unet2d_cct.AuxiliaryDecoder` for details.

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.

training: bool

pymic.net.net2d.unet2d_dual_branch module

class pymic.net.net2d.unet2d_dual_branch.UNet2D_DualBranch(params)

Bases: Module

A dual branch network using UNet2D as backbone.


The parameters for the backbone should be given in the params dictionary. See `pymic.net.net2d.unet2d.UNet2D` for details. In addition, the following field should be included:

Parameters

- output_mode – (str) How to obtain the result during the inference. average: taking average of the two branches. first: taking the result in the first branch. second: taking the result in the second branch.

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
training: bool
```

**pymic.net2d.unet2d_nest module**

```python
class pymic.net2d.unet2d_nest.NestedUNet2D(params)
    Bases: Module
    An implementation of the Nested U-Net.
```

Note that there are some modifications from the original paper, such as the use of dropout and leaky relu here.

Parameters are given in the `params` dictionary, and should include the following fields:

**Parameters**

- **in_chns** – (int) Input channel number.
- **feature_chns** – (list) Feature channel for each resolution level. The length should be 4 or 5, such as [16, 32, 64, 128, 256].
- **dropout** – (list) The dropout ratio for each resolution level. The length should be the same as that of `feature_chns`.
- **class_num** – (int) The class number for segmentation task.

```python
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
training: bool
```

**pymic.net2d.unet2d_scse module**

```python
class pymic.net2d.unet2d_scse.ConvScSEBlock(in_channels, out_channels, dropout_p)
    Bases: Module
    Two convolutional blocks followed by `ChannelSpatialSELayer`. Each block consists of Conv2d + BatchNorm2d + LeakyReLU. A dropout layer is used between the wo blocks.

Parameters

- **in_channels** – (int) Input channel number.
- **out_channels** – (int) Output channel number.
```

3.4. `pymic.net` package
• **dropout_p** – (int) Dropout probability.

**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.

**training:**  `bool`

**class** `pymic.net.net2d.unet2d_scse.DownBlock`(*in_channels, out_channels, dropout_p*)

**Bases:** `Module`

Downsampling followed by `ConvScSEBlock`.

**Parameters**

- **in_channels** – (int) Input channel number.
- **out_channels** – (int) Output channel number.
- **dropout_p** – (int) Dropout probability.

**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.

**training:**  `bool`

**class** `pymic.net.net2d.unet2d_scse.UNet2D_ScSE`(*params*)

**Bases:** `Module`

Combining 2D U-Net with SCSE module.


Parameters are given in the `params` dictionary, and should include the following fields:

**Parameters**

- **in_chns** – (int) Input channel number.
- **feature_chns** – (list) Feature channel for each resolution level. The length should be 5, such as `[16, 32, 64, 128, 256]`.
- **dropout** – (list) The dropout ratio for each resolution level. The length should be the same as that of `feature_chns`.
- **class_num** – (int) The class number for segmentation task.
• **bilinear** – (bool) Using bilinear for up-sampling or not. If False, deconvolution will be used for up-sampling.

```python
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
training: bool
class pymic.net.net2d.unet2d_scse.UpBlock(in_channels1, in_channels2, out_channels, dropout_p, bilinear=True)
```

Up-sampling followed by `ConvSeSEBlock` in U-Net structure.

**Parameters**

- **in_channels1** – (int) Input channel number for low-resolution feature map.
- **in_channels2** – (int) Input channel number for high-resolution feature map.
- **out_channels** – (int) Output channel number.
- **dropout_p** – (int) Dropout probability.
- **bilinear** – (bool) Use bilinear for up-sampling or not.

```python
forward(x1, x2)
```

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
training: bool
```

**pymic.net.net2d.unet2d_urpc module**

```python
pymic.net.net2d.unet2d_urpc.FeatureDropout(x)
class pymic.net.net2d.unet2d_urpc.FeatureNoise(uniform_range=0.3)
```

**feature_based_noise(x)**
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.

training: bool

class pymic.net.net2d.unet2d_urpc.UNet2D_URPC($params$)
Bases: Module

An modification the U-Net to obtain multi-scale prediction according to the URPC paper.


Also see: https://github.com/HiLab-git/SSL4MIS/blob/master/code/networks/unet.py

Parameters are given in the $params$ dictionary, and should include the following fields:

Parameters

- **in_chns** – (int) Input channel number.
- **feature_chns** – (list) Feature channel for each resolution level. The length should be 5, such as [16, 32, 64, 128, 256].
- **dropout** – (list) The dropout ratio for each resolution level. The length should be the same as that of feature_chns.
- **class_num** – (int) The class number for segmentation task.
- **bilinear** – (bool) Using bilinear for up-sampling or not. If False, deconvolution will be used for up-sampling.

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.

training: bool
Module contents

pymic.net.net3d package

Submodules

pymic.net.net3d.scse3d module

3D implementation of:
1. Channel Squeeze and Excitation
2. Spatial Squeeze and Excitation
3. Concurrent Spatial and Channel Squeeze & Excitation

Original file is on Github.

class pymic.net.net3d.scse3d.ChannelSELayer3D(num_channels, reduction_ratio=2)
Bases: Module

3D implementation of Squeeze-and-Excitation (SE) block.

Parameters

- num_channels – Number of input channels
- reduction_ratio – By how much should the num_channels should be reduced

forward(input_tensor)

Parameters

input_tensor – X, shape = (batch_size, num_channels, D, H, W)

Returns
output tensor

training: bool

class pymic.net.net3d.scse3d.ChannelSpatialSELayer3D(num_channels, reduction_ratio=2)
Bases: Module

3D Re-implementation of concurrent spatial and channel squeeze & excitation.
- Reference: Roy et al., Concurrent Spatial and Channel Squeeze & Excitation in Fully Convolutional Networks, MICCAI 2018.

Parameters

- num_channels – Number of input channels
- reduction_ratio – By how much should the num_channels should be reduced

forward(input_tensor)

Parameters

input_tensor – X, shape = (batch_size, num_channels, D, H, W)
### Returns
- `output_tensor`

**training**: `bool`

class `pymic.net.net3d.scse3d.SELayer(value)`

Bases: `Enum`

Enum restricting the type of SE Blockes available. So that type checking can be adding when adding these blockes to a neural network:

```python
if self.se_block_type == se.SELayer.CSE.value:
    self.SELayer = se.ChannelSpatialSELayer(params['num_filters'])
elif self.se_block_type == se.SELayer.SSE.value:
    self.SELayer = se.SpatialSELayer(params['num_filters'])
elif self.se_block_type == se.SELayer.CSSE.value:
    self.SELayer = se.ChannelSpatialSELayer(params['num_filters'])
```

CSE = 'CSE'

CSSE = 'CSSE'

NONE = 'NONE'

SSE = 'SSE'

class `pymic.net.net3d.scse3d.SpatialSELayer3D(num_channels)`

Bases: `Module`

3D Re-implementation of SE block – squeezing spatially and exciting channel-wise described in:

- Reference: Roy et al., Concurrent Spatial and Channel Squeeze & Excitation in Fully Convolutional Networks, MICCAI 2018.

**Parameters**
- `num_channels` – Number of input channels

**forward**(input_tensor, weights=None)

**Parameters**
- weights – weights for few shot learning
- input_tensor – X, shape = (batch_size, num_channels, D, H, W)

**Returns**
- `output_tensor`

**training**: `bool`
**pymic.net.net3d.unet2d5 module**

```python
class pymic.net.net3d.unet2d5.ConvBlockND(in_channels, out_channels, dim=2, dropout_p=0.0)
    Bases: Module

    2D or 3D convolutional block

    Parameters
    ----------
    in_channels : int
        Input channel number.
    out_channels : int
        Output channel number.
    dim : int
        Should be 2 or 3, for 2D and 3D convolution, respectively.
    dropout_p : float
        Dropout probability.

    forward(x)
    Defines the computation performed at every call.
    Should be overridden by all subclasses.

    training: bool
```

**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
class pymic.net.net3d.unet2d5.DownBlock(in_channels, out_channels, dim=2, dropout_p=0.0, downsample=True)
    Bases: Module

    ConvBlockND block followed by downsampling.

    Parameters
    ----------
    in_channels : int
        Input channel number.
    out_channels : int
        Output channel number.
    dim : int
        Should be 2 or 3, for 2D and 3D convolution, respectively.
    dropout_p : float
        Dropout probability.
    downsample : bool
        Use downsample or not after convolution.

    forward(x)
    Defines the computation performed at every call.
    Should be overridden by all subclasses.

    training: bool
```

**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.
class pymic.net.net3d.unet2d5.UNet2D5(params)

Bases: Module

A 2.5D network combining 3D convolutions with 2D convolutions.


Note that the attention module in the original paper is not used here.

Parameters are given in the params dictionary, and should include the following fields:

Parameters

- `in_chns` – (int) Input channel number.
- `feature_chns` – (list) Feature channel for each resolution level. The length should be 5, such as [16, 32, 64, 128, 256].
- `dropout` – (list) The dropout ratio for each resolution level. The length should be the same as that of `feature_chns`.
- `conv_dims` – (list) The convolution dimension (2 or 3) for each resolution level. The length should be the same as that of `feature_chns`.
- `class_num` – (int) The class number for segmentation task.
- `bilinear` – (bool) Using bilinear for up-sampling or not. If False, deconvolution will be used for up-sampling.

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.

training: bool

class pymic.net.net3d.unet2d5.UpBlock(in_channels1, in_channels2, out_channels, dim=2, dropout_p=0.0, bilinear=True)

Bases: Module

Upsampling followed by ConvBlockND block

Parameters

- `in_channels1` – (int) Input channel number for low-resolution feature map.
- `in_channels2` – (int) Input channel number for high-resolution feature map.
- `out_channels` – (int) Output channel number.
- `dim` – (int) Should be 2 or 3, for 2D and 3D convolution, respectively.
- `dropout_p` – (int) Dropout probability.
- `bilinear` – (bool) Use bilinear for up-sampling or not.
**forward**$(x_1, x_2)$

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

**pymic.net.net3d.unet3d module**

**class** `pymic.net.net3d.unet3d.ConvBlock`(*in_channels, out_channels, dropout_p*)

**Bases:** `Module`

Two 3D convolution layers with batch norm and leaky relu. Dropout is used between the two convolution layers.

**Parameters**

- **in_channels** – (int) Input channel number.
- **out_channels** – (int) Output channel number.
- **dropout_p** – (int) Dropout probability.

**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.

**training:** bool

**class** `pymic.net.net3d.unet3d.DownBlock`(*in_channels, out_channels, dropout_p*)

**Bases:** `Module`

3D downsampling followed by ConvBlock

**Parameters**

- **in_channels** – (int) Input channel number.
- **out_channels** – (int) Output channel number.
- **dropout_p** – (int) Dropout probability.

**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.

```
training: bool
class pymic.net.net3d.unet3d.UNet3D(params)
    Bases: Module
    An implementation of the U-Net.
    Note that there are some modifications from the original paper, such as the use of batch normalization, dropout, leaky relu and deep supervision.
    Parameters are given in the params dictionary, and should include the following fields:

    Parameters
    • in_chns – (int) Input channel number.
    • feature_chns – (list) Feature channel for each resolution level. The length should be 4 or 5, such as [16, 32, 64, 128, 256].
    • dropout – (list) The dropout ratio for each resolution level. The length should be the same as that of feature_chns.
    • class_num – (int) The class number for segmentation task.
    • trilinear – (bool) Using trilinear for up-sampling or not. If False, deconvolution will be used for up-sampling.
    • deep_supervise – (bool) Using deep supervision for training or not.

    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.

training: bool
class pymic.net.net3d.unet3d.UpBlock(in_channels1, in_channels2, out_channels, dropout_p, trilinear=True)
    Bases: Module
    3D upsampling followed by ConvBlock

    Parameters
    • in_channels1 – (int) Channel number of high-level features.
    • in_channels2 – (int) Channel number of low-level features.
    • out_channels – (int) Output channel number.
```
• dropout_p – (int) Dropout probability.
• trilinear – (bool) Use trilinear for up-sampling (by default). If False, deconvolution is used for up-sampling.

```python
def forward(x1, x2)
    # Define 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
defining: bool
```

### pymic.net.net3d.unet3d_scse module

#### class pymic.net.net3d.unet3d_scse.ConvScSEBlock3D

```python
class ConvScSEBlock3D(in_channels, out_channels, dropout_p)

Bases: Module

Two 3D convolutional blocks followed by `ChannelSpatialSELayer3D`. Each block consists of `Conv3d + BatchNorm3d + LeakyReLU`. A dropout layer is used between the two blocks.

**Parameters**

- **in_channels** – (int) Input channel number.
- **out_channels** – (int) Output channel number.
- **dropout_p** – (int) Dropout probability.

```python
def forward(x)
    # Define 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
defining: bool
```

#### class pymic.net.net3d.unet3d_scse.DownBlock

```python
class DownBlock(in_channels, out_channels, dropout_p)

Bases: Module

3D Downsampling followed by `ConvScSEBlock3D`.

**Parameters**

- **in_channels** – (int) Input channel number.
- **out_channels** – (int) Output channel number.
- **dropout_p** – (int) Dropout probability.
```
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.

training: bool
class pymic.net.net3d.unet3d_scse.UNet3D_ScSE(params)
Bases: Module
Combining 3D U-Net with SCSE module.


Parameters are given in the params dictionary, and should include the following fields:

Parameters
• in_chns – (int) Input channel number.
• feature_chns – (list) Feature channel for each resolution level. The length should be 5, such as [16, 32, 64, 128, 256].
• dropout – (list) The dropout ratio for each resolution level. The length should be the same as that of feature_chns.
• class_num – (int) The class number for segmentation task.
• trilinear – (bool) Using trilinear for up-sampling or not. If False, deconvolution will be used for up-sampling.

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.

training: bool
class pymic.net.net3d.unet3d_scse.UpBlock(in_channels1, in_channels2, out_channels, dropout_p, trilinear=True)
Bases: Module
3D Up-sampling followed by ConvScSEBlock3D in UNet3D_ScSE.

Parameters
• in_channels1 – (int) Input channel number for low-resolution feature map.
• in_channels2 – (int) Input channel number for high-resolution feature map.
• **out_channels** – (int) Output channel number.
• **dropout_p** – (int) Dropout probability.
• **trilinear** – (bool) Use trilinear for up-sampling or not.

```python
forward(x1, x2)
```
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
training: bool
```

### Module contents

#### 3.4.2 Submodules

#### 3.4.3 pymic.net.net_dict_cls module

Built-in networks for classification.

- resnet18: `pymic.net.cls.torch_pretrained_net.ResNet18`
- vgg16: `pymic.net.cls.torch_pretrained_net.VGG16`
- mobilenetv2: `pymic.net.cls.torch_pretrained_net.MobileNetV2`

#### 3.4.4 pymic.net.net_dict_seg module

Built-in networks for segmentation.

- UNet2D: `pymic.net.net2d.unet2d.UNet2D`
- UNet2D_DualBranch: `pymic.net.net2d.unet2d_dual_branch.UNet2D_DualBranch`
- UNet2D_URPC: `pymic.net.net2d.unet2d_urpc.UNet2D_URPC`
- UNet2D_CCT: `pymic.net.net2d.unet2d_cct.UNet2D_CCT`
- UNet2D_ScSE: `pymic.net.net2d.unet2d_scse.UNet2D_ScSE`
- AttentionUNet2D: `pymic.net.net2d.unet2d_attention.AttentionUNet2D`
- NestedUNet2D: `pymic.net.net2d.unet2d_nest.NestedUNet2D`
- COPLENet: `pymic.net.net2d.cople_net.COPLENet`
- UNet2D5: `pymic.net.net3d.unet2d5.UNet2D5`
- UNet3D: `pymic.net.net3d.unet3d.UNet3D`
- UNet3D_ScSE: `pymic.net.net3d.unet3d_scse.UNet3D_ScSE`
3.4.5 Module contents

3.5 pymic.net_run package

3.5.1 Submodules

3.5.2 pymic.net_run.agent_abstract module

class pymic.net_run.agent_abstract.NetRunAgent(config, stage='train')
    Bases: object

    The abstract class for medical image segmentation.

    Parameters
    ----------
    • config – (dict) A dictionary containing the configuration.
    • stage – (str) One of the stage in train (default), inference or test.

    Note: The config dictionary should have at least four sections: dataset, network, training and inference. See Quick Start and Fully Supervised Learning for example.

    convert_tensor_type(input_tensor)
        Convert the type of an input tensor to float or double based on configuration.

    create_dataset()
        Create datasets for training, validation or testing based on configuraiton.

    abstract create_loss_calculator()
        Create loss function object.

    abstract create_network()
        Create network based on configuration.

    create_optimizer(params)
        Create optimizer based on configuration.

    Parameters
    ----------
    params – network parameters for optimization. Usually it is obtained by self.get_parameters_to_update().

    get_checkpoint_name()
        Get the checkpoint name for inference based on config['testing']['ckpt_mode'].

    abstract get_loss_value(data, pred, gt, param=None)
        Get the loss value. Assume pred and gt has been sent to self.device. data is obtained by dataloader, and is a dictionary containing extra information, such as pixel-level weight. By default, such information is not used by standard loss functions such as Dice loss and cross entropy loss.

    Parameters
    ----------
    • data – (dictionary) A data dictionary obtained by dataloader.
    • pred – (tensor) Prediction result by the network.
    • gt – (tensor) Ground truth.
• **param** – (dictionary) Other parameters if needed.

**abstract get_parameters_to_update()**
Get parameters for update.

**abstract get_stage_dataset_from_config(stage)**
Create dataset based on training, validation or inference stage.

Parameters
- **stage** – (str) `train`, `valid` or `test`.

**abstract infer()**
Inference on testing set.

**run()**
Run the training or inference code according to configuration.

**set_datasets(train_set, valid_set, test_set)**
Set customized datasets for training and inference.

Parameters
- **test_set** – (torch.utils.data.Dataset) The testing set.

**set_inferer(inferer)**
Set the inferer.

Parameters
- **inferer** – An inferer object.

**set_loss_dict(loss_dict)**
Set the available loss functions, including customized loss functions.

Parameters
- **loss_dict** – (dictionary) A dictionary of available loss functions.

**set_net_dict(net_dict)**
Set the available networks, including customized networks.

Parameters
- **net_dict** – (dictionary) A dictionary of available networks.

**set_network(net)**
Set the network.

Parameters
- **net** – (nn.Module) A deep learning network.

**set_optimizer(optimizer)**
Set the optimizer.

Parameters
- **optimizer** – An optimizer.

**set_scheduler(scheduler)**
Set the learning rate scheduler.

Parameters
- **scheduler** – A learning rate scheduler.
**set_transform_dict**(*custom_transform_dict*)

Set the available Transforms, including customized Transforms.

**Parameters**
- **custom_transform_dict** – (dictionary) A dictionary of available Transforms.

**abstract train_valid()**
Train and valid.

**abstract training()**
Train the network

**abstract validation()**
Evaluate the performance on the validation set.

**abstract write_scalars(**
*train_scalars, valid_scalars, lr_value, glob_it*)**
Write scalars using SummaryWriter.

**Parameters**
- **train_scalars** – (dictionary) Scalars for training set.
- **valid_scalars** – (dictionary) Scalars for validation set.
- **lr_value** – (float) Current learning rate.
- **glob_it** – (int) Current iteration number.

**pymic.net_run.agent_abstract.seed_torch**(*seed=1*)
Set random seed.

**Parameters**
- **seed** – (int) the seed for random.

### 3.5.3 **pymic.net_run.agent_cls** module

**class** **pymic.net_run.agent_cls.ClassificationAgent**(*config, stage='train'*)

**Bases:** NetRunAgent

The agent for image classification tasks.

**Parameters**
- **config** – (dict) A dictionary containing the configuration.
- **stage** – (str) One of the stage in train (default), inference or test.

**Note:** The config dictionary should have at least four sections: dataset, network, training and inference. See Quick Start and Fully Supervised Learning for example.

**create_loss_calculator()**
Create loss function object.

**create_network()**
Create network based on configuration.
get_evaluation_score(outputs, labels)
Get evaluation score for a prediction.

Parameters

- **outputs** – (tensor) Prediction obtained by a network.
- **labels** – (tensor) The ground truth.

get_loss_value(data, pred, gt, param=None)
Get the loss value. Assume `pred` and `gt` has been sent to `self.device`. `data` is obtained by dataloader, and is a dictionary containing extra information, such as pixel-level weight. By default, such information is not used by standard loss functions such as Dice loss and cross entropy loss.

Parameters

- **data** – (dictionary) A data dictionary obtained by dataloader.
- **pred** – (tensor) Prediction result by the network.
- **gt** – (tensor) Ground truth.
- **param** – (dictionary) Other parameters if needed.

get_parameters_to_update()
Get parameters for update.

get_stage_dataset_from_config(stage)
Create dataset based on training, validation or inference stage.

Parameters

- **stage** – (str) `train`, `valid` or `test`.

infer()
Inference on testing set.

train_valid()
Train and valid.

training()
Train the network

validation()
Evaluate the performance on the validation set.

write_scalars(train_scalars, valid_scalars, lr_value, glob_it)
Write scalars using SummaryWriter.

Parameters

- **train_scalars** – (dictionary) Scalars for training set.
- **valid_scalars** – (dictionary) Scalars for validation set.
- **lr_value** – (float) Current learning rate.
- **glob_it** – (int) Current iteration number.
3.5.4 pymic.net_run.agent_seg module

```python
class pymic.net_run.agent_seg.SegmentationAgent(config, stage='train')
    Bases: NetRunAgent

class create_loss_calculator()
    Create loss function object.

class create_network()
    Create network based on configuration.

class get_loss_value(data, pred, gt, param=None)
    Get the loss value. Assume pred and gt has been sent to self.device. data is obtained by dataloader, and is a dictionary containing extra information, such as pixel-level weight. By default, such information is not used by standard loss functions such as Dice loss and cross entropy loss.

    Parameters
    • data – (dictionary) A data dictionary obtained by dataloader.
    • pred – (tensor) Prediction result by the network.
    • gt – (tensor) Ground truth.
    • param – (dictionary) Other parameters if needed.

class get_parameters_to_update()
    Get parameters for update.

class get_stage_dataset_from_config(stage)
    Create dataset based on training, validation or inference stage.

    Parameters
    stage – (str) train, valid or test.

class infer()
    Inference on testing set.

class infer_with_multiple_checkpoints()
    Inference with ensemble of multiple check points.

class save_outputs(data)
    Save prediction output.

    Parameters
    data – (dictionary) A data dictionary with prediction result and other information such as input image name.

class set_postprocessor(postprocessor)
    Set post processor after prediction.

    Parameters
    postprocessor – post processor, such as an instance of pymic.util.post_process.PostProcess.

class train_valid()
    Train and valid.

class training()
    Train the network
```
validation()
Evaluate the performance on the validation set.

write_scalars(train_scalars, valid_scalars, lr_value, glob_it)
Write scalars using SummaryWriter.

Parameters
- **train_scalars** – (dictionary) Scalars for training set.
- **valid_scalars** – (dictionary) Scalars for validation set.
- **lr_value** – (float) Current learning rate.
- **glob_it** – (int) Current iteration number.

3.5.5 pymic.net_run.get_optimizer module

pymic.net_run.get_optimizer.get_lr_scheduler(optimizer, sched_params)
pymic.net_run.get_optimizer.get_optimizer(name, net_params, optim_params)

3.5.6 pymic.net_run.infer_func module

class pymic.net_run.infer_func.Inferer(config)
Bases: object
The class for inference. The arguments should be written in the config dictionary, and it has the following fields:

Parameters
- **sliding_window_enable** – (optional, bool) Default is False.
- **sliding_window_size** – (optional, list) The sliding window size.
- **sliding_window_stride** – (optional, list) The sliding window stride.
- **tta_mode** – (optional, int) The test time augmentation mode. Default is 0 (no test time augmentation). The other option is 1 (augmentation with horizontal and vertical flipping).

run(model, image)
Using model for inference on image.

Parameters
- **model** – (nn.Module) a network.
- **image** – (tensor) An image.
3.5.7 `pymic.net_run.net_run module`

`pymic.net_run.net_run.main()`

The main function for running a network for training or inference.

3.5.8 Module contents

3.6 `pymic.net_run_nll package`

3.6.1 Submodules

3.6.2 `pymic.net_run_nll.nll_clslsr module`

```python
class pymic.net_run_nll.nll_clslsr.NLLCLSLSR(config, stage='test')

Bases: SegmentationAgent

An agent to estimate the confidence of noisy labels during inference.


Parameters

- `config` – (dict) A dictionary containing the configuration.
- `stage` – (str) One of the stage in `train` (default), `inference` or `test`.

`infer_with_cl()`

Inference with confidence estimation.

`pymic.net_run_nll.nll_clslsr.get_confidence_map()`

The main function to get the confidence map during inference.

`pymic.net_run_nll.nll_clslsr.get_confident_map(gt, pred, CL_type='both')`

Get the confidence map based on the label and prediction.

Parameters

- `gt` – (tensor) One-hot label with shape of NXC.
- `pred` – (tensor) Digit prediction of network with shape of NXC.
- `CL_type` – (str) A string in {'both', 'Qij', 'Cij', 'intersection', 'union', 'prune_by_class', 'prune_by_noise_rate'}.

Returns

A tensor representing the noisiness of each pixel.
3.6.3 pymic.net_run_nll.nll_co_teaching module

class pymic.net_run_nll.nll_co_teaching.BiNet(params)
    Bases: 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.

training: bool

class pymic.net_run_nll.nll_co_teaching.NLLCoTeaching(config, stage='train')
    Bases: SegmentationAgent

    Co-teaching for noisy-label learning.

        • Reference: Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, Masashi
          201.

    Parameters
        • config – (dict) A dictionary containing the configuration.
        • stage – (str) One of the stage in train (default), inference or test.

        Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference)
        used in fully supervised learning, an extra section noisy_label_learning is needed. See Noisy Label Learning
        for details.

create_network()
    Create network based on configuration.

training()
    Train the network

write_scalars(train_scalars, valid_scalars, lr_value, glob_it)
    Write scalars using SummaryWriter.

    Parameters
        • train_scalars – (dictionary) Scalars for training set.
        • valid_scalars – (dictionary) Scalars for validation set.
        • lr_value – (float) Current learning rate.
        • glob_it – (int) Current iteration number.
3.6.4 pymic.net_run_nll.nll_dast module

class pymic.net_run_nll.nll_dast.ConsistLoss
   Bases: Module

   forward(input1, input2, size_average=True)
       Defines the computation performed at every call.
       Should be overridden by all subclasses.

   kl_div_map(input, label)

   kl_loss(input, target, size_average=True)

   training: bool

class pymic.net_run_nll.nll_dast.NLLDAST(config, stage='train')
   Bases: SegmentationAgent

   Divergence-Aware Selective Training for noisy label learning.


   Parameters

   • config – (dict) A dictionary containing the configuration.
   • stage – (str) One of the stage in train (default), inference or test.

   create_dataset()
       Create datasets for training, validation or testing based on configuraiton.

   get_noisy_dataset_from_config()
       Create a dataset for images with noisy labels based on configuraiton.

   train_valid()
       Train and valid.

   training()
       Train the network

class pymic.net_run_nll.nll_dast.Rank(queue_length=100)
   Bases: object

   Dynamically rank the current training sample with specific metrics.
Parameters

queue_length – (int) The length for a queue.

add_val(val)
Update the queue and calculate the order of the input value.

Parameters

val – (float) a value adding to the queue.

Returns

rank of the input value with a range of (0, self.queue_length)

pymic.net_run_nll.nll_dast.get_ce(prob, soft_y, size_avg=True)

pymic.net_run_nll.nll_dast.select_criterion(no_noisy_sample, cl_noisy_sample, label)
Obtain the sample selection criterion score.

Parameters

• no_noisy_sample – noisy branch’s output probability for noisy sample.
• cl_noisy_sample – clean branch’s output probability for noisy sample.
• label – noisy label.

3.6.5 pymic.net_run_nll.nll_main module

pymic.net_run_nll.nll_main.main()
The main function for noisy label learning methods.

3.6.6 pymic.net_run_nll.nll_trinet module

class pymic.net_run_nll.nll_trinet.NLLTriNet(config, stage='train')
Bases: SegmentationAgent

Implementation of tri-net for learning from noisy samples for segmentation tasks.


Parameters

• config – (dict) A dictionary containing the configuration.
• stage – (str) One of the stage in train (default), inference or test.

Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section noisy_label_learning is needed. See Noisy Label Learning for details.

create_network()
Create network based on configuration.

get_loss_and_confident_mask(pred, labels_prob, conf_ratio)
training()
Train the network

write_scalars(train_scalars, valid_scalars, lr_value, glob_it)
Write scalars using SummaryWriter.

Parameters

- train_scalars -- (dictionary) Scalars for training set.
- valid_scalars -- (dictionary) Scalars for validation set.
- lr_value -- (float) Current learning rate.
- glob_it -- (int) Current iteration number.

class pymic.net_run_nll.nll_trinet.TriNet(params)
Bases: 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.

training: bool

3.6.7 Module contents

3.7 pymic.net_run_ssl package

3.7.1 Submodules

3.7.2 pymic.net_run_ssl.ssl_abstract module

class pymic.net_run_ssl.ssl_abstract.SSLSegAgent(config, stage='train')
Bases: SegmentationAgent
Abstract class for semi-supervised segmentation.

Parameters

- config -- (dict) A dictionary containing the configuration.
- stage -- (str) One of the stage in train (default), inference or test.

Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section semi_supervised_learning is needed. See Semi-Supervised Learning for details.
create_dataset()
Create datasets for training, validation or testing based on configuration.

get_unlabeled_dataset_from_config()
Create a dataset for the unlabeled images based on configuration.

train_valid()
Train and valid.

write_scalars(train_scalars, valid_scalars, lr_value, glob_it)
Write scalars using SummaryWriter.

Parameters
- train_scalars – (dictionary) Scalars for training set.
- valid_scalars – (dictionary) Scalars for validation set.
- lr_value – (float) Current learning rate.
- glob_it – (int) Current iteration number.

3.7.3 pymic.net_run_ssl.ssl_cct module

class pymic.net_run_ssl.ssl_cct.SSLCCT(config, stage='train')
Bases: SSLSegAgent
Cross-Consistency Training for semi-supervised segmentation. It requires a network with multiple decoders for learning, such as pymic.net.net2d.unet2d_cct.UNet2D_CCT.

The Code is adapted from Github

Parameters
- config – (dict) A dictionary containing the configuration.
- stage – (str) One of the stage in train (default), inference or test.

Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section semi_supervised_learning is needed. See Semi-Supervised Learning for details.

training()
Train the network

pymic.net_run_ssl.ssl_cct.softmax_js_loss(inputs, targets, **_)

pymic.net_run_ssl.ssl_cct.softmax_kl_loss(inputs, targets, conf_mask=False, threshold=None, use_softmax=False)

pymic.net_run_ssl.ssl_cct.softmax_mse_loss(inputs, targets, conf_mask=False, threshold=None, use_softmax=False)
### 3.7.4 pymic.net_run_ssl.ssl_cps module

**class** pymic.net_run_ssl.ssl_cps.BiNet(*params*)

    Bases: 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.

    **training:** `bool`

**class** pymic.net_run_ssl.ssl_cps.SSLCPS(*config, stage='train'*)

    Bases: SSLSegAgent

    Using cross pseudo supervision for semi-supervised segmentation.


    **Parameters**

    - **config** – (dict) A dictionary containing the configuration.
    - **stage** – (str) One of the stage in `train` (default), `inference` or `test`.

    **Note:** In the configuration dictionary, in addition to the four sections (`dataset, network, training` and `inference`) used in fully supervised learning, an extra section `semi_supervised_learning` is needed. See *Semi-Supervised Learning* for details.

    **create_network()**
    
    Create network based on configuration.

    **training()**
    
    Train the network

    **write_scalars**(train_scalars, valid_scalars, lr_value, glob_it)
    
    Write scalars using SummaryWriter.

    **Parameters**

    - **train_scalars** – (dictionary) Scalars for training set.
    - **valid_scalars** – (dictionary) Scalars for validation set.
    - **lr_value** – (float) Current learning rate.
    - **glob_it** – (int) Current iteration number.
3.7.5 pymic.net_run_ssl.ssl_em module

class pymic.net_run_ssl.ssl_em.SSLEntropyMinimization(config, stage='train')
   Bases: SSLSegAgent

Using Entropy Minimization for semi-supervised segmentation.


Parameters

- **config** – (dict) A dictionary containing the configuration.
- **stage** – (str) One of the stage in train (default), inference or test.

Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section semi_supervised_learning is needed. See Semi-Supervised Learning for details.

```python
training()
```
Train the network

3.7.6 pymic.net_run_ssl.ssl_main module

pymic.net_run_ssl.ssl_main.main()

Main function for running a semi-supervised method.

3.7.7 pymic.net_run_ssl.ssl_mt module

class pymic.net_run_ssl.ssl_mt.SSLMeanTeacher(config, stage='train')
   Bases: SSLSegAgent

Mean Teacher for semi-supervised segmentation.


Parameters

- **config** – (dict) A dictionary containing the configuration.
- **stage** – (str) One of the stage in train (default), inference or test.

Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section semi_supervised_learning is needed. See Semi-Supervised Learning for details.

```python
create_network()
```
Create network based on configuration.
training()
Train the network

3.7.8 pymic.net_run_ssl.ssl_uamt module
class pymic.net_run_ssl.ssl_uamt.SSLUncertaintyAwareMeanTeacher(config, stage='train')
Bases: SSLMeanTeacher
Uncertainty Aware Mean Teacher for semi-supervised segmentation.


Parameters
- config – (dict) A dictionary containing the configuration.
- stage – (str) One of the stage in train (default), inference or test.

Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section semi_supervised_learning is needed. See Semi-Supervised Learning for details.

training()
Train the network

3.7.9 pymic.net_run_ssl.ssl_urpc module
class pymic.net_run_ssl.ssl_urpc.SSLURPC(config, stage='train')
Bases: SSLSegAgent
Uncertainty-Rectified Pyramid Consistency for semi-supervised segmentation.


Parameters
- config – (dict) A dictionary containing the configuration.
- stage – (str) One of the stage in train (default), inference or test.

Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section semi_supervised_learning is needed. See Semi-Supervised Learning for details.

training()
Train the network
3.7.10 Module contents

3.8 pymic.net_run_wsl package

3.8.1 Submodules

3.8.2 pymic.net_run_wsl.wsl_abstract module

```python
class pymic.net_run_wsl.wsl_abstract.WSLSegAgent(config, stage='train')
```

Bases: `SegmentationAgent`

Abstract agent for weakly supervised segmentation.

**Parameters**
- `config` – (dict) A dictionary containing the configuration.
- `stage` – (str) One of the stage in `train` (default), `inference` or `test`.

**Note:** In the configuration dictionary, in addition to the four sections (`dataset`, `network`, `training` and `inference`) used in fully supervised learning, an extra section `weakly_supervised_learning` is needed. See *Weakly-Supervised Learning* for details.

```python
write_scalars(train_scalars, valid_scalars, lr_value, glob_it)
```

Write scalars using SummaryWriter.

**Parameters**
- `train_scalars` – (dictionary) Scalars for training set.
- `valid_scalars` – (dictionary) Scalars for validation set.
- `lr_value` – (float) Current learning rate.
- `glob_it` – (int) Current iteration number.

3.8.3 pymic.net_run_wsl.wsl_dmpls module

```python
class pymic.net_run_wsl.wsl_dmpls.WSLDMPLS(config, stage='train')
```

Bases: `WSLSegAgent`

Weakly supervised segmentation based on Dynamically Mixed Pseudo Labels Supervision.


**Parameters**
- `config` – (dict) A dictionary containing the configuration.
- `stage` – (str) One of the stage in `train` (default), `inference` or `test`.
Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section weakly_supervised_learning is needed. See Weakly-Supervised Learning for details.

training()
Train the network

### 3.8.4 pymic.net_run_wsl.wsl_em module

class pymic.net_run_wsl.wsl_em.WSLEntropyMinimization(config, stage='train')
Bases: WSLSegAgent

Weakly supervised segmentation based on Entropy Minimization.


Parameters

- config – (dict) A dictionary containing the configuration.
- stage – (str) One of the stage in train (default), inference or test.

Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section weakly_supervised_learning is needed. See Weakly-Supervised Learning for details.

training()
Train the network

### 3.8.5 pymic.net_run_wsl.wsl_gatedcrf module

class pymic.net_run_wsl.wsl_gatedcrf.WSLGatedCRF(config, stage='train')
Bases: WSLSegAgent

Implementation of the Gated CRF loss for weakly supervised segmentation.


Parameters

- config – (dict) A dictionary containing the configuration.
- stage – (str) One of the stage in train (default), inference or test.

Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section weakly_supervised_learning is needed. See Weakly-Supervised Learning for details.
training()
    Train the network

3.8.6 pymic.net_run_wsl.wsl_main module

pymic.net_run_wsl.wsl_main.main()
    The main function for training and inference of weakly supervised segmentation.

3.8.7 pymic.net_run_wsl.wsl_mumford_shah module

class pymic.net_run_wsl.wsl_mumford_shah.WSMumfordShah(config, stage='train')
    Bases: WSLSegAgent
    Weakly supervised learning with Mumford Shah Loss.
    • Reference: Boah Kim and Jong Chul Ye: Mumford–Shah Loss Functional for Image Segmentation With

    Parameters
    • config – (dict) A dictionary containing the configuration.
    • stage – (str) One of the stage in train (default), inference or test.

    Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section weakly_supervised_learning is needed. See Weakly-Supervised Learning for details.

training()
    Train the network

3.8.8 pymic.net_run_wsl.wsl_tv module

class pymic.net_run_wsl.wsl_tv.WSLTotalVariation(config, stage='train')
    Bases: WSLSegAgent
    Weakly supervised segmentation with Total Variation regularization.

    Parameters
    • config – (dict) A dictionary containing the configuration.
    • stage – (str) One of the stage in train (default), inference or test.

    Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section weakly_supervised_learning is needed. See Weakly-Supervised Learning for details.

training()
    Train the network
3.8.9 `pymic.net_run_wsl.wsl_ustm` module

```python
class pymic.net_run_wsl.wsl_ustm.WSLUSTM(config, stage='train')
    Bases: WSLSegAgent

USTM for scribble-supervised segmentation.


Parameters

- `config` – (dict) A dictionary containing the configuration.
- `stage` – (str) One of the stage in `train` (default), `inference` or `test`.

Note: In the configuration dictionary, in addition to the four sections (dataset, network, training and inference) used in fully supervised learning, an extra section **weakly_supervised_learning** is needed. See Weakly-Supervised Learning for details.

create_network()
Create network based on configuration.

training()
Train the network
```

3.8.10 Module contents

3.9 `pymic.transform` package

3.9.1 Submodules

3.9.2 `pymic.transform.abstract_transform` module

```python
class pymic.transform.abstract_transform.AbstractTransform(params)
    Bases: object

The abstract class for Transform.

inverse_transform_for_prediction(sample)
Inverse transform for the sample dictionary. Especially, it will update sample[‘predict’] obtained by a network’s prediction based on the inverse transform. This function is only useful for spatial transforms.
```
3.9.3 pymic.transform.crop module

class pymic.transform.crop.CenterCrop(params)
    Bases: AbstractTransform
    Crop the given image at the center. Input shape should be [C, D, H, W] or [C, H, W].
    The arguments should be written in the params dictionary, and it has the following fields:

    Parameters
    • CenterCrop_output_size – (list or tuple) The output size. [D, H, W] for 3D images and [H, W] for 2D images. If D is None, then the z-axis is not cropped.
    • CenterCrop_inverse – (optional, bool) Is inverse transform needed for inference. Default is True.

    inverse_transform_for_prediction(sample)
    Inverse transform for the sample dictionary. Especially, it will update sample['predict'] obtained by a network’s prediction based on the inverse transform. This function is only useful for spatial transforms.

class pymic.transform.crop.CropWithBoundingBox(params)
    Bases: CenterCrop
    Crop the image (shape [C, D, H, W] or [C, H, W]) based on a bounding box. The arguments should be written in the params dictionary, and it has the following fields:

    Parameters
    • CropWithBoundingBox_start – (None, or list/tuple) The start index along each spatial axis. If None, calculate the start index automatically so that the cropped region is centered at the non-zero region.
    • CropWithBoundingBox_output_size – (None or tuple/list): Desired spatial output size. If None, set it as the size of bounding box of non-zero region.
    • CropWithBoundingBox_inverse – (optional, bool) Is inverse transform needed for inference. Default is True.

class pymic.transform.crop.RandomCrop(params)
    Bases: CenterCrop
    Randomly crop the input image (shape [C, D, H, W] or [C, H, W]).
    The arguments should be written in the params dictionary, and it has the following fields:

    Parameters
    • RandomCrop_output_size – (list/tuple) Desired output size [D, H, W] or [H, W]. The output channel is the same as the input channel. If D is None for 3D images, the z-axis is not cropped.
    • RandomCrop_foreground_focus – (optional, bool) If true, allow crop around the foreground. Default is False.
    • RandomCrop_foreground_ratio – (optional, float) Specifying the probability of foreground focus cropping when RandomCrop_foreground_focus is True.
    • RandomCrop_mask_label – (optional, None, or list/tuple) Specifying the foreground labels for foreground focus cropping when RandomCrop_foreground_focus is True.
    • RandomCrop_inverse – (optional, bool) Is inverse transform needed for inference. Default is True.
class pymic.transform.crop.RandomResizedCrop(params)

Bases: CenterCrop

Randomly crop the input image (shape [C, H, W]). Only 2D images are supported.

The arguments should be written in the params dictionary, and it has the following fields:

Parameters

- \textbf{RandomResizedCrop\_output\_size} – (list/tuple) Desired output size [H, W]. The output channel is the same as the input channel.
- \textbf{RandomResizedCrop\_scale} – (list/tuple) Range of scale, e.g. (0.08, 1.0).
- \textbf{RandomResizedCrop\_ratio} – (list/tuple) Range of aspect ratio, e.g. (0.75, 1.33).
- \textbf{RandomResizedCrop\_inverse} – (optional, bool) Is inverse transform needed for inference. Default is \textit{False}. Currently, the inverse transform is not supported, and this transform is assumed to be used only during training stage.

3.9.4 pymic.transform.flip module

class pymic.transform.flip.RandomFlip(params)

Bases: AbstractTransform

Random flip the image. The shape is [C, D, H, W] or [C, H, W].

The arguments should be written in the params dictionary, and it has the following fields:

Parameters

- \textbf{RandomFlip\_flip\_depth} – (bool) Random flip along depth axis or not, only used for 3D images.
- \textbf{RandomFlip\_flip\_height} – (bool) Random flip along height axis or not.
- \textbf{RandomFlip\_flip\_width} – (bool) Random flip along width axis or not.
- \textbf{RandomFlip\_inverse} – (optional, bool) Is inverse transform needed for inference. Default is True.

inverse\_transform\_for\_prediction(sample)

Inverse transform for the sample dictionary. Especially, it will update sample['predict'] obtained by a network’s prediction based on the inverse transform. This function is only useful for spatial transforms.

3.9.5 pymic.transform.intensity module

class pymic.transform.intensity.GammaCorrection(params)

Bases: AbstractTransform

Apply random gamma correction to given channels.

The arguments should be written in the params dictionary, and it has the following fields:

Parameters

- \textbf{GammaCorrection\_channels} – (list) A list of int for specifying the channels.
- \textbf{GammaCorrection\_gamma\_min} – (float) The minimal gamma value.
- \textbf{GammaCorrection\_gamma\_max} – (float) The maximal gamma value.
• **GammaCorrection_probability** – (optional, float) The probability of applying Gamma-Correction. Default is 0.5.

• **GammaCorrection_inverse** – (optional, bool) Is inverse transform needed for inference. Default is *False*.

class pymic.transform.intensity.GaussianNoise(params)

    Bases: AbstractTransform

    Add Gaussian Noise to given channels.

    The arguments should be written in the `params` dictionary, and it has the following fields:

    Parameters

    • **GaussianNoise_channels** – (list) A list of int for specifying the channels.
    • **GaussianNoise_mean** – (float) The mean value of noise.
    • **GaussianNoise_std** – (float) The std of noise.
    • **GaussianNoise_probability** – (optional, float) The probability of applying Gaussian-Noise. Default is 0.5.
    • **GaussianNoise_inverse** – (optional, bool) Is inverse transform needed for inference. Default is *False*.

class pymic.transform.intensity.GrayscaleToRGB(params)

    Bases: AbstractTransform

    Convert gray scale images to RGB by copying channels.

3.9.6 pymic.transform.label_convert module

class pymic.transform.label_convert.LabelConvert(params)

    Bases: AbstractTransform

    Convert the label based on a source list and target list.

    The arguments should be written in the `params` dictionary, and it has the following fields:

    Parameters

    • **LabelConvert_source_list** – (list) A list of labels to be converted.
    • **LabelConvert_target_list** – (list) The target label list.
    • **LabelConvert_inverse** – (optional, bool) Is inverse transform needed for inference. Default is *False*.

class pymic.transform.label_convert.LabelConvertNonzero(params)

    Bases: AbstractTransform

    Convert label into binary, i.e., setting nonzero labels as 1.

class pymic.transform.label_convert.LabelToProbability(params)

    Bases: AbstractTransform

    Convert one-channel label map to one-hot multi-channel probability map.

    The arguments should be written in the `params` dictionary, and it has the following fields:

    Parameters
- **LabelToProbability_class_num** – (int) The class number in the label map.
- **LabelToProbability_inverse** – (optional, bool) Is inverse transform needed for inference. Default is False.

class pymic.transform.label_convert.PartialLabelToProbability(params)

Bases: AbstractTransform

Convert one-channel partial label map to one-hot multi-channel probability map. This is used for segmentation tasks only. In the input label map, 0 represents the background class, 1 to C-1 represent the foreground classes, and C represents unlabeled pixels. In the output dictionary, `label_prob` is the one-hot probability map, and `pixel_weight` represents a weighting map, where the weight for a pixel is 0 if the label is unknown.

The arguments should be written in the `params` dictionary, and it has the following fields:

Parameters

- **PartialLabelToProbability_class_num** – (int) The class number for the segmentation task.
- **PartialLabelToProbability_inverse** – (optional, bool) Is inverse transform needed for inference. Default is False.

class pymic.transform.label_convert.ReduceLabelDim(params)

Bases: AbstractTransform

Remove the first dimension of label tensor.

### 3.9.7 pymic.transform.normalize module

class pymic.transform.normalize.NormalizeWithMeanStd(params)

Bases: AbstractTransform

Normalize the image based on mean and std. The image should have a shape of [C, D, H, W] or [C, H, W].

The arguments should be written in the `params` dictionary, and it has the following fields:

Parameters

- **NormalizeWithMeanStd_channels** – (list/tuple or None) A list or tuple of int for specifying the channels. If None, the transform operates on all the channels.
- **NormalizeWithMeanStd_mean** – (list/tuple or None) The mean values along each specified channel. If None, the mean values are calculated automatically.
- **NormalizeWithMeanStd_std** – (list/tuple or None) The std values along each specified channel. If None, the std values are calculated automatically.
- **NormalizeWithMeanStd_ignore_non_positive** – (optional, bool) Only used when mean and std are not given. Default is False. If True, calculate mean and std in the positive region for normalization, and set non-positive region to random. If False, calculate the mean and std values in the entire image region.
- **NormalizeWithMeanStd_inverse** – (optional, bool) Is inverse transform needed for inference. Default is False.

class pymic.transform.normalize.NormalizeWithMinMax(params)

Bases: AbstractTransform

Normalizes the image to [0, 1]. The shape should be [C, D, H, W] or [C, H, W].

The arguments should be written in the `params` dictionary, and it has the following fields:
Parameters

- **NormalizeWithMinMax_channels** – (list/tuple or None) A list or tuple of int for specifying the channels. If None, the transform operates on all the channels.

- **NormalizeWithMinMax_threshold_lower** – (list/tuple or None) The min values along each specified channel. If None, the min values are calculated automatically.

- **NormalizeWithMinMax_threshold_upper** – (list/tuple or None) The max values along each specified channel. If None, the max values are calculated automatically.

- **NormalizeWithMinMax_inverse** – (optional, bool) Is inverse transform needed for inference. Default is False.

**class** `pymic.transform.normalize.NormalizeWithPercentiles(params)`

**Bases:** `AbstractTransform`

Normralize the image to [0, 1] with percentiles for given channels. The shape should be [C, D, H, W] or [C, H, W].

The arguments should be written in the `params` dictionary, and it has the following fields:

Parameters

- **NormalizeWithPercentiles_channels** – (list/tuple or None) A list or tuple of int for specifying the channels. If None, the transform operates on all the channels.

- **NormalizeWithPercentiles_percentile_lower** – (float) The min percentile, which must be between 0 and 100 inclusive.

- **NormalizeWithPercentiles_percentile_upper** – (float) The max percentile, which must be between 0 and 100 inclusive.

- **NormalizeWithMinMax_inverse** – (optional, bool) Is inverse transform needed for inference. Default is False.

3.9.8 pymic.transform.pad module

**class** `pymic.transform.pad.Pad(params)`

**Bases:** `AbstractTransform`

Pad an image to an new spatial shape. The image has a shape of [C, D, H, W] or [C, H, W]. The real output size will be max(image_size, output_size).

The arguments should be written in the `params` dictionary, and it has the following fields:

Parameters

- **Pad_output_size** – (list/tuple) The output size along each spatial axis.

- **Pad ceil_mode** – (optional, bool) If true (by default), the real output size will be the minimal integer multiples of output_size higher than the input size. For example, the input image has a shape of [3, 100, 100], Pad_output_size = [32, 32], and the real output size will be [3, 128, 128] if Pad ceil_mode = True.

- **Pad_inverse** – (optional, bool) Is inverse transform needed for inference. Default is True.

**inverse_transform_for_prediction(sample)**

Inverse transform for the sample dictionary. Especially, it will update sample['predict'] obtained by a network's prediction based on the inverse transform. This function is only useful for spatial transforms.
3.9.9 pymic.transform.rescale module

class pymic.transform.rescale.RandomRescale(params)
    Bases: AbstractTransform
    Rescale the input image randomly along each spatial axis.
    The arguments should be written in the params dictionary, and it has the following fields:
    Parameters
    • RandomRescale_lower_bound – (list/tuple or int) Desired minimal rescale ratio. If tuple/list, the length should be 3 or 2.
    • RandomRescale_upper_bound – (list/tuple or int) Desired maximal rescale ratio. If tuple/list, the length should be 3 or 2.
    • RandomRescale_inverse – (optional, bool) Is inverse transform needed for inference. Default is True.

    inverse_transform_for_prediction(sample)
    Inverse transform for the sample dictionary. Especially, it will update sample['predict'] obtained by a network’s prediction based on the inverse transform. This function is only useful for spatial transforms.

class pymic.transform.rescale.Rescale(params)
    Bases: AbstractTransform
    Rescale the image to a given size.
    The arguments should be written in the params dictionary, and it has the following fields:
    Parameters
    • Rescale_output_size – (list/tuple or int) The output size along each spatial axis, such as [D, H, W] or [H, W]. If D is None, the input image is only rescaled in 2D. If int, the smallest axis is matched to output_size keeping aspect ratio the same as the input.
    • Rescale_inverse – (optional, bool) Is inverse transform needed for inference. Default is True.

    inverse_transform_for_prediction(sample)
    Inverse transform for the sample dictionary. Especially, it will update sample['predict'] obtained by a network’s prediction based on the inverse transform. This function is only useful for spatial transforms.

3.9.10 pymic.transform.rotate module

class pymic.transform.rotate.RandomRotate(params)
    Bases: AbstractTransform
    Random rotate an image, with a shape of [C, D, H, W] or [C, H, W].
    The arguments should be written in the params dictionary, and it has the following fields:
    Parameters
    • RandomRotate_angle_range_d – (list/tuple or None) Rotation angle (degree) range along depth axis (x-y plane), e.g., (-90, 90). If None, no rotation along this axis.
    • RandomRotate_angle_range_h – (list/tuple or None) Rotation angle (degree) range along height axis (x-z plane), e.g., (-90, 90). If None, no rotation along this axis. Only used for 3D images.
• **RandomRotate_angle_range_w** – (list/tuple or None) Rotation angle (degree) range along width axis (y-z plane), e.g., (-90, 90). If None, no rotation along this axis. Only used for 3D images.

• **RandomRotate_inverse** – (optional, bool) Is inverse transform needed for inference. Default is `True`.

  `inverse_transform_for_prediction(sample)`

  Inverse transform for the sample dictionary. Especially, it will update sample['predict'] obtained by a network's prediction based on the inverse transform. This function is only useful for spatial transforms.

### 3.9.11 pymic.transform.threshold module

**class pymic.transform.threshold.ChannelWiseThreshold(params)**

Bases: `AbstractTransform`

Thresholding the image for given channels.

The arguments should be written in the `params` dictionary, and it has the following fields:

**Parameters**

- **ChannelWiseThreshold_channels** – (list/tuple or None) A list of specified channels for thresholding. If None (by default), all the channels will be thresholded.

- **ChannelWiseThreshold_threshold_lower** – (list/tuple or None) The lower threshold for the given channels.

- **ChannelWiseThreshold_threshold_upper** – (list/tuple or None) The upper threshold for the given channels.

- **ChannelWiseThreshold_replace_lower** – (list/tuple or None) The output value for pixels with an input value lower than the threshold_lower.

- **ChannelWiseThreshold_replace_upper** – (list/tuple or None) The output value for pixels with an input value higher than the threshold_upper.

- **ChannelWiseThreshold_inverse** – (optional, bool) Is inverse transform needed for inference. Default is `False`.

**class pymic.transform.threshold.ChannelWiseThresholdWithNormalize(params)**

Bases: `AbstractTransform`

Apply thresholding and normalization for given channels. Pixel intensity will be truncated to the range of (lower, upper) and then normalized. If mean_std_mode is True, the mean and std values for pixel in the target range is calculated for normalization, and input intensity outside that range will be replaced by random values. Otherwise, the intensity will be normalized to [0, 1].

The arguments should be written in the `params` dictionary, and it has the following fields:

**Parameters**

- **ChannelWiseThresholdWithNormalize_channels** – (list/tuple or None) A list of specified channels for thresholding. If None (by default), all the channels will be affected by this transform.

- **ChannelWiseThresholdWithNormalize_threshold_lower** – (list/tuple or None) The lower threshold for the given channels.

- **ChannelWiseThresholdWithNormalize_threshold_upper** – (list/tuple or None) The upper threshold for the given channels.
3.9.12 pymic.transform.trans_dict module

The built-in transforms in PyMIC are:

```
'ChannelWiseThreshold': ChannelWiseThreshold,
'ChannelWiseThresholdWithNormalize': ChannelWiseThresholdWithNormalize,
'CropWithBoundingBox': CropWithBoundingBox,
'CenterCrop': CenterCrop,
'GrayscaleToRGB': GrayscaleToRGB,
'GammaCorrection': GammaCorrection,
'GaussianNoise': GaussianNoise,
'LabelConvert': LabelConvert,
'LabelConvertNonzero': LabelConvertNonzero,
'LabelToProbability': LabelToProbability,
'NormalizeWithMeanStd': NormalizeWithMeanStd,
'NormalizeWithMinMax': NormalizeWithMinMax,
'NormalizeWithPercentiles': NormalizeWithPercentiles,
'PartialLabelToProbability': PartialLabelToProbability,
'RandomCrop': RandomCrop,
'RandomResizedCrop': RandomResizedCrop,
'RandomRescale': RandomRescale,
'RandomFlip': RandomFlip,
'RandomRotate': RandomRotate,
'ReduceLabelDim': ReduceLabelDim,
'Rescale': Rescale,
'Pad': Pad.
```

3.9.13 Module contents

3.10 pymic.util package

3.10.1 Submodules

3.10.2 pymic.util.evaluation_cls module

Evaluation module for classification tasks.

```
pymic.util.evaluation_cls.accuracy(gt_label, pred_label)
```
Calculate the accuracy.

```
pymic.util.evaluation_cls.binary_evaluation(config)
```
Evaluation of binary classification performance. The arguments are given in the `config` dictionary. It should have the following fields:

Parameters
3.10. pymic.util package

- **metric_list** – (list) A list of evaluation metrics. The supported metrics are \{accuracy, recall, sensitivity, specificity, precision, auc\}.
- **ground_truth_csv** – (str) The csv file for ground truth.
- **predict_prob_csv** – (str) The csv file for prediction probability.

**pymic.util.evaluation_cls.get_evaluation_score(gt_label, pred_prob, metric)**

Get an evaluation score for binary classification.

**Parameters**

- **gt_label** – (array) Ground truth label.
- **pred_prob** – (array) Predicted positive probability.
- **metric** – (str) One of the evaluation metrics in \{accuracy, recall, sensitivity, specificity, precision, auc\}.

**pymic.util.evaluation_cls.main()**

Main function for evaluation of classification results. A configuration file is needed for running. e.g.,

```
pymic_evaluate_cls config.cfg
```

The configuration file should have an `evaluation` section with the following fields:

**Parameters**

- **task_type** – (str) cls or cls_nexcl.
- **metric_list** – (list) A list of evaluation metrics. The supported metrics are \{accuracy, recall, sensitivity, specificity, precision, auc\}.
- **ground_truth_csv** – (str) The csv file for ground truth.
- **predict_prob_csv** – (str) The csv file for prediction probability.

**pymic.util.evaluation_cls.nexcl_evaluation(config)**

Evaluation of non-exclusive binary classification performance. The arguments are given in the `config` dictionary. It should have the following fields:

**Parameters**

- **metric_list** – (list) A list of evaluation metrics. The supported metrics are \{accuracy, recall, sensitivity, specificity, precision, auc\}.
- **ground_truth_csv** – (str) The csv file for ground truth.
- **predict_prob_csv** – (str) The csv file for prediction probability.

**pymic.util.evaluation_cls.sensitivity(gt_label, pred_label)**

Calculate the sensitivity for binary prediction.

**pymic.util.evaluation_cls.specificity(gt_label, pred_label)**

Calculate the specificity for binary prediction.
3.10.3 pymic.util.evaluation_seg module

3.10.4 pymic.util.general module

pymic.util.general.get_one_hot_seg(label, class_num)
    Convert a segmentation label to one-hot.

    Parameters
    • label – A tensor with a shape of [N, 1, D, H, W] or [N, 1, H, W]
    • class_num – Class number.

    Returns
    a one-hot tensor with a shape of [N, C, D, H, W] or [N, C, H, W].

pymic.util.general.keyword_match(a, b)
    Test if two string are the same when converted to lower case.

3.10.5 pymic.util.image_process module

pymic.util.image_process.convert_label(label, source_list, target_list)
    Convert a label map based a source list and a target list of labels

    Parameters
    • label – (numpy.array) The input label map.
    • source_list – A list of labels that will be converted, e.g. [0, 1, 2, 4]
    • target_list – A list of target labels, e.g. [0, 1, 2, 3]

pymic.util.image_process.crop_ND_volume_with_bounding_box(volume, bb_min, bb_max)
    Extract a subregion form an ND image.

    Parameters
    • volume – The input ND array.
    • bb_min – (list) The lower bound of the bounding box for each axis.
    • bb_max – (list) The upper bound of the bounding box for each axis.

    Returns
    A croped ND image.

pymic.util.image_process.crop_and_pad_ND_array_to_desired_shape(image, out_shape, pad_mod)
    Crop and pad an image to a given shape.

    Parameters
    • image – The input ND array.
    • out_shape – (list) The desired output shape.
    • pad_mod – (str) See numpy.pad

pymic.util.image_process.get_ND_bounding_box(volume, margin=None)
    Get the bounding box of nonzero region in an ND volume.

    Parameters
    • volume – An ND numpy array.
• **margin** – (list) The margin of bounding box along each axis.

**Return bb_min**
(list) A list for the minimal value of each axis of the bounding box.

**Return bb_max**
(list) A list for the maximal value of each axis of the bounding box.

```python
pymic.util.image_process.get_euclidean_distance(image, dim=3, spacing=[1.0, 1.0, 1.0])
```

Get euclidean distance transform of 3D binary images. The output distance map is unsigned.

**Parameters**
- **image** – The input 3D array.
- **dim** – (int) Using 2D (dim = 2) or 3D (dim = 3) distance transforms.
- **spacing** – (list) The spacing along each axis.

```python
pymic.util.image_process.get_largest_k_components(image, k=1)
```

Get the largest K components from 2D or 3D binary image.

**Parameters**
- **image** – The input ND array for binary segmentation.
- **k** – (int) The value of k.

**Returns**
An output array with only the largest K components of the input.

```python
pymic.util.image_process.resample_sitk_image_to_given_spacing(image, spacing, order)
```

Resample an sitk image object to a given spacing.

**Parameters**
- **image** – The input sitk image object.
- **spacing** – (list/tuple) Target spacing along x, y, z direction.
- **order** – (int) Order for interpolation.

**Returns**
A resampled sitk image object.

```python
pymic.util.image_process.set_ND_volume_roi_with_bounding_box_range(volume, bb_min, bb_max, sub_volume, addition=True)
```

Set the subregion of an ND image. If *addition* is True, the original volume is added by the given sub volume.

**Parameters**
- **volume** – The input ND volume.
- **bb_min** – (list) The lower bound of the bounding box for each axis.
- **bb_max** – (list) The upper bound of the bounding box for each axis.
- **sub_volume** – The sub volume to replace the target region of the original volume.
- **addition** – (optional, bool) If True, the sub volume will be added to the target region of the input volume.
## 3.10.6 `pymic.util.parse_config` module

```python
pymic.util.parse_config.is_bool(var_str)
pymic.util.parse_config.is_float(val_str)
pymic.util.parse_config.is_int(val_str)
pymic.util.parse_config.is_list(val_str)
pymic.util.parse_config.logging_config(config)
pymic.util.parse_config.parse_bool(var_str)
pymic.util.parse_config.parse_config(filename)
pymic.util.parse_config.parse_list(val_str)
pymic.util.parse_config.parse_value_from_string(val_str)
pymic.util.parse_config.synchronize_config(config)
```

## 3.10.7 `pymic.util.post_process` module

```python
class pymic.util.post_process.PostKeepLargestComponent(params):
    Bases: PostProcess
    Post process by keeping the largest component.
    The arguments should be written in the `params` dictionary, and it has the following fields:
    Parameters
    - `KeepLargestComponent_mode` – (int) 1 means keep the largest component of the union of foreground classes. 2 means keep the largest component for each foreground class.

class pymic.util.post_process.PostProcess(params):
    Bases: object
    The abstract class for post processing.
```

## 3.10.8 `pymic.util.preprocess` module

```python
pymic.util.preprocess.get_transform_list(trans_config_file)
Create a list of transforms given a configuration file.
```

```python
pymic.util.preprocess.preprocess_with_transform(transforms, img_in_name, img_out_name,
                                           lab_in_name=None, lab_out_name=None)
Using a list of data transforms for preprocessing, such as image normalization, cropping, etc. TODO: support multip-modality preprocessing.
Parameters
- `transforms` – (list) A list of transform objects.
- `img_in_name` – (str) Input file name.
- `img_out_name` – (str) Output file name.
```
3.10.9 pymic.util.ramps module

Functions for ramping hyperparameters up or down. Each function takes the current training step or epoch, and the ramp length (start and end step or epoch), and returns a multiplier between 0 and 1.

pymic.util.ramps.get_rampdown_ratio(i, start, end, mode='linear')

Obtain the rampdown ratio.

Parameters

• i – (int) The current iteration.
• start – (int) The start iteration.
• end – (int) The end iteration.
• mode – (str) Valid values are {linear, sigmoid, cosine}.

pymic.util.ramps.get_rampup_ratio(i, start, end, mode='linear')

Obtain the rampup ratio.

Parameters

• i – (int) The current iteration.
• start – (int) The start iteration.
• end – (int) The end iteration.
• mode – (str) Valid values are {linear, sigmoid, cosine}.

3.10.10 Module contents
If you use PyMIC for your research, please acknowledge it accordingly by citing our paper:


BibTeX entry:

```latex
@article{Wang2022pymic,
author = {Guotai Wang and Xiangde Luo and Ran Gu and Shuojue Yang and Yijie Qu and Shuwei Zhai and Qianfei Zhao and Kang Li and Shaoting Zhang},
title = {{PyMIC: A deep learning toolkit for annotation-efficient medical image segmentation}},
year = {2022},
url = {http://arxiv.org/abs/2208.09350},
journal = {arXiv},
volume = {2208.09350},
pages = {1-10},
}
```
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pymic.io</td>
<td>23</td>
</tr>
<tr>
<td>pymic.io.image_read_write</td>
<td>21</td>
</tr>
<tr>
<td>pymic.io.nifty_dataset</td>
<td>22</td>
</tr>
<tr>
<td>pymic.layer</td>
<td>27</td>
</tr>
<tr>
<td>pymic.layer.activation</td>
<td>23</td>
</tr>
<tr>
<td>pymic.layer.convolution</td>
<td>23</td>
</tr>
<tr>
<td>pymic.layer.deconvolution</td>
<td>25</td>
</tr>
<tr>
<td>pymic.layer.space2channel</td>
<td>27</td>
</tr>
<tr>
<td>pymic.loss</td>
<td>41</td>
</tr>
<tr>
<td>pymic.loss cls</td>
<td>30</td>
</tr>
<tr>
<td>pymic.loss cls.basic</td>
<td>27</td>
</tr>
<tr>
<td>pymic.loss cls.util</td>
<td>29</td>
</tr>
<tr>
<td>pymic.loss loss_dict_cls</td>
<td>40</td>
</tr>
<tr>
<td>pymic.loss loss_dict_seg</td>
<td>41</td>
</tr>
<tr>
<td>pymic.loss seg</td>
<td>40</td>
</tr>
<tr>
<td>pymic.loss seg.abstract</td>
<td>30</td>
</tr>
<tr>
<td>pymic.loss seg.ce</td>
<td>30</td>
</tr>
<tr>
<td>pymic.loss seg.combined</td>
<td>32</td>
</tr>
<tr>
<td>pymic.loss seg.deep_sup</td>
<td>32</td>
</tr>
<tr>
<td>pymic.loss seg.dice</td>
<td>33</td>
</tr>
<tr>
<td>pymic.loss seg.exp_log</td>
<td>35</td>
</tr>
<tr>
<td>pymic.loss seg.gatedcrf</td>
<td>35</td>
</tr>
<tr>
<td>pymic.loss seg.mse</td>
<td>36</td>
</tr>
<tr>
<td>pymic.loss seg.mumford_shah</td>
<td>37</td>
</tr>
<tr>
<td>pymic.loss seg.slsr</td>
<td>38</td>
</tr>
<tr>
<td>pymic.loss seg.ssl</td>
<td>39</td>
</tr>
<tr>
<td>pymic.loss seg.util</td>
<td>39</td>
</tr>
<tr>
<td>pymic.net</td>
<td>66</td>
</tr>
<tr>
<td>pymic.net cls</td>
<td>43</td>
</tr>
<tr>
<td>pymic.net cls.torch_pretrained_net</td>
<td>41</td>
</tr>
<tr>
<td>pymic.net net2d</td>
<td>57</td>
</tr>
<tr>
<td>pymic.net net2d.cople_net</td>
<td>43</td>
</tr>
<tr>
<td>pymic.net net2d.scse2d</td>
<td>45</td>
</tr>
<tr>
<td>pymic.net net2d.unet2d</td>
<td>47</td>
</tr>
<tr>
<td>pymic.net net2d.unet2d_attention</td>
<td>50</td>
</tr>
<tr>
<td>pymic.net net2d.unet2d_cct</td>
<td>51</td>
</tr>
<tr>
<td>pymic.net net2d.unet2d_dual_branch</td>
<td>52</td>
</tr>
<tr>
<td>pymic.net net2d.unet2d_nest</td>
<td>53</td>
</tr>
<tr>
<td>pymic.net net2d.unet2d_scse</td>
<td>53</td>
</tr>
<tr>
<td>pymic.net net2d.unet2d_urpc</td>
<td>55</td>
</tr>
<tr>
<td>pymic.net.net3d</td>
<td>65</td>
</tr>
<tr>
<td>pymic.net.net3d.scse3d</td>
<td>57</td>
</tr>
<tr>
<td>pymic.net.net3d.unet2d</td>
<td>59</td>
</tr>
<tr>
<td>pymic.net.net3d.unet2d5</td>
<td>59</td>
</tr>
<tr>
<td>pymic.net.net3d.unet3d</td>
<td>61</td>
</tr>
<tr>
<td>pymic.net.net3d.unet3d_scse</td>
<td>63</td>
</tr>
<tr>
<td>pymic.net.net_dict_cls</td>
<td>65</td>
</tr>
<tr>
<td>pymic.net.net_dict_seg</td>
<td>65</td>
</tr>
<tr>
<td>pymic.net_run</td>
<td>72</td>
</tr>
<tr>
<td>pymic.net_run.agent_abstract</td>
<td>66</td>
</tr>
<tr>
<td>pymic.net_run.agent_cls</td>
<td>68</td>
</tr>
<tr>
<td>pymic.net_run.agent_seg</td>
<td>70</td>
</tr>
<tr>
<td>pymic.net_run.get_optimizer</td>
<td>71</td>
</tr>
<tr>
<td>pymic.net_run.infer_func</td>
<td>71</td>
</tr>
<tr>
<td>pymic.net_run.net_run</td>
<td>72</td>
</tr>
<tr>
<td>pymic.net_run_nll</td>
<td>76</td>
</tr>
<tr>
<td>pymic.net_run_nll.nll_clslsr</td>
<td>72</td>
</tr>
<tr>
<td>pymic.net_run_nll.nll_co_teaching</td>
<td>73</td>
</tr>
<tr>
<td>pymic.net_run_nll.nll_dast</td>
<td>74</td>
</tr>
<tr>
<td>pymic.net_run_nll.nll_main</td>
<td>75</td>
</tr>
<tr>
<td>pymic.net_run_nll.nll_trinet</td>
<td>75</td>
</tr>
<tr>
<td>pymic.net_run_ssl</td>
<td>81</td>
</tr>
<tr>
<td>pymic.net_run_ssl.ssl_abstract</td>
<td>76</td>
</tr>
<tr>
<td>pymic.net_run_ssl.ssl_cct</td>
<td>77</td>
</tr>
<tr>
<td>pymic.net_run_ssl.ssl_cps</td>
<td>78</td>
</tr>
<tr>
<td>pymic.net_run_ssl.ssl_em</td>
<td>79</td>
</tr>
<tr>
<td>pymic.net_run_ssl.ssl_main</td>
<td>79</td>
</tr>
<tr>
<td>pymic.net_run_ssl.ssl_mt</td>
<td>79</td>
</tr>
<tr>
<td>pymic.net_run_ssl.ssl_uamt</td>
<td>80</td>
</tr>
<tr>
<td>pymic.net_run_ssl.ssl_urpc</td>
<td>80</td>
</tr>
<tr>
<td>pymic.net_run_ssl.wsl</td>
<td>84</td>
</tr>
<tr>
<td>pymic.net_run_wsl.wsl_abstract</td>
<td>81</td>
</tr>
<tr>
<td>pymic.net_run_wsl.wsl_dmplsl</td>
<td>81</td>
</tr>
<tr>
<td>pymic.net_run_wsl.wsl_em</td>
<td>82</td>
</tr>
<tr>
<td>pymic.net_run_wsl.wsl_gatedcrf</td>
<td>82</td>
</tr>
<tr>
<td>pymic.net_run_wsl.wsl_main</td>
<td>83</td>
</tr>
<tr>
<td>pymic.net_run_wsl.wsl_mumford_shah</td>
<td>83</td>
</tr>
<tr>
<td>pymic.net_run_wsl.wsl_tv</td>
<td>83</td>
</tr>
<tr>
<td>pymic.net_run_wsl.wsl_ustm</td>
<td>84</td>
</tr>
<tr>
<td>pymic.transform</td>
<td>92</td>
</tr>
<tr>
<td>pymic.transform.abstract_transform</td>
<td>84</td>
</tr>
<tr>
<td>pymic.transform.crop</td>
<td>85</td>
</tr>
<tr>
<td>pymic.transform.flip</td>
<td>86</td>
</tr>
<tr>
<td>pymic.transform.intensity</td>
<td>86</td>
</tr>
</tbody>
</table>
pymic.transform.label_convert, 87
pymic.transform.normalize, 88
pymic.transform.pad, 89
pymic.transform.rescale, 90
pymic.transform.rotate, 90
pymic.transform.threshold, 91
pymic.transform.trans_dict, 92
pymic.util, 97
pymic.util.evaluation_cls, 92
pymic.util.general, 94
pymic.util.image_process, 94
pymic.util.parse_config, 96
pymic.util.post_process, 96
pymic.util.preprocess, 96
pymic.util.ramps, 97
<table>
<thead>
<tr>
<th>Class/Method</th>
<th>Module/Class Information</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstractClassificationLoss</td>
<td><code>pymic.loss.cls.basic</code></td>
<td>27</td>
</tr>
<tr>
<td>AbstractSegLoss</td>
<td><code>pymic.loss.seg.abstract</code></td>
<td>30</td>
</tr>
<tr>
<td>AbstractTransform</td>
<td><code>pymic.transform.abstract_transform</code></td>
<td>84</td>
</tr>
<tr>
<td>accuracy()</td>
<td><code>pymic.util.evaluation_cls</code></td>
<td>92</td>
</tr>
<tr>
<td>add_val()</td>
<td><code>pymic.net_run_nll.nll_dast.Rank</code> method</td>
<td>75</td>
</tr>
<tr>
<td>ASPPBlock</td>
<td><code>pymic.net.net2d.cople_net</code></td>
<td>43</td>
</tr>
<tr>
<td>AttentionGateBlock</td>
<td><code>pymic.net.net2d.unet2d_attention</code></td>
<td>50</td>
</tr>
<tr>
<td>AttentionUNet2D</td>
<td><code>pymic.net.net2d.unet2d_attention</code></td>
<td>50</td>
</tr>
<tr>
<td>AuxiliaryDecoder</td>
<td><code>pymic.net.net2d.unet2d_cct</code></td>
<td>51</td>
</tr>
<tr>
<td>binary_evaluation()</td>
<td><code>pymic.util.evaluation_cls</code></td>
<td>92</td>
</tr>
<tr>
<td>BiNet</td>
<td><code>pymic.net_run_nll.nll_co_teaching</code></td>
<td>73</td>
</tr>
<tr>
<td>BiNet</td>
<td><code>pymic.net_run_ssl.ssl_cps</code></td>
<td>78</td>
</tr>
<tr>
<td>BuiltInNet</td>
<td><code>pymic.net.cls.torch_pretrained_net</code></td>
<td>41</td>
</tr>
<tr>
<td>CenterCrop</td>
<td><code>pymic.transform.crop</code></td>
<td>85</td>
</tr>
<tr>
<td>ChannelSELayer</td>
<td><code>pymic.net.net2d.scse2d</code></td>
<td>45</td>
</tr>
<tr>
<td>ChannelSELayer3D</td>
<td><code>pymic.net.net3d.scse3d</code></td>
<td>57</td>
</tr>
<tr>
<td>ChannelSpatialSELayer</td>
<td><code>pymic.net.net2d.scse2d</code></td>
<td>46</td>
</tr>
<tr>
<td>ChannelSpatialSELayer3D</td>
<td><code>pymic.net.net3d.scse3d</code></td>
<td>57</td>
</tr>
<tr>
<td>ChannelToSpace3D</td>
<td><code>pymic.layer.space2channel</code></td>
<td>27</td>
</tr>
<tr>
<td>ChannelWiseThreshold</td>
<td><code>pymic.transform.threshold</code></td>
<td>91</td>
</tr>
<tr>
<td>ChannelWiseThresholdWithNormalize</td>
<td><code>pymic.transform.threshold</code></td>
<td>91</td>
</tr>
<tr>
<td>ClassificationAgent</td>
<td><code>pymic.net_run.agent_cls</code></td>
<td>68</td>
</tr>
<tr>
<td>ClassificationDataset</td>
<td><code>pymic.io.nifti_dataset</code></td>
<td>22</td>
</tr>
<tr>
<td>CombinedLoss</td>
<td><code>pymic.loss.seg.combined</code></td>
<td>32</td>
</tr>
<tr>
<td>ConsistLoss</td>
<td><code>pymic.net_run_nll.nll_dast</code></td>
<td>74</td>
</tr>
<tr>
<td>ConvBlock</td>
<td><code>pymic.net.net2d.unet2d</code></td>
<td>47</td>
</tr>
<tr>
<td>ConvBlock3D</td>
<td><code>pymic.net.net3d.unet3d</code></td>
<td>61</td>
</tr>
<tr>
<td>ConvBlockND</td>
<td><code>pymic.net.net3d.unet2d_sce</code></td>
<td>59</td>
</tr>
<tr>
<td>ConvBNActBlock</td>
<td><code>pymic.net.net2d.cople_net</code></td>
<td>44</td>
</tr>
<tr>
<td>convert_label()</td>
<td><code>pymic.util.image_process</code></td>
<td>94</td>
</tr>
<tr>
<td>convert_tensor_type()</td>
<td><code>pymic.net_run.agent_abstract.NetRunAgent</code> method</td>
<td>66</td>
</tr>
<tr>
<td>ConvLayer</td>
<td><code>pymic.net.net2d.cople_net</code></td>
<td>44</td>
</tr>
<tr>
<td>ConvolutionLayer</td>
<td><code>pymic.layer.convolution</code></td>
<td>23</td>
</tr>
<tr>
<td>ConvSCSEBlock</td>
<td><code>pymic.net.net2d.unet2d_scse</code></td>
<td>53</td>
</tr>
<tr>
<td>ConvSCSEBlock3D</td>
<td><code>pymic.net.net3d.unet2d_sce</code></td>
<td>63</td>
</tr>
<tr>
<td>COPLENet</td>
<td><code>pymic.net.net2d.cople_net</code></td>
<td>43</td>
</tr>
<tr>
<td>create_dataset()</td>
<td><code>pymic.net_run.agent_abstract.NetRunAgent</code> method</td>
<td>66</td>
</tr>
<tr>
<td>create_dataset()</td>
<td><code>pymic.net_run_nll.nll_dast.NLLDAST</code> method</td>
<td>74</td>
</tr>
<tr>
<td>create_dataset()</td>
<td><code>pymic.net_run_ssl.ssl_abstract.SSLSegAgent</code> method</td>
<td>76</td>
</tr>
<tr>
<td>create_loss_calculator()</td>
<td><code>pymic.net_run.agent_abstract.NetRunAgent</code> method</td>
<td>66</td>
</tr>
<tr>
<td>create_loss_calculator()</td>
<td><code>pymic.net_run.agent_cls.ClassificationAgent</code> method</td>
<td>68</td>
</tr>
<tr>
<td>create_loss_calculator()</td>
<td><code>pymic.net_run.agent_seg.SegmentationAgent</code> method</td>
<td>70</td>
</tr>
<tr>
<td>create_network()</td>
<td><code>pymic.net_run.agent_abstract.NetRunAgent</code> method</td>
<td>66</td>
</tr>
<tr>
<td>create_network()</td>
<td><code>pymic.net_run.agent_cls.ClassificationAgent</code> method</td>
<td>68</td>
</tr>
<tr>
<td>create_network()</td>
<td><code>pymic.net_run.agent_seg.SegmentationAgent</code> method</td>
<td>70</td>
</tr>
</tbody>
</table>
PyMIC, Release 0.1

create_network() (pymic.net_run_nll.nll_co_teaching.NLLCoTeaching), 73
create_network() (pymic.net_run_nll.nll_trinet.NLLTriNet), 75
create_network() (pymic.net_run_ssl.ssl_cps.SSLCPS), 78
create_network() (pymic.net_run_ssl.ssl_mt.SSLMeanTeacher), 79
create_network() (pymic.net_run_wsl.wsl_ustm.WSLUSTM), 84
create_optimizer() (pymic.net_run_agent_abstract.NetRunAgent), 66
crop_and_pad_ND_array_to_desired_shape() (in module pymic.util.image_process), 94
crop_ND_volume_with_bounding_box() (in module pymic.util.image_process), 94
CropWithBoundingBox (class in pymic.transform.crop), 85

CrossEntropyLoss (class in pymic.loss.cls.basic), 28
CrossEntropyLoss (class in pymic.loss.seg.ce), 30
CSE (pymic.net.net2d.sce2d.SELayer attribute), 46
CSE (pymic.net.net3d.sce3d.SELayer attribute), 58
CSSE (pymic.net.net2d.sce2d.SELayer attribute), 47
CSSE (pymic.net.net3d.sce3d.SELayer attribute), 58

Decoder (class in pymic.net.net2d.unet2d), 47
DeconvolutionLayer (class in pymic.layer.deconvolution), 25

DeepSuperviseLoss (class in pymic.loss.seg.deep_sup), 32
DepthSeperableConvolutionLayer (class in pymic.layer.convolution), 24
DepthSeperableDeconvolutionLayer (class in pymic.layer.deconvolution), 26

DiceLoss (class in pymic.loss.seg.dice), 33
DownBlock (class in pymic.net.net2d.cople_net), 44
DownBlock (class in pymic.net.net2d.unet2d), 48
DownBlock (class in pymic.net.net2d.unet2d_scse), 54
DownBlock (class in pymic.net.net3d.unet2d_scse), 59
DownBlock (class in pymic.net.net3d.unet3d), 61
DownBlock (class in pymic.net.net3d.unet3d_scse), 63

Encoder (class in pymic.net.net2d.unet2d), 48

EntropyLoss (class in pymic.loss.seg.entropy), 48
ExpLogLoss (class in pymic.loss.seg.exp_log), 35

F

feature_based_noise() (pymic.net.net2d.unet2d_cct.AuxiliaryDecoder), 51

method), 70

FeatureDropout() (in module pymic.net.net2d.unet2d_urpc), 55
FocalDiceLoss (class in pymic.loss.seg.dice), 33
forward() (pymic.layer.convolution.ConvolutionLayer), 66
forward() (pymic.layer.convolution.DepthSeperableConvolutionLayer), 25
forward() (pymic.layer.convolution.DepthSeperableDeconvolutionLayer), 26
forward() (pymic.layer.space2channel.ChannelToSpace3D), 27
forward() (pymic.layer.space2channel.SpaceToChannel3D), 27
forward() (pymic.loss.cls.basic.AbstractClassificationLoss), 27
forward() (pymic.loss.cls.basic.CrossEntropyLoss), 28
forward() (pymic.loss.cls.basic.L1Loss), 28
forward() (pymic.loss.cls.basic.MSELoss), 28
forward() (pymic.loss.cls.basic.NLLLoss), 29
forward() (pymic.loss.cls.basic.SigmoidCELoss), 29
forward() (pymic.loss.seg.abstract.AbstractSegLoss), 30
forward() (pymic.loss.seg.ce.CrossEntropyLoss), 30
forward() (pymic.loss.seg.ce.GeneralizedCELoss), 31
forward() (pymic.loss.seg.combined.CombinedLoss), 32
forward() (pymic.loss.seg.deep_sup.DeepSuperviseLoss), 32
forward() (pymic.loss.seg.dice.DiceLoss), 33
forward() (pymic.loss.seg.dice.FocalDiceLoss), 33
forward() (pymic.loss.seg.dice.NoiseRobustDiceLoss), 34
forward() (pymic.loss.seg.exp_log.ExpLogLoss), 35
forward() (pymic.loss.seg.gatedcrf.GatedCRFLoss), 35
forward() (pymic.loss.seg.mse.MAEloss), 36
forward() (pymic.loss.seg.mse.NLLLoss), 37
forward() (pymic.loss.seg.mumford_shah.MumfordShahLoss), 37
forward() (pymic.loss.seg.slsr.SLSRLoss method), 38
forward() (pymic.loss.seg.ssl.EntropyLoss method), 39
forward() (pymic.loss.seg.ssl.TotalVariationLoss method), 39
forward() (pymic.net.cls.torch_pretrained_net.BuiltInNet method), 41
forward() (pymic.net2d.cople_net.ASPPBlock method), 43
forward() (pymic.net2d.cople_net.ConvBNActBlock method), 44
forward() (pymic.net2d.UpBlock method), 45
forward() (pymic.net2d.ConvBlock method), 46
forward() (pymic.net2d.ConvLayer method), 47
forward() (pymic.net2d.DownBlock method), 48
forward() (pymic.net2d.Encoder method), 49
forward() (pymic.net2d.UNet2D method), 50
forward() (pymic.net2d.UpBlock method), 51
forward() (pymic.net2d_down_attention.AttentionGateBlock method), 52
forward() (pymic.net2d_dual_branch.UNet2D_ScSE method), 55
forward() (pymic.net2d_unet2d_sce.UpBlock method), 55
forward() (pymic.net2d_unet2d.urpc.FeatureNoise method), 55
forward() (pymic.net2d_unet2d.urpc.UNet2D_URPC method), 56
forward() (pymic.net3d.scse3d.ChannelSELayer3D method), 57
forward() (pymic.net3d.scse3d.ChannelSpatialSELayer3D method), 57
forward() (pymic.net3d.unet2d.SpatialSELayer3D method), 58
forward() (pymic.net3d.unet2d5.ConvBlockND method), 59
forward() (pymic.net3d.unet2d5.DownBlock method), 59
forward() (pymic.net3d.unet2d5.UNet2D5 method), 60
forward() (pymic.net3d.unet2d5.UpBlock method), 60
forward() (pymic.net3d.unet3d.ConvBlock method), 61
forward() (pymic.net3d.unet3d.ConvBlock3D method), 61
forward() (pymic.net3d.unet3d.DownBlock method), 62
forward() (pymic.net3d.unet3d.DownBlock method), 63
forward() (pymic.net3d.unet3d_UPscEnc.Encoder method), 63
forward() (pymic.net3d.unet3d_UPscEnc.Encoder method), 64
forward() (pymic.net3d.unet3d_UPscEnc.UpBlock method), 65
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 73
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 74
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 75
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 76
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 77
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 78
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 79
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 80
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 81
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 82
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 83
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 84
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 85
forward() (pymic.net_run_nll.nll_co_teaching.BiNet method), 86
GatedCRFLoss (class in pymic.loss.seg.gatedcrf), 35
GaussianNoise (class in pymic.transform.intensity), 87
GeneralizedCELoss (class in pymic.loss.seg.ce), 31
get_acti_func() (in module pymic.layer.activation), 23
get_ce() (in module pymic.net_run_nll.nll_dast), 75
pymic.net_run_ssl, 81
pymic.net_run_ssl.ssl_abstract, 76
pymic.net_run_ssl.ssl_cct, 77
pymic.net_run_ssl.ssl_cps, 78
pymic.net_run_ssl.ssl_em, 79
pymic.net_run_ssl.ssl_main, 79
pymic.net_run_ssl.ssl_quant, 80
pymic.net_run_ssl.ssl_urpc, 80
pymic.net_run_wsl, 84
pymic.net_run_wsl.wsl_abstract, 81
pymic.net_run_wsl.wsl_dmpls, 81
pymic.net_run_wsl.wsl_dmpls, 81
pymic.net_run_wsl.wsl_em, 82
pymic.net_run_wsl.wsl_gatedcrf, 82
pymic.net_run_wsl.wsl_mumford_shah, 83
pymic.net_run_wsl.wsl_main, 83
pymic.net_run_wsl.wsl_tv, 83
pymic.net_run_wsl.wsl_ustm, 84
pymic.transform, 92
pymic.transform.abstract_transform, 84
pymic.transform.crop, 85
pymic.transform.flip, 86
pymic.transform.intensity, 86
pymic.transform.label_convert, 87
pymic.transform.normalize, 88
pymic.transform.pad, 89
pymic.transform.rescale, 90
pymic.transform.rotate, 90
pymic.transform.threshold, 91
pymic.transform.trans_dict, 92
pymic.util, 97
pymic.util.evaluation_cls, 92
pymic.util.general, 94
pymic.util.image_process, 94
pymic.util.parse_config, 96
pymic.util.post_process, 96
pymic.util.preprocess, 96
pymic.util.ramps, 97
MSELoss (class in pymic.loss.cls.basic), 28
MSELoss (class in pymic.loss.seg.mse), 36
MumfordShahLoss (class in pymic.loss.seg.mumford_shah), 37

N
NestedUNet2D (class in pymic.net.net2d.unet2d_nest), 53
NetRunAgent (class in pymic.net_run.agent_abstract), 66
nexcl_evaluation() (in module pymic.util.evaluation_cls), 93
NiftyDataset (class in pymic.io.nifty_dataset), 23
NLLCLSLSR (class in pymic.net_run_nll.nll_clslsr), 72
NLLCoTeaching (class in pymic.net_run_nll.nll_co_teaching), 73
NLLDAST (class in pymic.net_run_nll.nll_dast), 74
NLLLoss (class in pymic.loss.cls.basic), 29
NLLTriNet (class in pymic.net_run_nll.nll_trinet), 75
NoiseRobustDiceLoss (class in pymic.loss.seg.dice), 34
NONE (pymic.net2d.scse2d.SELayer attribute), 47
NONE (pymic.net3d.scse3d.SELayer attribute), 58
NormalizeWithMeanStd (class in pymic.transform.normalize), 88
NormalizeWithMinMax (class in pymic.transform.normalize), 88
NormalizeWithPercentiles (class in pymic.transform.normalize), 89

P
Pad (class in pymic.transform.pad), 89
parse_bool() (in module pymic.util.parse_config), 96
parse_config() (in module pymic.util.parse_config), 96
parse_list() (in module pymic.util.parse_config), 96
parse_value_from_string() (in module pymic.util.parse_config), 96
PartialLabelToProbability (class in pymic.transform.label_convert), 88
PostKeepLargestComponent (class in pymic.util.post_process), 96
PostProcess (class in pymic.util.post_process), 96
preprocess_with_transform() (in module pymic.util.preprocess), 96
pymic.io module, 23
pymic.io.image_read_write module, 21
pymic.io.nifty_dataset module, 22
pymic.layer module, 27
pymic.layer.activation module, 23
pymic.layer.convolution module, 23
pymic.layer.deconvolution module, 25
pymic.layer.space2channel module, 27
pymic.loss module, 41
pymic.loss.cls module, 30
pymic.loss.cls.basic module, 27
pymic.loss.cls.util module, 29
pymic.loss.loss_dict_cls module, 29

Index
module, 40
pymic.loss.loss_dict_seg
module, 41
pymic.loss.seg
module, 40
pymic.loss.seg.abstract
module, 30
pymic.loss.seg.ce
module, 30
pymic.loss.seg.combined
module, 32
pymic.loss.seg.deep_sup
module, 32
pymic.loss.seg.dice
module, 33
pymic.loss.seg.exp_log
module, 35
pymic.loss.seg.gatedcrf
module, 35
pymic.loss.seg.mse
module, 36
pymic.loss.seg.mumford_shah
module, 37
pymic.loss.seg.slsr
module, 38
pymic.loss.seg.ssl
module, 39
pymic.loss.seg.util
module, 39
pymic.net
module, 66
pymic.net.cls
module, 43
pymic.net.cls.torch_pretrained_net
module, 41
pymic.net.net2d
module, 57
pymic.net.net2d.cople_net
module, 43
pymic.net.net2d.scse2d
module, 45
pymic.net.net2d.unet2d
module, 47
pymic.net.net2d.unet2d_attention
module, 50
pymic.net.net2d.unet2d_cct
module, 51
pymic.net.net2d.unet2d_dual_branch
module, 52
pymic.net.net2d.unet2d_nested
module, 53
pymic.net.net2d.unet2d_scse
module, 53
pymic.net.net2d.unet2d_urpc
module, 55
pymic.net.net3d
module, 65
pymic.net.net3d.scse3d
module, 57
pymic.net.net3d.unet2d5
module, 59
pymic.net.net3d.unet3d
module, 61
pymic.net.net3d.unet3d_scse
module, 63
pymic.net.net_dict_cls
module, 65
pymic.net.net_dict_seg
module, 65
pymic.net.run
module, 72
pymic.net.run.agent_abstract
module, 66
pymic.net.run.agent_cls
module, 68
pymic.net.run.agent_seg
module, 70
pymic.net.run.get_optimizer
module, 70
pymic.net.run.infer_func
module, 71
pymic.net.run.net_run
module, 72
pymic.net.run_nll
module, 76
pymic.net.run_nll.nll_clslsr
module, 72
pymic.net.run_nll.nll_co_teaching
module, 73
pymic.net.run_nll.nll_dast
module, 74
pymic.net.run_nll.nll_main
module, 75
pymic.net.run_nll.nll_trinet
module, 75
pymic.net.run_ssl
module, 81
pymic.net.run_ssl.ssl_abstract
module, 76
pymic.net.run_ssl.ssl_cct
module, 77
pymic.net.run_ssl.ssl_cps
module, 78
pymic.net.run_ssl.ssl_em
module, 79
pymic.net.run_ssl.ssl_main
module, 79
pymic.net.run_ssl.ssl_mt
module, 79
module, 79
pymic.net_run_ssl.ssl_uamt module, 80
pymic.net_run_ssl.ssl_urpc module, 80
pymic.net_run_wsl module, 84
pymic.net_run_wsl.wsl_abstract module, 81
pymic.net_run_wsl.wsl_dmpls module, 81
pymic.net_run_wsl.wsl_em module, 82
pymic.net_run_wsl.wsl_gatedcrf module, 82
pymic.net_run_wsl.wsl_main module, 83
pymic.net_run_wsl.wsl_mumford_shah module, 83
pymic.net_run_wsl.wsl_tv module, 83
pymic.net_run_wsl.wsl_ustm module, 84
pymic.transform module, 92
pymic.transform.abstract_transform module, 84
pymic.transform.crop module, 85
pymic.transform.flip module, 86
pymic.transform.intensity module, 86
pymic.transform.label_convert module, 87
pymic.transform.normalize module, 88
pymic.transform.pad module, 89
pymic.transform.rescale module, 90
pymic.transform.rotate module, 90
pymic.transform.threshold module, 91
pymic.transform.trans_dict module, 92
pymic.util module, 97
pymic.util.parse_config module, 96
pymic.util.post_process module, 96
pymic.util.preprocess module, 96
pymic.util.ramps module, 97
R
RandomCrop (class in pymic.transform.crop), 85
RandomFlip (class in pymic.transform.flip), 86
RandomRescale (class in pymic.transform.rescale), 90
RandomResizedCrop (class in pymic.transform.crop), 85
RandomRotate (class in pymic.transform.rotate), 90
Rank (class in pymic.net_run_nll.nll_dast), 74
ReduceLabelDim (class in pymic.transform.label_convert), 88
resample_sitk_image_to_given_spacing() (in module pymic.util.image_process), 95
Rescale (class in pymic.transform.rescale), 90
reshape_prediction_and_ground_truth() (in module pymic.loss.seg.util), 40
reshape_tensor_to_2D() (in module pymic.loss.seg.util), 40
ResNet18 (class in pymic.net.cls.torch_pretrained_net), 42
rotate_nifty_volume_to_LPS() (in module pymic.io.image_read_write), 22
run() (pymic.net_run.agent_abstract.NetRunAgent method), 67
run() (pymic.net_run.infer_func.Inferer method), 71
S
save_array_as_nifty_volume() (in module pymic.io.image_read_write), 22
save_array_as_rgb_image() (in module pymic.io.image_read_write), 22
save_nd_array_as_image() (in module pymic.io.image_read_write), 22
save_outputs() (pymic.net_run.agent_seg.SegmentationAgent method), 70
SEBlock (class in pymic.net.net2d.cople_net), 45
seed_torch() (in module pymic.net_run.agent_abstract), 68
SegmentationAgent (class in pymic.net_run.agent_abstract), 68
SELayer (class in pymic.net.net2d.scse2d), 46
SELayer (class in pymic.net.net3d.scse3d), 58
select_criterion() (in module pymic.net_run_nll.nll_dast), 75
sensitivity() (in module pymic.util.evaluation_cls), 93
set_datasets() (pymic.net_run.agent_abstract.NetRunAgent method), 67
set_inferer() (pymic.net_run.agent_abstract.NetRunAgent method), 67
set_loss_dict() (pymic.net_run.agent_abstract.NetRunAgent method), 67
set_network() (pymic.net_run.agent_abstract.NetRunAgent method), 67
set_ND_volume_roi_with_bounding_box_range() (in module pymic.util.image_process), 95
set_net_dict() (pymic.net_run.agent_abstract.NetRunAgent method), 67
set_network() (pymic.net_run.agent_abstract.NetRunAgent method), 67
set_optimizer() (pymic.net_run.agent_abstract.NetRunAgent method), 67
set_postprocessor() (pymic.net_run.agent_segmentation.NetRunAgent method), 70
set_scheduler() (pymic.net_run.agent_abstract.NetRunAgent method), 67
set_transform_dict() (pymic.net_run.agent_abstract.NetRunAgent method), 68

SigmoidCELoss (class in pymic.loss.cls.basic), 29
SSLRLoss (class in pymic.loss.seg.slsr), 38
softmax_js_loss() (in module pymic.net_run.ssl_ssl_cct), 77
softmax_kl_loss() (in module pymic.net_run.ssl_ssl_cct), 77
softmax_mse_loss() (in module pymic.net_run.ssl_ssl_cct), 77
SpaceToChannel3D (class in pymic.layer.space2channel), 27
SpatialSELayer (class in pymic.net.net2d.scse2d), 47
SpatialSELayer3D (class in pymic.net.net3d.scse3d), 58
specificity() (in module pymic.util.evaluation_cls), 93
SSE (pymic.net.net2d.scse2d.SELayer attribute), 47
SSE (pymic.net.net3d.scse3d.SELayer attribute), 58
SSLCC (class in pymic.net_run.ssl_ssl_cct), 77
SSLCPS (class in pymic.net_run.ssl_ssl_cps), 78
SSLEntropyMinimization (class in pymic.net_run.ssl_ssl_em), 79
SSLMeanTeacher (class in pymic.net_run.ssl_sslMt), 79
SSLSegAgent (class in pymic.net_run.ssl_ssl_abstract), 76
SSLUncertaintyAwareMeanTeacher (class in pymic.net_run.ssl_ssl_uamt), 80
SSLURPC (class in pymic.net_run.ssl_ssl urpc), 80
synchronize_config() (in module pymic.util.parse_config), 96

TotalVariationLoss (class in pymic.loss.seg.ssl), 39

Index

T
training (pymic.loss.seg.ssl.SLSRLoss attribute), 38
training (pymic.loss.seg.ssl.EntropyLoss attribute), 39
training (pymic.loss.seg.ssl.TotalVariationLoss attribute), 39
training (pymic.net.cls.torch_pretrained_net.BuiltInNet attribute), 42
training (pymic.net.cls.torch_pretrained_net.MobileNetV2 attribute), 42
training (pymic.net.cls.torch_pretrained_net.ResNet18 attribute), 42
training (pymic.net.cls.torch_pretrained_net.VGG16 attribute), 43
training (pymic.net.net2d.cople_net.ASPPBlock attribute), 43
training (pymic.net.net2d.cople_net.ConvBNActBlock attribute), 44
training (pymic.net.net2d.cople_net.ConvLayer attribute), 44
training (pymic.net.net2d.cople_net.COPLENet attribute), 44
training (pymic.net.net2d.cople_net.DownBlock attribute), 45
training (pymic.net.net2d.cople_net.SEBlock attribute), 45
training (pymic.net.net2d.cople_net.UpBlock attribute), 45
training (pymic.net.net3d.unet3d_scse.DownBlock attribute), 45
training (pymic.net.net3d.unet3d_scse.UpBlock attribute), 46
training (pymic.net.net3d.unet3d.DownBlock attribute), 46
training (pymic.net.net3d.unet3d.UpBlock attribute), 46
training (pymic.net.net3d.scse.DownBlock attribute), 47
training (pymic.net.net3d.scse.UpBlock attribute), 47
training (pymic.net.net3d.unet2d.ConvBlock attribute), 48
training (pymic.net.net3d.unet2d.UpBlock attribute), 48
training (pymic.net.net3d.unet2d.Encoder attribute), 49
training (pymic.net.net3d.unet2d.UNet2D attribute), 50
training (pymic.net.net3d.unet2d.UpBlock attribute), 50
training (pymic.net.net3d.unet2d.unet2d_attention.AttentionGateBlock attribute), 50
training (pymic.net.net3d.unet2d.unet2d_attention.AttentionUNet2D attribute), 50
training (pymic.net.net3d.unet2d.unet2d_attention.UpBlockWithAttention attribute), 51
training (pymic.net.net3d.unet2d.cct.AuxiliaryDecoder attribute), 51
training (pymic.net.net3d.unet2d.cct.UNet2D_CCT attribute), 52
training (pymic.net.net2d.unet2d_dual_branch.UNet2D_DualBranch attribute), 53
training (pymic.net.net2d.unet2d_nest.NestedUNet2D attribute), 53
training (pymic.net.net2d.unet2d_scse.ConvScSEBlock attribute), 54
training (pymic.net.net2d.unet2d_scse.DownBlock attribute), 54
training (pymic.net.net2d.unet2d_scse.UpBlock attribute), 55
training (pymic.net.net2d.unet2d_scse.UNet2D_ScSE attribute), 55
training (pymic.net.net2d.unet2d_scse.UNet2D_SCSE attribute), 55
training (pymic.net.net2d.unet2d_scse. FeatureNoise attribute), 56
training (pymic.net.net2d.unet2d_scse.UNet2D URPC attribute), 56
training (pymic.net.net3d.unet3d.neunet3d.ChannelSELayer3D attribute), 57
training (pymic.net.net3d.unet3d.ChannelSpatialSELayer3D attribute), 58
training (pymic.net.net3d.unet3d.DownBlock attribute), 59
training (pymic.net.net3d.unet2d5.ConvBlockND attribute), 59
training (pymic.net.net3d.unet2d5.DownBlock attribute), 60
training (pymic.net.net3d.unet2d5.UNet2D attribute), 60
training (pymic.net.net3d.unet2d5.UpBlock attribute), 61
training (pymic.net.net3d.unet3d.ConvBlock attribute), 61
training (pymic.net.net3d.unet3d.DownBlock attribute), 62
training (pymic.net.net3d.unet3d. UNet3D attribute), 62
training (pymic.net.net3d.unet3d.UpBlock attribute), 63
training (pymic.net.net3d.unet3d.scse.ConvScSEBlock3D attribute), 63
training (pymic.net.net3d.unet3d.scse.DownBlock attribute), 63
training (pymic.net.net3d.unet3d.scse. DownBlock attribute), 64
training (pymic.net.net3d.unet3d.scse.UNet3D ScSE attribute), 64
training (pymic.net.net3d.unet3d.scse.UpBlock attribute), 65
training (pymic.net_run_nll.nll_co_teaching.BiNet attribute), 73
training (pymic.net_run_nll.nll_co_teaching.BiNet attribute), 73
training (pymic.net_run_nll.nll_co_teaching.BiNet attribute), 73
training (pymic.net_run_nll.nll_co_teaching.BiNet attribute), 73
training (pymic.net_run_nll.nll_dast.ConsistLoss attribute), 74
training (pymic.net_run_nll.nll_dast.ConsistLoss attribute), 74
training (pymic.net_run_nll.nll_dast.ConsistLoss attribute), 74
training (pymic.net_run_nll.nll_dast.ConsistLoss attribute), 74
training (pymic.net_run_nll.nll_trinet.TriNet attribute), 76
training (pymic.net_run_ssl.ssl_cps.BiNet attribute), 78
training () (pymic.net_run.agent_abstract.NetRunAgent method), 68
training () (pymic.net_run.agent_abstract.NetRunAgent method), 69
training() (pymic.net_run.agent_seg.SegmentationAgent
method), 70
training() (pymic.net_run_nll.nll_co_teaching.NLLCoTeaching
method), 73
training() (pymic.net_run_nll.nll_dast.NLLDAST
method), 74
training() (pymic.net_run_nll.nll_trinet.NLLTriNet
method), 75
training() (pymic.net_run_ssl.ssl_cct.SSLCCT
method), 77
training() (pymic.net_run_ssl.ssl_cps.SSLCPS
method), 78
training() (pymic.net_run_ssl.ssl_em.SSLEntropyMinimization
method), 79
training() (pymic.net_run_ssl.ssl_uamt. SSLUncertaintyAwareMeanTeacher
method), 80
training() (pymic.net_run_ssl.ssl_urpc.SSLURPC
method), 81
training() (pymic.net_run_wsl.wsl_dmplps.WSLDMPLS
method), 82
training() (pymic.net_run_wsl.wsl_em.WSLEntropyMinimization
method), 82
training() (pymic.net_run_wsl.wsl_gatedcrf.WSLGatedCRF
method), 83
training() (pymic.net_run_wsl.wsl_mumford_shah.WSLMumfordShah
method), 83
training() (pymic.net_run_wsl.wsl_tv.WSLTotalVariation
method), 83
training() (pymic.net_run_wsl.wsl_ustm.WSLUSTM
method), 84
TriNet (class in pymic.net_run_nll.nll_trinet), 76

V
validation() (pymic.net_run.agent_abstract.NetRunAgent
method), 68
validation() (pymic.net_run.agent_cls.ClassificationAgent
method), 69
validation() (pymic.net_run.agent_seg.SegmentationAgent
method), 70
VGG16 (class in pymic.cls.torch_pretrained_net), 42

W
write_scalars() (pymic.net_run.agent_abstract.NetRunAgent
method), 68
write_scalars() (pymic.net_run.agent_cls.ClassificationAgent
method), 69
write_scalars() (pymic.net_run.agent_seg.SegmentationAgent
method), 71
write_scalars() (pymic.net_run_nll.nll_co_teaching.NLLCoTeaching
method), 76
write_scalars() (pymic.net_run_nll.nll_trinet.NLLTriNet
method), 76
write_scalars() (pymic.net_run_ssl.ssl_abstract.SSLSegAgent
method), 77
write_scalars() (pymic.net_run_ssl.ssl_cps.SSLCPS
method), 78
write_scalars() (pymic.net_run_ssl.ssl_em.WSLEntropyMinimization
method), 78
write_scalars() (pymic.net_run_ssl.ssl_gatedcrf.WSLGatedCRF
method), 78
write_scalars() (pymic.net_run_ssl.ssl_mumford_shah.WSLMumfordShah
method), 81
WSLDMPLS (class in pymic.net_run_wsl.wsl_dmplps), 81
WSLEntropyMinimization (class in
pymic.net_run_wsl.wsl_em), 82
WSLGatedCRF (class in
pymic.net_run_wsl.wsl_gatedcrf), 82
WSLMumfordShah (class in
pymic.net_run_wsl.wsl_mumford_shah), 83
WSLSegAgent (class in pymic.net_run_wsl.wsl_abstract), 81
WSLTotalVariation (class in
pymic.net_run_wsl.wsl_tv), 83
WSLUSTM (class in pymic.net_run_wsl.wsl_ustm), 84

U
UNet2D (class in pymic.net.net2d.unet2d), 49
UNet2D5 (class in pymic.net.net3d.unet2d5), 59
UNet2D_CCT (class in pymic.net.net2d.unet2d_cct), 51
UNet2D_DualBranch (class in
pymic.net.net2d.unet2d_dual_branch), 52
UNet2D_ScSE (class in pymic.net.net2d.unet2d_scse), 54
UNet2D_URPC (class in pymic.net.net2d.unet2d_urpc), 56
UNet3D (class in pymic.net.net3d.unet3d), 62
UNet3D_ScSE (class in pymic.net.net3d.unet3d_scse), 64
UpBlock (class in pymic.net.net2d.cople_net), 45
UpBlock (class in pymic.net.net2d.unet2d), 50
UpBlock (class in pymic.net.net2d.unet2d_scse), 55
UpBlock (class in pymic.net.net3d.unet2d5), 60
UpBlock (class in pymic.net.net3d.unet3d), 62
UpBlock (class in pymic.net.net3d.unet3d_scse), 64
UpBlockWithAttention (class in
pymic.net.net2d.unet2d_attention), 51