ivadomed

Nov 10, 2020

Overview

Note: This website is under construction.

`ivadomed` is an integrated framework for medical image analysis with deep learning. The name is a portmanteau between *IVADO* (The Institute for data valorization) and *Medical*.

The purpose of the `ivadomed` project is to:

- Provide researchers with an open-source framework for training deep learning models for applications in medical imaging;
- Provide ready-to-use *Models* trained on multi-center data.

CHAPTER 1

Comparison with other projects

We acknowledge the existence of similar projects. The specificity of `ivadomed` is highlighted in the table below:

Note: `TODO: Add comparative table`

CHAPTER 2

Technical features

2.1 Physics-informed network

CNNs can be modulated, at each layer, using the Feature-wise Linear Modulation (FiLM) technique. FiLM permits to add priors during training/inference based on the imaging physics (e.g. acquisition parameters), thereby improving the performance of the output segmentations.

2.2 Uncertainty measures

At inference time, uncertainty can be estimated via two ways: - model-based uncertainty (epistemic) based on Monte Carlo Dropout. - image-based uncertainty (aleatoric) based on test-time augmentation.

From the Monte Carlo samples, different measures of uncertainty can be derived: - voxel-wise entropy - structure-wise intersection over union - structure-wise coefficient of variation - structure-wise averaged voxel-wise uncertainty within the structure

These measures can be used to perform some post-processing based on the uncertainty measures.

2.3 Two-step training scheme with class sampling

Class sampling, coupled with a transfer learning strategy, can mitigate class imbalance issues, while addressing the limitations of classical under-sampling (risk of loss of information) or over-sampling (risk of overfitting) approaches.

During a first training step, the CNN is trained on an equivalent proportion of positive and negative samples, negative samples being under-weighted dynamically at each epoch. During the second step, the CNN is fine-tuned on the realistic (i.e. class-imbalanced) dataset.

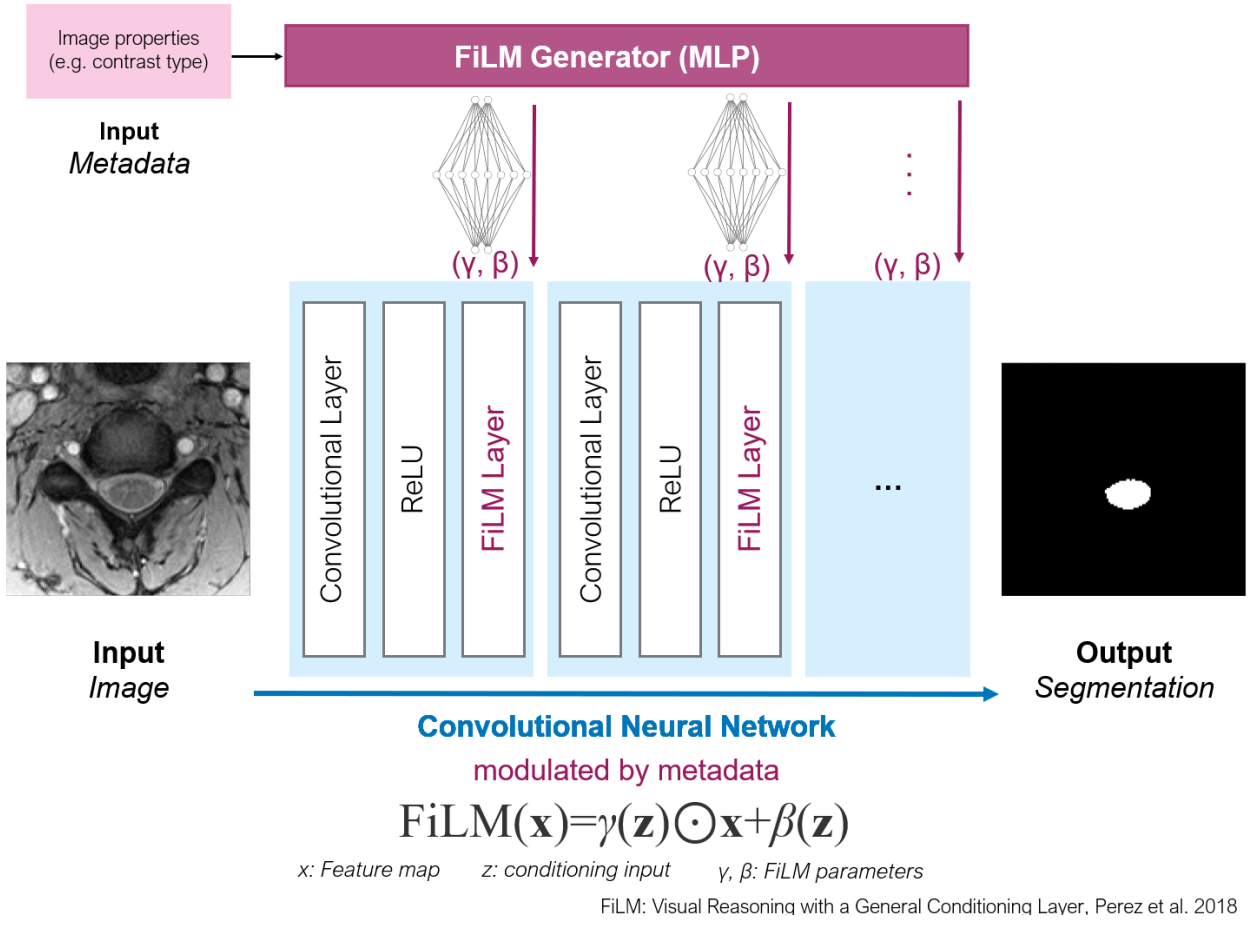


Fig. 1: Figure FiLM

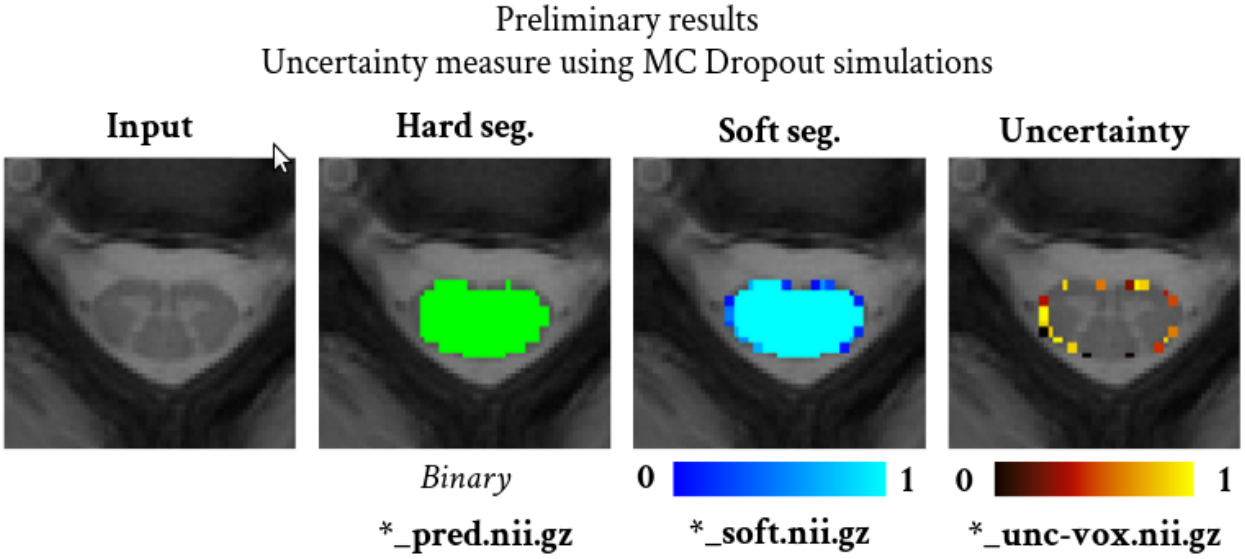


Fig. 2: Figure Uncertainty

2.4 Mixup

Mixup is a data augmentation technique, wherein training is performed on samples that are generated by combining two random samples from the training set and from the associated labels. The motivation is to regularize the network while extending the training distribution.

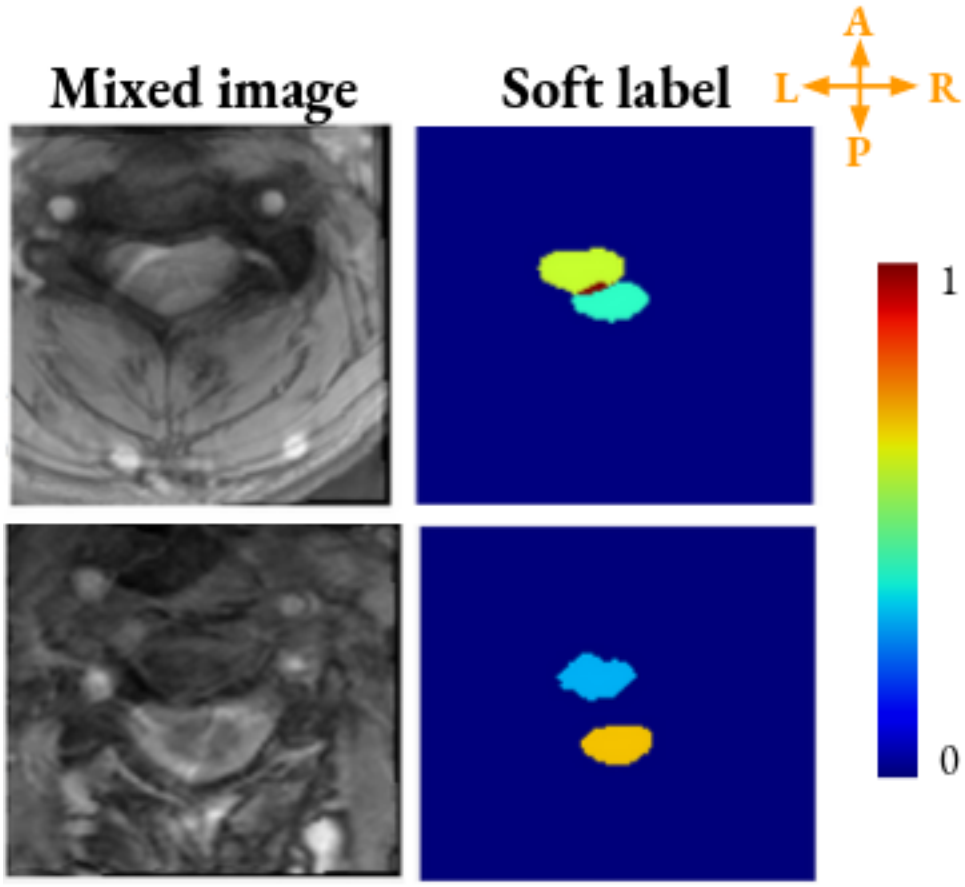


Fig. 3: Figure mixup

2.5 Data augmentation on lesion labels

This data augmentation is motivated by the large inter-rater variability that is common in medical image segmentation tasks. Typically, raters disagree on the boundaries of pathologies (e.g., tumors, lesions). A soft mask is constructed by morphological dilation of the binary segmentation (i.e. mask provided by expert), where expert-labeled voxels have one as value while the augmented voxels are assigned a soft value which depends on the distance to the core of the lesion. Thus, the prior knowledge about the subjective lesion borders is then leveraged to the network.

2.6 Network architectures

- UNet, with control of the network depth.
- HeMIS-UNet: integrates the HeMIS strategy to deal with missing modalities within a UNet training scheme.

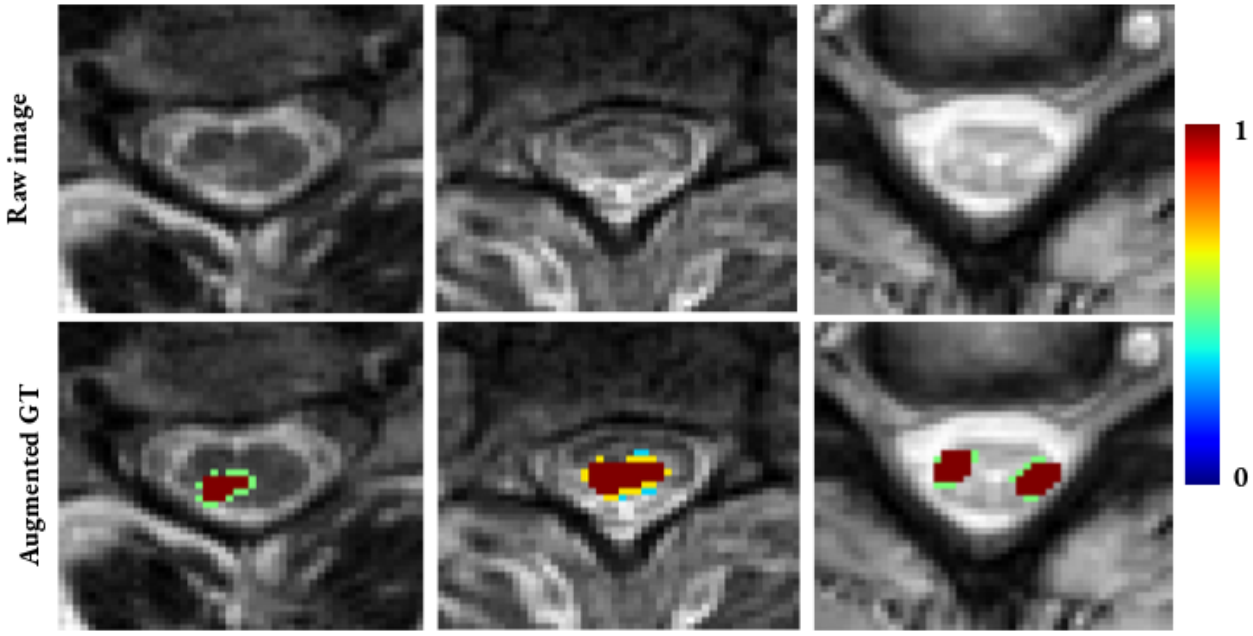


Fig. 4: Figure Data Augmentation on lesion ground truths

- FiLMed-UNet, based on FiLM strategy adapted to the *segmentation task*.
- Countception: modified implementation of Countception for keypoints detection.

2.7 Loss functions

- Dice Loss. Also adapted for multi-label segmentation tasks, by averaging the loss for each class.
- Focal Loss.
- Focal-Dice Loss: Linear combination of the Focal and Dice losses.
- Generalized Dice Loss. An additional feature compared to the published reference, is that the background volume can be weighted by the inverse of its area, which could be of interest in high class imbalance scenarios.
- Adaptive wing loss. Loss function used to detect key points with Gaussian representation of the target.
- Loss Combination: Linear combination of any other implemented losses.

CHAPTER 3

Installation

`ivadomed` requires Python >= 3.6 and PyTorch >= 1.5.0. We recommend working under a virtual environment, which could be set as follows:

```
virtualenv venv-ivadomed --python=python3.6
source venv-ivadomed/bin/activate
```

3.1 Install from release (recommended)

Install `ivadomed` and its requirements from Pypi:

```
pip install --upgrade pip
pip install ivadomed
```

3.2 Install from source

Bleeding-edge developments are available on the project's master branch on Github. Installation procedure is the following:

```
git clone https://github.com/neuropoly/ivadomed.git
cd ivadomed
pip install -e .
```

CHAPTER 4

Getting started

New model can be generated using the command-line tool from the terminal:

```
ivadomed ivadomed/config/config.json
```

where `config.json` is a configuration file, which parameters are described in the *Configuration File*.

To fully benefit from all the features of `ivadomed`, please see the tutorials:Tutorials.

CHAPTER 5

Configuration File

5.1 General parameters

5.1.1 command

Run the specified command. Choices: `"train"`, `"test"`, `"eval"`, to train, test and evaluate a model respectively.

5.1.2 gpu

Integer. ID of the GPU to use.

5.1.3 log_directory

Folder name that will contain the output files (e.g., trained model, predictions, results).

5.1.4 debugging

Bool. Extended verbosity and intermediate outputs.

5.2 Loader parameters

5.2.1 bids_path

String. Path of the BIDS folder.

5.2.2 target_suffix

List. Suffix list of the derivative file containing the ground-truth of interest (e.g. `["_seg-manual", "_lesion-manual"]`). The length of this list controls the number of output channels of the model (i.e. `out_channel`). If the list has a length greater than 1, then a multi-class model will be trained.

5.2.3 contrasts

- `train_validation`: List. List of image contrasts (e.g. `T1w`, `T2w`) loaded for the training and validation. If `multichannel` is `true`, this list represents the different channels of the input tensors (i.e. its length equals model's `in_channel`). Otherwise, the contrasts are mixed and the model has only one input channel (i.e. model's `in_channel=1`).
- `test`: List. List of image contrasts (e.g. `T1w`, `T2w`) loaded in the testing dataset. Same comment than for `train_validation` regarding `multichannel`.
- `balance`: Dict. Enables to weight the importance of specific channels (or contrasts) in the dataset: e.g. `{"T1w": 0.1}` means that only 10% of the available `T1w` images will be included into the training/validation/test set. Please set `multichannel` to `false` if you are using this parameter.

5.2.4 multichannel

Bool. Indicated if more than a contrast (e.g. `T1w` and `T2w`) is used by the model. See details in both `train_validation` and `test` for the contrasts that are input.

5.2.5 slice_axis

Choice between `"sagittal"`, `"coronal"`, and `"axial"`. Sets the slice orientation for on which the model will be used.

5.2.6 slice_filter

Dict. Discard a slice from the dataset if it meets a condition, see below. - `filter_empty_input`: Bool. Discard slices where all voxel intensities are zeros. - `filter_empty_mask`: Bool. Discard slices where all voxel labels are zeros.

5.2.7 roi

Dict. of parameters about the region of interest - `suffix`: String. Suffix of the derivative file containing the ROI used to crop (e.g. `"_seg-manual"`) with `ROICrop` as transform. Please use `null` if you do not want to use an ROI to crop. - `slice_filter_roi`: int. If the ROI mask contains less than `slice_filter_roi` non-zero voxels, the slice will be discarded from the dataset. This feature helps with noisy labels, e.g., if a slice contains only 2-3 labeled voxels, we do not want to use these labels to crop the image. This parameter is only considered when using `"ROICrop"`.

5.2.8 soft_gt

Bool. Indicated if usage of soft mask. Ground truth will be non-binarized images with float32 encoding.

5.3 Split dataset

5.3.1 fname_split

String. File name of the log (joblib) that contains the list of training/validation/testing subjects. This file can later be used to re-train a model using the same data splitting scheme. If `null`, a new splitting scheme is performed.

5.3.2 random_seed

Int. Seed used by the random number generator to split the dataset between training/validation/testing. The use of the same seed ensures the same split between the sub-datasets, which is useful to reproduce results.

5.3.3 method

`{"per_patient", "per_center"}`. `"per_patient"`: all subjects are shuffled, then split between train/validation/test according to `"train_fraction"` and `"test_fraction"`, regardless their institution. `"per_center"`: all subjects are split so as not to mix institutions between the train/validation/test sets according to `"train_fraction"` and `"center_test"`. The latter option enables to ensure the model is working across domains (institutions). Note: the institution information is contained within the `institution_id` column in the `participants.tsv` file.

5.3.4 train_fraction

Float. Between `0` and `1` representing the fraction of the dataset used as training set.

5.3.5 test_fraction

Float. Between `0` and `1` representing the fraction of the dataset used as test set. This parameter is only used if the `method` is `"per_patient"`.

5.3.6 center_test

List of strings. Each string corresponds to an institution/center to only include in the testing dataset (not validation). This parameter is only used if the `method` is `"per_center"`. If used, the file `bids_dataset/participants.tsv` needs to contain a column `institution_id`, which associates a subject with an institution/center.

5.4 Training parameters

5.4.1 batch_size

Strictly positive integer.

5.4.2 loss

- `name`: Name of the loss function class. See *`ivadomed.losses`*
- Other parameters that could be needed in the Loss function definition: see attributes of the Loss function of interest (e.g. `"gamma": 0.5` for `FocalLoss`).

5.4.3 training_time

- `num_epochs`: Strictly positive integer.
- `early_stopping_epsilon`: Float. If the validation loss difference during one epoch (i.e. `abs(validation_loss[n] - validation_loss[n-1]` where n is the current epoch) is inferior to this epsilon for `early_stopping_patience` consecutive epochs, then training stops.
- `early_stopping_patience`: Strictly positive integer. Number of epochs after which the training is stopped if the validation loss improvement is smaller than `early_stopping_epsilon`.

5.4.4 scheduler

- `initial_lr`: Float. Initial learning rate.
- `scheduler_lr`:

1. `name`: Choice between: `"CosineAnnealingLR"`, `"CosineAnnealingWarmRestarts"` and `"CyclicLR"`. Please find documentation here.
2. Other parameters that are needed for the scheduler of interest (e.g. `"base_lr": 1e-5, "max_lr": 1e-2` for `"CosineAnnealingLR"`).

5.4.5 balance_samples

Bool. Balance positive and negative labels in both the training and the validation datasets.

5.4.6 mixup_alpha

Float. Alpha parameter of the Beta distribution, see original paper on the Mixup technique.

5.4.7 transfer_learning

- `retrain_model`: Filename of the pretrained model (`path/to/pretrained-model`). If `null`, no transfer learning is performed and the network is trained from scratch.
- `retrain_fraction`: Float between 0. and 1. Controls the fraction of the pre-trained model that will be fine-tuned. For instance, if set to 0.5, the second half of the model will be fine-tuned while the first layers will be frozen.

5.5 Architecture

Architectures for both segmentation and classification are available and described in the *Models* section. If the selected architecture is listed in the loader file, a classification (not segmentation) task is run. In the case of a classification task, the ground truth will correspond to a single label value extracted from `target`, instead being an array (the latter being used for the segmentation task).

5.5.1 default_model (Mandatory)

Dict. Define the default model (`Unet`) and mandatory parameters that are common to all available architectures (listed in the *Models* section). For more specific models (see below), the default parameters are merged with the parameters that are specific to the tailored model. - `name`: `Unet` (default) - `dropout_rate`: Float (e.g. 0.4). - `batch_norm_momentum`: Float (e.g. 0.1). - `depth`: Strictly positive integer. Number of down-sampling operations.

5.5.2 FiLMedUnet (Optional)

- `applied`: Bool. Set to `true` to use this model.
- `metadata`: String. Choice between `"mri_params"` or `"contrast"`. `"mri_params"`: Vectors of `[FlipAngle, EchoTime, RepetitionTime, Manufacturer]` (defined in the json of each image) are input to the FiLM generator. `"contrast"`: Image contrasts (according to `config/contrast_dct.json`) are input to the FiLM generator.

5.5.3 HeMISUnet (Optional)

- `applied`: Bool. Set to `true` to use this model.
- `missing_probability`: Float between 0 and 1. Initial probability of missing image contrasts as model's input (e.g. 0.25 results in a quarter of the image contrasts, i.e. channels, that will not been sent to the model for training).
- `missing_probability_growth`: Float. Controls missing probability growth at each epoch: at each epoch, the `missing_probability` is modified with the exponent `missing_probability_growth`.

5.5.4 UNet3D (Optional)

- `length_3D`: (Int, Int, Int). Size of the 3D patches used as model's input tensors.
- `stride_3D`: [Int, Int, Int]. Voxels' shift over the input matrix to create patches. Ex: Stride of [1, 2, 3] will cause a patch translation of 1 voxel in the 1st dimension, 2 voxels in the 2nd dimension and 3 voxels in the 3rd dimension at every iteration until the whole input matrix is covered.
- `attention_unet`: Bool. Use attention gates in the Unet's decoder.

5.6 Testing parameters

- `binarize_prediction`: Bool. Binarize output predictions using a threshold of 0.5. If `false`, output predictions are float between 0 and 1.

5.6.1 uncertainty

Uncertainty computation is performed if `n_it>0` and at least `epistemic` or `aleatoric` is `true`. Note: both `epistemic` and `aleatoric` can be `true`. - `epistemic`: Bool. Model-based uncertainty with Monte Carlo Dropout. - `aleatoric`: Bool. Image-based uncertainty with test-time augmentation. - `n_it`: Integer. Number of Monte Carlo iterations. Set to 0 for no uncertainty computation.

5.7 Cascaded Architecture Features

5.7.1 object_detection_params (Optional)

- `object_detection_path`: String. Path to object detection model. The model's prediction will be used to generate bounding boxes.
- `safety_factor`: List. List of length 3 containing the factors to multiply each dimension of the bounding box. Ex: If the original bounding box has a size of 10x20x30 with a safety factor of [1.5, 1.5, 1.5], the final dimensions of the bounding box will be 15x30x45 with an unchanged center.

5.8 Transformations

Transformations applied during data augmentation. Transformations are sorted in the order they are applied to the image samples. For each transformation, the following parameters are customizable: - `applied_to`: list betweem `"im", "gt", "roi"`. If not specified, then the transformation is applied to all loaded samples. Otherwise, only applied to the specified types: eg `["gt"]` implies that this transformation is only applied to the ground-truth data. - `dataset_type`: list between `"training", "validation", "testing"`. If not specified, then the transformation is applied to the three sub-datasets. Otherwise, only applied to the specified subdatasets: eg `["testing"]` implies that this transformation is only applied to the testing sub-dataset.

5.8.1 Available transformations:

- `NumpyToTensor`
- `CenterCrop2D` (parameters: `size`)
- `ROICrop2D` (parameters: `size`)
- `NormalizeInstance`
- `RandomAffine` (parameters: `degrees` (Positive integer), `translate` (List of floats between 0. and 1.), `scale` (List of floats between 0. and 1.))
- `RandomShiftIntensity` (parameters: `shift_range`)
- `ElasticTransform` (parameters: `alpha_range`, `sigma_range`, `p`)
- `Resample` (parameters: `wspace`, `hspace`, `dspace`)
- `AdditionGaussianNoise` (parameters: `mean`, `std`)
- `DilateGT` (parameters: `dilation_factor`) Float. Controls the number of iterations of ground-truth dilation depending on the size of each individual lesion, data augmentation of the training set. Use `0` to disable.
- `HistogramClipping` (parameters: `min_percentile`, `max_percentile`)
- `Clage` (parameters: `clip_limit`, `kernel_size`)

- `RandomReverse`

5.9 Examples

Examples of configuration files: config_config.json.

In particular:

- config_classification.json. Is dedicated to classification task.
- config_sctTesting.json. Is a user case of 2D segmentation using a U-Net model.
- config_spineGeHemis.json. Shows how to use the HeMIS-UNet.
- config_tumorSeg.json. Runs a 3D segmentation using a 3D UNet.

CHAPTER 6

Data

Without data, nothing can be done. To get you started, we recommend you download the Spinal Cord MRI Public Database. This dataset is composed of 248+ subjects from different imaging centers and includes original images in NIfTI format as well as manual segmentations and labels. The data are organized according to the BIDS convention, to be fully compatible with `ivadomed` loader:

```
dataset/
└── dataset_description.json
└── participants.tsv
└── sub-01
    └── anat
        └── sub-siteX01_T1w_reg.nii.gz
        └── sub-siteX01_T1w_reg.json
        └── sub-siteX01_T2w_reg.nii.gz
        └── sub-siteX01_T2w_reg.json
        └── sub-siteX01_acq-MTon_MTS_reg.nii.gz
        └── sub-siteX01_acq-MTon_MTS_reg.json
        └── sub-siteX01_acq-MToff_MTS_reg.nii.gz
        └── sub-siteX01_acq-MToff_MTS_reg.json
        └── sub-siteX01_acq-T1w_MTS.nii.gz
        └── sub-siteX01_acq-T1w_MTS.json
        └── sub-siteX01_T2star_reg.nii.gz
        └── sub-siteX01_T2star_reg.json
└── derivatives
    └── labels
        └── sub-siteX01
            └── anat
                └── sub-siteX01_T1w_seg.nii.gz
```

Note: `participants.tsv` should, at least, include a column `participant_id`, which is used when loading the dataset.

Warning: TODO: Update openneuro site to include derivatives

CHAPTER 7

Models

7.1 Unet

class ivadomed.models.**Unet**(*in_channel=1*, *out_channel=1*, *depth=3*, *drop_rate=0.4*, *bn_momentum=0.1*, ***kwargs*)

Bases: torch.nn.modules.module.Module

A reference U-Net model.

See also:

Ronneberger, O., et al (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation ArXiv link: https://arxiv.org/abs/1505.04597

Parameters

- **in_channel** (*int*) – Number of channels in the input image.
- **out_channel** (*int*) – Number of channels in the output image.
- **depth** (*int*) – Number of down convolutions minus bottom down convolution.
- **drop_rate** (*float*) – Probability of dropout.
- **bn_momentum** (*float*) – Batch normalization momentum.
- ****kwargs** –

Attributes

- **encoder** (*Encoder*) – U-Net encoder.
- **decoder** (*Decoder*) – U-net decoder.

__init__(*in_channel=1*, *out_channel=1*, *depth=3*, *drop_rate=0.4*, *bn_momentum=0.1*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

7.2 FiLMedUnet

class ivadomed.models.**FiLMedUnet**(*in_channel=1*, *out_channel=1*, *depth=3*, *drop_rate=0.4*, *bn_momentum=0.1*, *n_metadata=None*, *film_layers=None*, ***kwargs*)

Bases: *ivadomed.models.Unet*

U-Net network containing FiLM layers to condition the model with another data type (i.e. not an image).

Parameters

- **n_channel** (*int*) – Number of channels in the input image.
- **out_channel** (*int*) – Number of channels in the output image.
- **depth** (*int*) – Number of down convolutions minus bottom down convolution.
- **drop_rate** (*float*) – Probability of dropout.
- **bn_momentum** (*float*) – Batch normalization momentum.
- **n_metadata** (*dict*) – FiLM metadata see ivadomed.loader.film for more details.
- **film_layers** (*list*) – List of 0 or 1 indicating on which layer FiLM is applied.
- ****kwargs** –

Attributes

- **encoder** (*Encoder*) – U-Net encoder.
- **decoder** (*Decoder*) – U-net decoder.

__init__(*in_channel=1*, *out_channel=1*, *depth=3*, *drop_rate=0.4*, *bn_momentum=0.1*, *n_metadata=None*, *film_layers=None*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*, *context=None*)
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.

7.3 HeMISUnet

class ivadomed.models.**HeMISUnet**(*contrasts*, *out_channel=1*, *depth=3*, *drop_rate=0.4*, *bn_momentum=0.1*, ***kwargs*)

Bases: torch.nn.modules.module.Module

A U-Net model inspired by HeMIS to deal with missing contrasts.

1) It has as many encoders as contrasts but only one decoder.

2) Skip connections are the concatenations of the means and var of all encoders skip connections.

Param: contrasts: list of all the possible contrasts. ['T1', 'T2', 'T2S', 'F']

See also:

Havaei, M., Guizard, N., Chapados, N., Bengio, Y.: Hemis: Hetero-modal image segmentation. ArXiv link: https://arxiv.org/abs/1607.05194

Reuben Dorent and Samuel Joutard and Marc Modat and Sébastien Ourselin and Tom Vercauteren Hetero-Modal Variational Encoder-Decoder for Joint Modality Completion and Segmentation ArXiv link: https://arxiv.org/abs/1907.11150

Parameters

- **contrasts** (*list*) – List of contrasts.
- **out_channel** (*int*) – Number of output channels.
- **depth** (*int*) – Number of down convolutions minus bottom down convolution.
- **drop_rate** (*float*) – Probability of dropout.
- **bn_momentum** (*float*) – Batch normalization momentum.
- ****kwargs** –

Attributes

- **depth** (*int*) – Number of down convolutions minus bottom down convolution.
- **contrasts** (*list*) – List of contrasts.
- **Encoder_mod** (*ModuleDict*) – Contains encoder for each modality.
- **decoder** (*Decoder*) – U-Net decoder.

__init__(*contrasts*, *out_channel=1*, *depth=3*, *drop_rate=0.4*, *bn_momentum=0.1*, ***kwargs*)

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x_mods*, *indexes_mod*)

X is list like X = [x_T1, x_T2, x_T2S, x_F] indexes_mod: list of arrays like [[1, 1, 1], [1, 1, 0], [1, 0, 1], [1, 1, 0]] N.B. len(list) = number of contrasts. len(list[i]) = Batch size

7.4 UNet3D

class ivadomed.models.**UNet3D**(*in_channel*, *out_channel*, *n_filters=16*, *attention=False*, *drop_rate=0.6*, *bn_momentum=0.1*, ***kwargs*)

Bases: torch.nn.modules.module.Module

Code from the following repository: https://github.com/pykao/Modified-3D-UNet-Pytorch The main differences with the original UNet resides in the use of LeakyReLU instead of ReLU, InstanceNormalisation instead of BatchNorm due to small batch size in 3D and the addition of segmentation layers in the decoder.

If attention=True, attention gates are added in the decoder to help focus attention on important features for a given task. Code related to the attentions gates is inspired from: https://github.com/ozan-oktay/Attention-Gated-Networks

Parameters

- **in_channel** (*int*) – Number of channels in the input image.
- **out_channel** (*int*) – Number of channels in the output image.
- **n_filters** (*int*) – Number of base filters in the U-Net.
- **attention** (*bool*) – Boolean indicating whether the attention module is on or not.
- **drop_rate** (*float*) – Probability of dropout.
- **bn_momentum** (*float*) – Batch normalization momentum.
- ****kwargs** –

Attributes

- **in_channels** (*int*) – Number of channels in the input image.
- **n_classes** (*int*) – Number of channels in the output image.
- **base_n_filter** (*int*) – Number of base filters in the U-Net.
- **attention** (*bool*) – Boolean indicating whether the attention module is on or not.
- **momentum** (*float*) – Batch normalization momentum.

Note: All layers are defined as attributes and used in the forward method.

__init__(*in_channel*, *out_channel*, *n_filters=16*, *attention=False*, *drop_rate=0.6*, *bn_momentum=0.1*, ***kwargs*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

7.5 Countception

class ivadomed.models.**Countception**(*in_channel=3*, *out_channel=1*, *use_logits=False*, *logits_per_output=12*, *name='CC'*)
Bases: `torch.nn.modules.module.Module`

Countception model. Fully convolutional model using inception module and used for keypoints detection. The inception model extracts several patches within each image. Every pixel is therefore processed by the network several times, allowing to average multiple predictions and minimize false negatives.

See also:

Cohen JP et al. “Count-ception: Counting by fully convolutional redundant counting.” Proceedings of the IEEE International Conference on Computer Vision Workshops. 2017.

Parameters

- **in_channel** (*int*) – number of channels on input image
- **out_channel** (*int*) – number of channels on output image
- **use_logits** (*bool*) – boolean to change output
- **logits_per_output** (*int*) – number of outputs of final convolution which will multiplied by the number of channels
- **name** (*str*) – model’s name used for call in configuration file.

__init__(*in_channel=3*, *out_channel=1*, *use_logits=False*, *logits_per_output=12*, *name='CC'*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

CHAPTER 8

Scripts

This section contains a collection of useful scripts for quality control during the training of models.

8.1 visualize_transforms

visualize_transforms.**run_visualization**(*fname_config*, *n_slices*, *folder_output*, *fname_roi*)

Utility function to visualize Data Augmentation transformations.

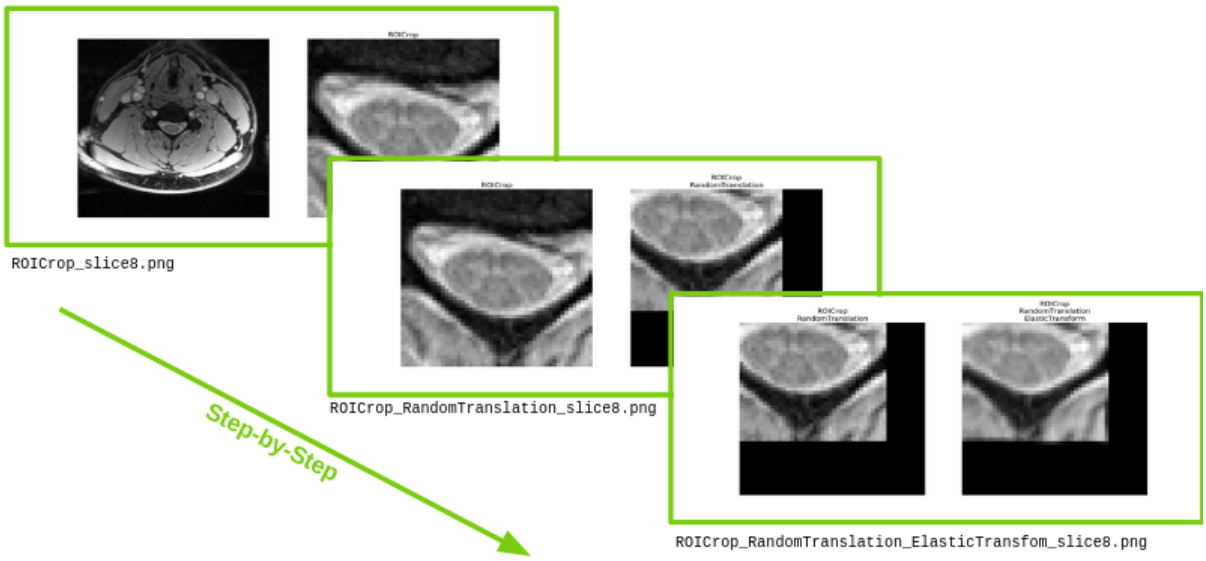
Data augmentation is a key part of the Deep Learning training scheme. This script aims at facilitating the fine-tuning of data augmentation parameters. To do so, this script provides a step-by-step visualization of the transformations that are applied on data.

This function applies a series of transformations (defined in a configuration file `-c`) to `-n` 2D slices randomly extracted from an input image (`-i`), and save as png the resulting sample after each transform.

For example:

```
python visualize_transforms.py -i t2s.nii.gz -n 1 -c config.json -r t2s_seg.nii.gz
```

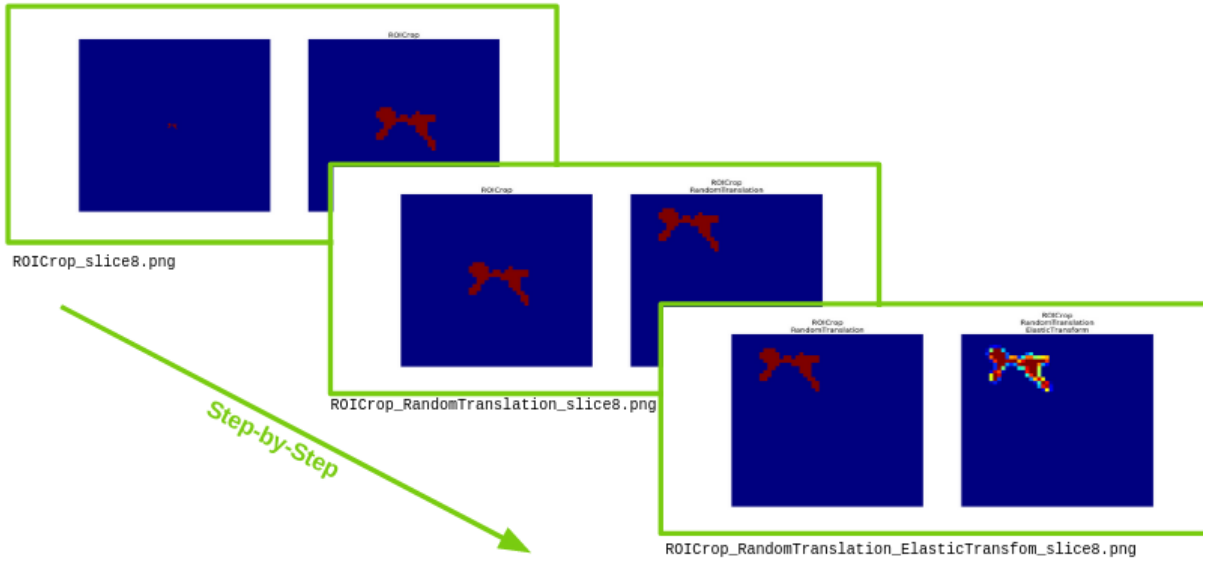
Provides a visualization of a series of three transformation on a randomly selected slice:



And on a binary mask:

```
python visualize_transforms.py -i t2s_gmseg.nii.gz -n 1 -c config.json -r t2s_seg.
↪nii.gz
```

Gives:



Parameters

- **fname_input** (*string*) – Image filename.
- **fname_config** (*string*) – Configuration file filename.
- **n_slices** (*int*) – Number of slices randomly extracted.
- **folder_output** (*string*) – Folder path where the results are saved.
- **fname_roi** (*string*) – Filename of the region of interest. Only needed if ROICrop is part of the transformations.

8.2 convert_to_onnx

convert_to_onnx.**convert_pytorch_to_onnx**(*dimension*, *gpu=0*)

Convert PyTorch model to ONNX.

The integration of Deep Learning models into the clinical routine requires cpu optimized models. To export the PyTorch models to ONNX format and to run the inference using ONNX Runtime is a time and memory efficient way to answer this need.

This function converts a model from PyTorch to ONNX format, with information of whether it is a 2D or 3D model (`-d`).

Parameters

- **fname_model** (*string*) – Model filename.
- **dimension** (*int*) – Indicates whether the model is 2D or 3D. Choice between 2 or 3.
- **gpu** (*string*) – GPU ID, if available

8.3 automate_training

automate_training.**automate_training**(*fname_param*, *fixed_split*, *all_combinations*, *n_iterations=1*, *run_test=False*)

Automate multiple training processes on multiple GPUs.

Hyperparameter optimization of models is tedious and time-consuming. This function automatizes this optimization across multiple GPUs. It runs trainings, on the same training and validation datasets, by combining a given set of parameters and set of values for each of these parameters. Results are collected for each combination and reported into a dataframe to allow their comparison. The script efficiently allocates each training to one of the available GPUs.

TODO: add example of DF

Parameters

- **fname_config** (*string*) – Configuration filename, which is used as skeleton to configure the training. Some of its parameters (defined in *fname_param* file) are modified across experiments.
- **fname_param** (*string*) – json file containing parameters configurations to compare. Parameter "keys" of this file need to match the parameter "keys" of *fname_config* file. Parameter "values" are in a list. Example:

```
"default_model": {"depth": [2, 3, 4]}
```

- **fixed_split** (*bool*) – If True, all the experiments are run on the same training/validation/testing subdatasets.
- **all_combinations** (*bool*) – If True, all parameters combinations are run.
- **n_iterations** (*int*) – Controls the number of time that each experiment (ie set of parameter) are run.
- **run_test** (*bool*) – If True, the trained model is also run on the testing subdataset.

8.4 compare_models

compare_models.**compute_statistics**(*n_iterations*, *run_test=True*)

Compares the performance of models at inference time on a common testing dataset using paired t-tests.

It uses a dataframe generated by scripts/automate_training.py with the parameter --run-test (used to run t
models on the testing dataset).

TODO: add example of DF

Parameters

- **dataframe** (*pandas.Dataframe*) – Dataframe of results generated by automate_training.
- **n_iterations** (*int*) – Indicates the number of time that each experiment (ie set of parameter) was run.
- **run_test** (*int*) – Indicates if the comparison is done on the performances on either the testing subdataset (True) either on the training/validation subdatasets.

8.5 prepare_dataset_vertebral_labeling

prepare_dataset_vertebral_labeling.**extract_mid_slice_and_convert_coordinates_to_heatmaps**(*su*
ai
1)

This function takes as input a path to a dataset and generates a set of images: (i) mid-sagittal image and (ii) heatmap of disc labels associated with the mid-sagittal image.

Example:

```
python scripts/prepare_dataset_vertebral_labeling -p path/to/bids -s _T2w -a 0
```

Parameters

- **bids_path** (*string*) – path to BIDS dataset form which images will be generated
- **suffix** (*string*) – suffix of image that will be processed (e.g., T2w)
- **aim** (*int*) – If aim is not 0, retrieves only labels with value = aim, else create heatmap with all labels.

Returns None. Images are saved in BIDS folder

8.6 extract_small_dataset

extract_small_dataset.**extract_small_dataset**(*ofolder*, *n=10*, *contrast_list=None*, *include_derivatives=True*, *seed=-1*)

Extract small BIDS dataset from a larger BIDS dataset.

Example:

```
python extract_small_dataset.py -i path/to/BIDS/dataset -o path/of/small/BIDS/
↪dataset -n 10 -c T1w,T2w -d 0 -s 1234
```

Parameters

- **ifolder** (*str*) – Input BIDS folder.
- **ofolder** (*str*) – Output folder.
- **n** (*int*) – Number of subjects in the output folder.
- **contrast_list** (*list*) – List of image contrasts to include. If set to None, then all available contrasts are included.
- **include_derivatives** (*bool*) – If True, derivatives/labels/ content is also copied, only the raw images otherwise.
- **seed** (*int*) – Set np.random.RandomState to ensure reproducibility: the same subjects will be selected if the function is run several times on the same dataset. If set to -1, each function run is independent.

CHAPTER 9

Contributing to ivadomed

9.1 Introduction

First off, thanks for taking the time to contribute!

When contributing to this repository, please first discuss the change you wish to make by opening a new Github issue.

Contributions relating to content of the Github repository can be submitted through Github pull requests (PR).

PR for bug fixes or new features should be based on the `master` branch.

The following Github documentation may be useful:

- See Using Pull Requests for more information about Pull Requests.
- See Fork A Repo for an introduction to forking a repository.
- See Creating branches for an introduction on branching within GitHub.

For external contributors: Please fork this repository, make the desired changes, and open a Pull request to have your code reviewed and merged.

For internal contributor: You can open a branch (see *Opening a Branch*) directly in this repository. If you don't have the rights, contact the team leader.

9.2 Opening an issue

Issues (bugs, feature requests, or others) can be submitted on our project's issue page.

9.2.1 Before Submitting a New Issue

Please take a few seconds to search the issue database in case the issue has already been raised.

When reporting an issue, make sure your installation has not been tempered with (and if you can, update to the latest release, maybe the problem was fixed).

9.2.2 Submitting an Issue

Issue Title

Try to have a self-descriptive, meaningful issue title, summarizing the problem you see.

Examples:

- *Crashes during image cropping*
- *Add a special mode for multi-class segmentation*

Issue Body

Describe the issue and mention the IVADO Medical Imaging version and OS that you are using.

If you experience an error, copy/paste the Terminal output (include your syntax) and please follow these guidelines for clarity:

- If there is less than 10 lines of text, embed it directly in your comment in Github. Use "~~~" to format as code.
- If there is 10+ lines, either use an external website such as pastebin (copy/paste your text and include the URL in your comment), or use collapsable Github markdown capabilities.

Add useful information such as screenshots, etc.

If you submit a feature request, provide a *usage scenario*, imagining how the feature would be used (ideally inputs, a sequence of commands, and a desired outcome). Also provide references to any theoretical work to help the reader better understand the feature.

9.3 Opening a Branch

If you are part of the core developer team, you can open a branch directly in this repository. Prefix the branch name with a personal identifier and a forward slash; If the branch you are working on is in response to an issue, provide the issue number; Add some text that make the branch name meaningful.

Examples:

- `ol/100-fixup-lr-scheduler`
- `ab/loader-pep8`

9.4 Developing

9.4.1 Conflicts

Make sure the PR changes are not in conflict with the master branch.

9.4.2 Code style

Please review your changes for styling issues, clarity, according to the PEP8 convention. Correct any code style suggested by an analyzer on your changes. PyCharm has a code analyser integrated or you can use pyflakes.

Do not address your functional changes in the same commits as any styling clean-up you may be doing on existing code.

9.4.3 Documentation and docstrings

If you are implementing a new feature, update the documentation to describe the feature, and comment the code (things that are not trivially understandable from the code) to improve its maintainability.

Make sure to cite any papers, algorithms or articles that can help understand the implementation of the feature. If you are implementing an algorithm described in a paper, add pointers to the section / steps.

Please use the Google style docstrings.

9.4.4 Testing

Please add tests, especially with new code. Unit tests are located under `testing/`. They are straightforward to augment, but we understand it's the extra mile; it would still be appreciated if you provide something lighter (eg. in the commit messages or in the PR or issue text) that demonstrates that an issue was fixed, or a feature is functional.

Consider that if you add test cases, they will ensure that your feature – which you probably care about – does not stop working in the future.

9.4.5 Licensing

Ensure that you are the original author of your changes, and if that is not the case, ensure that the borrowed/adapted code is compatible with the project's license.

9.5 Committing

9.5.1 Commit Titles

Provide a concise and self-descriptive title (avoid > 80 characters). You may "scope" the title using the applicable command name(s), folder or other "module" as a prefix. If a commit is responsible for fixing an issue, post-fix the description with `(fixes #ISSUE_NUMBER)`.

Examples:

```
testing: add testing function for validation metrics
loader: add timer
documentation: add slice_axis to the config files
model: add HeMIS network
```

9.5.2 Commit Sequences

Update your branch to be baseline on the latest master if new developments were merged while you were developing. Please prefer **rebasing** to merging, as explained in this tutorial. Note that if you do rebases after review have started, they will be cancelled, so at this point it may be more appropriate to do a pull.

Clean-up your commit sequence. If your are not familiar with git, this good tutorial on the subject may help you.

Focus on committing 1 logical change at a time. See this article on the subject.

9.6 Submitting a Pull Request

9.6.1 PR Title

The PR title is used to automatically generate the Changelog for each new release, so please follow the following rules:

- Provide a concise and self-descriptive title (see *Issue Title*).
- Do not include the applicable issue number in the title, do it in the PR body (see *PR Body*).
- If the PR is not ready for review, convert it to a draft.

9.6.2 PR Body

Describe what the PR is about, explain the approach and possible drawbacks. Don't hesitate to repeat some of the text from the related issue (easier to read than having to click on the link).

If the PR fixes issue(s), indicate it after your introduction: `Fixes #XXXX, Fixes #YYYY`. Note: it is important to respect the syntax above so that the issue(s) will be closed upon merging the PR.

9.6.3 Work in progress

If your PR is not ready for review yet, you can convert it to a "Draft", so the team is informed.

A draft pull request is styled differently to clearly indicate that it's in a draft state. Merging is blocked in draft pull requests. Change the status to "Ready for review" near the bottom of your pull request to remove the draft state and allow merging according to your project's settings.

9.6.4 Continuous Integration

The PR can't be merged if Github Actions "Run tests" hasn't succeeded. If you are familiar with it, consult the test results to fix the problem.

9.6.5 Reviewers

Any changes submitted for inclusion to the master branch will have to go through a review.

Only request a review when you deem the PR as "good to go". If the PR is not ready for review, convert it to a "Draft".

Github may suggest you to add particular reviewers to your PR. If that's the case and you don't know better, add all of these suggestions. The reviewers will be notified when you add them.

9.7 Versioning

Versioning uses the following convention: MAJOR.MINOR.PATCH, where:

PATCH version when there are backwards-compatible bug fixes or enhancements, without alteration to Python's modules or data/binaries. MINOR version when there are minor API changes or new functionality in a backwards-compatible manner, or when there are alteration to Python's modules or data/binaries (which requires to re-run installer for people working on the dev version), MAJOR version when there are major incompatible API changes, Beta releases follow the following convention:

MAJOR.MINOR.PATCH-beta.x (with x = 0, 1, 2, etc.) Stable version is indicated in the file version.txt. For development version (on master), the version is “dev”.

CHAPTER 10

API Reference

This document is for developers of `ivadomed`, it contains the API functions.

10.1 Loader API

10.1.1 loader.adaptative

class ivadomed.loader.adaptative.**Bids_to_hdf5**(*root_dir*, *subject_lst*, *target_suffix*, *contrast_lst*, *hdf5_name*, *contrast_balance=None*, *slice_axis=2*, *metadata_choice=False*, *slice_filter_fn=None*, *roi_suffix=None*, *transform=None*, *object_detection_params=None*)

Bases: `object`

Converts a BIDS dataset to a HDF5 file.

Parameters

- **root_dir** (*str*) – Path to the BIDS dataset.
- **subject_lst** (*list*) – Subject names list.
- **target_suffix** (*list*) – List of suffixes for target masks.
- **roi_suffix** (*str*) – List of suffixes for ROI masks.
- **contrast_lst** (*list*) – List of the contrasts.
- **hdf5_name** (*str*) – Path and name of the hdf5 file.
- **contrast_balance** (*dict*) – Dictionary controlling image contrasts balance.
- **slice_axis** (*int*) – Indicates the axis used to extract slices: “axial”: 2, “sagittal”: 0, “coronal”: 1.

- **metadata_choice** (*str*) – Choice between "mri_params", "contrasts", None or False, related to FiLM.
- **slice_filter_fn** (SliceFilter) – Class that filters slices according to their content.
- **transform** (Compose) – Transformations.
- **object_detection_params** (*dict*) – Object detection parameters.

Attributes

- **bids_ds** (*BIDS*) – BIDS dataset.
- **dt** (*dtype*) – hdf5 special dtype.
- **hdf5_file** (*hdf5*) – hdf5 file containing dataset information.
- **filename_pairs** (*list*) – A list of tuples in the format (input filename list containing all modalities,ground truth filename, ROI filename, metadata).
- **metadata** (*dict*) – Dictionary containing metadata of input and gt.
- **prepro_transforms** (*Compose*) – Transforms to be applied before training.
- **transform** (*Compose*) – Transforms to be applied during training.
- **has_bounding_box** (*bool*) – True if all metadata contains bounding box coordinates, else False.
- **slice_axis** (*int*) – Indicates the axis used to extract slices: "axial": 2, "sagittal": 0, "coronal": 1.
- **slice_filter_fn** (*SliceFilter*) – Object that filters slices according to their content.

__init__(*root_dir*, *subject_lst*, *target_suffix*, *contrast_lst*, *hdf5_name*, *contrast_balance=None*, *slice_axis=2*, *metadata_choice=False*, *slice_filter_fn=None*, *roi_suffix=None*, *transform=None*, *object_detection_params=None*)

Initialize self. See help(type(self)) for accurate signature.

class ivadomed.loader.adaptative.**Dataframe**(*hdf5*, *contrasts*, *path*, *target_suffix=None*, *roi_suffix=None*, *filter_slices=False*, *dim=2*)

Bases: object

This class aims to create a dataset using an HDF5 file, which can be used by an adapative loader to perform curriculum learning, Active Learning or any other strategy that needs to load samples in a specific way. It works on RAM or on the fly and can be saved for later.

Parameters

- **hdf5** (*hdf5*) – hdf5 file containing dataset information
- **contrasts** (*list of str*) – List of the contrasts of interest.
- **path** (*str*) – Dataframe path.
- **target_suffix** (*list of str*) – List of suffix of targetted structures.
- **roi_suffix** (*str*) – List of suffix of ROI masks.
- **filter_slices** (SliceFilter) – Object that filters slices according to their content.
- **dim** (*int*) – Choice 2 or 3, for 2D or 3D data respectively.

Attributes

- **dim** (*int*) – Choice 2 or 3, for 2D or 3D data respectively.

- **contrasts** (*list of str*) – List of the contrasts of interest.
- **filter_slices** (*SliceFilter*) – Object that filters slices according to their content.
- **df** (*pd.Dataframe*) – Dataframe containing dataset information

__init__(*hdf5*, *contrasts*, *path*, *target_suffix=None*, *roi_suffix=None*, *filter_slices=False*, *dim=2*)
Initialize self. See help(type(self)) for accurate signature.

clean(*contrasts*)
Aims to remove lines where one of the contrasts in not available.

Agrs: contrasts (list of str): List of contrasts.

create_df(*hdf5*)
Generate the Data frame using the hdf5 file.

Parameters **hdf5** (*hdf5*) – File containing dataset information

load_dataframe(*path*)
Load the dataframe from a csv file.

Parameters **path** (*str*) – Path to hdf5 file.

save(*path*)
Save the dataframe into a csv file.

Parameters **path** (*str*) – Path to hdf5 file.

shuffle()
Shuffle the whole data frame.

class ivadomed.loader.adaptative.**HDF5Dataset**(*root_dir*, *subject_lst*, *model_params*, *target_suffix*, *contrast_params*, *slice_axis=2*, *transform=None*, *metadata_choice=False*, *dim=2*, *complet=True*, *slice_filter_fn=None*, *roi_suffix=None*, *object_detection_params=None*)

Bases: object

HDF5 dataset object.

Parameters

- **root_dir** (*path*) – Path of bids and data.
- **subject_lst** (*list of str*) – List of subjects.
- **model_params** (*dict*) – Dictionary containing model parameters.
- **target_suffix** (*list of str*) – List of suffixes of the target structures.
- **contrast_params** (*dict*) – Dictionary containing contrast parameters.
- **slice_axis** (*int*) – Indicates the axis used to extract slices: "axial": 2, "sagittal": 0, "coronal": 1.
- **transform** (Compose) – Transformations.
- **metadata_choice** (*str*) – Choice between "mri_params", "contrasts", None or False, related to FiLM.
- **dim** (*int*) – Choice 2 or 3, for 2D or 3D data respectively.
- **complet** (*bool*) – If True removes lines where contrasts is not available.

- **slice_filter_fn** (SliceFilter) – Object that filters slices according to their content.
- **roi_suffix** (*str*) – Suffix of the ROI mask.
- **object_detection_params** (*dict*) – Object detection parameters.

Attributes

- **cst_lst** (*list*) – Contrast list.
- **gt_lst** (*list*) – Contrast label used for ground truth.
- **roi_lst** (*list*) – Contrast label used for ROI cropping.
- **dim** (*int*) – Choice 2 or 3, for 2D or 3D data respectively.
- **filter_slices** (*SliceFilter*) – Object that filters slices according to their content.
- **prepro_transforms** (*Compose*) – Transforms to be applied before training.
- **transform** (*Compose*) – Transforms to be applied during training.
- **df_object** (*pd.Dataframe*) – Dataframe containing dataset information.

__getitem__(*index*)
Get samples.

Warning: For now, this method only supports one gt / roi.

Parameters **index** (*int*) – Sample index.

Returns Dictionary containing image and label tensors as well as metadata.

Return type dict

__init__(*root_dir*, *subject_lst*, *model_params*, *target_suffix*, *contrast_params*, *slice_axis=2*, *transform=None*, *metadata_choice=False*, *dim=2*, *complet=True*, *slice_filter_fn=None*, *roi_suffix=None*, *object_detection_params=None*)
Initialize self. See help(type(self)) for accurate signature.

__len__()
Get the dataset size, ie he number of subvolumes.

load_into_ram(*contrast_lst=None*)
Aims to load into RAM the contrasts from the list.

Parameters **contrast_lst** (*list of str*) – List of contrasts of interest.

set_transform(*transform*)
Set the transforms.

update(*strategy='Missing'*, *p=0.0001*)
Update the Dataframe itself.

Parameters

- **p** (*float*) – Float between 0 and 1, probability of the contrast to be missing.
- **strategy** (*str*) – Update the dataframe using the corresponding strategy. For now the only the strategy implemented is the one used by HeMIS (i.e. by removing contrasts with a certain probability.) Other strategies that could be implemented are Active Learning, Curriculum Learning, ...

ivadomed.loader.adaptative.**HDF5_to_Bids**(*HDF5*, *subjects*, *path_dir*)
Convert HDF5 file to BIDS dataset.

Parameters

- **HDF5** (*str*) – Path to the HDF5 file.
- **subjects** (*list*) – List of subject names.
- **path_dir** (*str*) – Output folder path, already existing.

10.1.2 loader.film

class ivadomed.loader.film.**Kde_model**
Bases: object

Kernel Density Estimation.

Apply this clustering method to metadata values, using (sklearn implementation.)

Attributes

- **kde** (*sklearn.neighbors.KernelDensity*)
- **minima** (*float*) – Local minima.

__init__()
Initialize self. See help(type(self)) for accurate signature.

ivadomed.loader.film.**check_isMRIparam**(*mri_param_type*, *mri_param*, *subject*, *metadata*)
Check if a given metadata belongs to the MRI parameters.

Parameters

- **mri_param_type** (*str*) – Metadata type name.
- **mri_param** (*list*) – List of MRI params names.
- **subject** (*str*) – Current subject name.
- **metadata** (*dict*) – Metadata.

Returns True if *mri_param_type* is part of *mri_param*.

Return type bool

ivadomed.loader.film.**clustering_fit**(*dataset*, *key_lst*)
This function creates clustering models for each metadata type, using Kernel Density Estimation algorithm.

Parameters

- **datasets** (*list*) – data
- **key_lst** (*list of str*) – names of metadata to cluster

Returns Clustering model for each metadata type in a dictionary where the keys are the metadata names.

Return type dict

ivadomed.loader.film.**get_film_metadata_models**(*ds_train*, *metadata_type*, *debugging=False*)
Get FiLM models.

This function pulls the clustering and one-hot encoder models that are used by FiLMedUnet. It also calls the normalization of metadata.

Parameters

- **ds_train** (MRI2DSegmentationDataset) – training dataset
- **metadata_type** (*string*) – eg mri_params, contrasts

- **debugging** (*bool*) –

Returns dataset, one-hot encoder and KDE model

Return type *MRI2DSegmentationDataset*, OneHotEncoder, KernelDensity

ivadomed.loader.film.**normalize_metadata**(*ds_in*, *clustering_models*, *debugging*, *metadata_type*, *train_set=False*)

Categorize each metadata value using a KDE clustering method, then apply a one-hot-encoding.

Parameters

- **ds_in** (BidsDataset) – Dataset with metadata.
- **clustering_models** – Pre-trained clustering model that has been trained on metadata of the training set.
- **debugging** (*bool*) – If True, extended verbosity and intermediate outputs.
- **metadata_type** (*str*) – Choice between 'mri_params' and 'constrasts'.
- **train_set** (*bool*) – Indicates if the input dataset is the training dataset (True) or the validation or testing dataset (False).

Returns

Dataset with normalized metadata. If train_set is True, then the one-hot-encoder model is also returned.

Return type *BidsDataset*

10.1.3 loader.loader

class ivadomed.loader.loader.**Bids3DDataset**(*root_dir*, *subject_lst*, *target_suffix*, *model_params*, *contrast_params*, *slice_axis=2*, *cache=True*, *transform=None*, *metadata_choice=False*, *roi_suffix=None*, *multichannel=False*, *object_detection_params=None*)

Bases: *ivadomed.loader.loader.MRI3DSubVolumeSegmentationDataset*

BIDS specific dataset loader for 3D dataset.

Parameters

- **root_dir** (*str*) – Path to the BIDS dataset.
- **subject_lst** (*list*) – Subject names list.
- **target_suffix** (*list*) – List of suffixes for target masks.
- **model_params** (*dict*) – Dictionary containing model parameters.
- **contrast_params** (*dict*) – Contains image contrasts related parameters.
- **slice_axis** (*int*) – Indicates the axis used to extract slices: "axial": 2, "sagittal": 0, "coronal": 1.
- **cache** (*bool*) – If the data should be cached in memory or not.
- **transform** (*list*) – Transformation list (length 2) composed of preprocessing transforms (Compose) and transforms to apply during training (Compose).
- **metadata_choice** – Choice between "mri_params", "contrasts", None or False, related to FiLM.

- **roi_suffix** (*list*) – List of suffixes for ROI masks.
- **multichannel** (*bool*) – If True, the input contrasts are combined as input channels for the model. Otherwise, each contrast is processed individually (ie different sample / tensor).
- **object_detection_params** (*dict*) – Object dection parameters.

__init__(*root_dir*, *subject_lst*, *target_suffix*, *model_params*, *contrast_params*, *slice_axis=2*, *cache=True*, *transform=None*, *metadata_choice=False*, *roi_suffix=None*, *multichannel=False*, *object_detection_params=None*)

Initialize self. See help(type(self)) for accurate signature.

class ivadomed.loader.loader.**BidsDataset**(*root_dir*, *subject_lst*, *target_suffix*, *contrast_params*, *slice_axis=2*, *cache=True*, *transform=None*, *metadata_choice=False*, *slice_filter_fn=None*, *roi_suffix=None*, *multichannel=False*, *object_detection_params=None*, *task='segmentation'*, *soft_gt=False*)

Bases: *ivadomed.loader.loader.MRI2DSegmentationDataset*

BIDS specific dataset loader.

Parameters

- **root_dir** (*str*) – Path to the BIDS dataset.
- **subject_lst** (*list*) – Subject names list.
- **target_suffix** (*list*) – List of suffixes for target masks.
- **contrast_params** (*dict*) – Contains image contrasts related parameters.
- **slice_axis** (*int*) – Indicates the axis used to extract slices: "axial": 2, "sagittal": 0, "coronal": 1.
- **cache** (*bool*) – If the data should be cached in memory or not.
- **transform** (*list*) – Transformation list (length 2) composed of preprocessing transforms (Compose) and transforms to apply during training (Compose).
- **metadata_choice** (*str*) – Choice between "mri_params", "contrasts", None or False, relatec to FiLM.
- **slice_filter_fn** (SliceFilter) – Class that filters slices according to their content.
- **roi_suffix** (*list*) – List of suffixes for ROI masks.
- **multichannel** (*bool*) – If True, the input contrasts are combined as input channels for the model. Otherwise, each contrast is processed individually (ie different sample / tensor).
- **object_detection_params** (*dict*) – Object dection parameters.
- **task** (*str*) – Choice between segmentation or classification. If classification: GT is discrete values, If segmentation: GT is binary mask.
- **soft_gt** (*bool*) – If True, ground truths will be converted to float32, otherwise to uint8 and binarized (to save memory).

Attributes

- **bids_ds** (*BIDS*) – BIDS dataset.
- **filename_pairs** (*list*) – A list of tuples in the format (input filename list containing all modalities,ground truth filename, ROI filename, metadata).

- **metadata** (*dict*) – Dictionary containing FiLM metadata.

__init__(*root_dir*, *subject_lst*, *target_suffix*, *contrast_params*, *slice_axis=2*, *cache=True*, *transform=None*, *metadata_choice=False*, *slice_filter_fn=None*, *roi_suffix=None*, *multichannel=False*, *object_detection_params=None*, *task='segmentation'*, *soft_gt=False*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.loader.loader.**MRI2DSegmentationDataset**(*filename_pairs*, *slice_axis=2*, *cache=True*, *transform=None*, *slice_filter_fn=None*, *task='segmentation'*, *soft_gt=False*)

Bases: torch.utils.data.dataset.Dataset

Generic class for 2D (slice-wise) segmentation dataset.

Parameters

- **filename_pairs** (*list*) – a list of tuples in the format (input filename list containing all modalities,ground truth filename, ROI filename, metadata).
- **slice_axis** (*int*) – axis to make the slicing (default axial).
- **cache** (*bool*) – if the data should be cached in memory or not.
- **transform** (*torchvision.Compose*) – transformations to apply.
- **slice_filter_fn** (*dict*) – Slice filter parameters, see *Configuration File* for more details.
- **task** (*str*) – choice between segmentation or classification. If classification: GT is discrete values, If segmentation: GT is binary mask.

Attributes

- **indexes** (*list*) – List of indices corresponding to each slice or subvolume in the dataset.
- **filename_pairs** (*list*) – List of tuples in the format (input filename list containing all modalities,ground truth filename, ROI filename, metadata).
- **prepro_transforms** (*Compose*) – Transformations to apply before training.
- **transform** (*Compose*) – Transformations to apply during training.
- **cache** (*bool*) – Tf the data should be cached in memory or not.
- **slice_axis** (*int*) – Indicates the axis used to extract slices: "axial": 2, "sagittal": 0, "coronal": 1.
- **slice_filter_fn** (*dict*) – Slice filter parameters, see *Configuration File* for more details.
- **n_contrasts** (*int*) – Number of input contrasts.
- **has_bounding_box** (*bool*) – True if bounding box in all metadata, else False.
- **task** (*str*) – Choice between segmentation or classification. If classification: GT is discrete values, If segmentation: GT is binary mask.
- **soft_gt** (*bool*) – If True, ground truths are expected to be non-binarized images encoded in float32 and will be fed as is to the network. Otherwise, ground truths are converted to uint8 and binarized to save memory space.

__getitem__(*index*)
Return the specific processed data corresponding to index (input, ground truth, roi and metadata).

Parameters **index** (*int*) – Slice index.

__init__ (*filename_pairs*, *slice_axis=2*, *cache=True*, *transform=None*, *slice_filter_fn=None*, *task='segmentation'*, *soft_gt=False*)
Initialize self. See help(type(self)) for accurate signature.

load_filenames()
Load preprocessed pair data (input and gt) in handler.

class ivadomed.loader.loader.**MRI3DSubVolumeSegmentationDataset**(*filename_pairs*, *transform=None*, *length=(64, 64, 64)*, *stride=(0, 0, 0)*, *slice_axis=0*)

Bases: torch.utils.data.dataset.Dataset

This is a class for 3D segmentation dataset. This class splits the initials volumes in several subvolumes. Each subvolumes will be of the sizes of the length parameter.

This class also implement a stride parameter corresponding to the amount of voxels subvolumes are translated in each dimension at every iteration.

Be careful, the input's dimensions should be compatible with the given lengths and strides. This class doesn't handle missing dimensions.

Parameters

- **filename_pairs** (*list*) – A list of tuples in the format (input filename, ground truth filename).
- **transform** (Compose) – Transformations to apply.
- **length** (*tuple*) – Size of each dimensions of the subvolumes, length equals 3.
- **stride** (*tuple*) – Size of the overlapping per subvolume and dimensions, length equals 3.
- **slice_axis** (*int*) – Indicates the axis used to extract slices: "axial": 2, "sagittal": 0, "coronal": 1.

__getitem__(*index*)
Return the specific index pair subvolume (input, ground truth).

Parameters **index** (*int*) – Subvolume index.

__init__(*filename_pairs*, *transform=None*, *length=(64, 64, 64)*, *stride=(0, 0, 0)*, *slice_axis=0*)
Initialize self. See help(type(self)) for accurate signature.

__len__()
Return the dataset size. The number of subvolumes.

class ivadomed.loader.loader.**SegmentationPair**(*input_filenames*, *gt_filenames*, *metadata=None*, *slice_axis=2*, *cache=True*, *prepro_transforms=None*, *soft_gt=False*)

Bases: object

This class is used to build segmentation datasets. It represents a pair of of two data volumes (the input data and the ground truth data).

Parameters

- **input_filenames** (*list of str*) – The input filename list (supported by nibabel). For single channel, the list will contain 1 input filename.

- **gt_filenames** (*list of str*) – The ground-truth filenames list.
- **metadata** (*list*) – Metadata list with each item corresponding to an image (contrast) in input_filenames. For single channel, the list will contain metadata related to one image.
- **cache** (*bool*) – If the data should be cached in memory or not.
- **slice_axis** (*int*) – Indicates the axis used to extract slices: "axial": 2, "sagittal": 0, "coronal": 1.
- **prepro_transforms** (*dict*) – Output of get_preprocessing_transforms.
- **soft_gt** (*bool*) – If True, ground truths will be converted to float32, otherwise to uint8 and binarized (to save memory).

Attributes

- **input_filenames** (*list*) – List of input filenames.
- **gt_filenames** (*list*) – List of ground truth filenames.
- **metadata** (*dict*) – Dictionary containing metadata of input and gt.
- **cache** (*bool*) – If the data should be cached in memory or not.
- **slice_axis** (*int*) – Indicates the axis used to extract slices: "axial": 2, "sagittal": 0, "coronal": 1.
- **prepro_transforms** (*dict*) – Transforms to be applied before training.
- **input_handle** (*list*) – List of input nifty data.
- **gt_handle** (*list*) – List of gt nifty data.

__init__(*input_filenames*, *gt_filenames*, *metadata=None*, *slice_axis=2*, *cache=True*, *prepro_transforms=None*, *soft_gt=False*)
Initialize self. See help(type(self)) for accurate signature.

get_pair_data()
Return the tuple (input, ground truth) with the data content in numpy array.

get_pair_metadata(*slice_index=0*, *coord=None*)
Return dictionary containing input and gt metadata.

Parameters

- **slice_index** (*int*) – Index of 2D slice if 2D model is used, else 0.
- **coord** (*tuple or list*) – Coordinates of subvolume in volume if 3D model is used, else None.

Returns Input and gt metadata.

Return type dict

get_pair_shapes()
Return the tuple (input, ground truth) representing both the input and ground truth shapes.

get_pair_slice(*slice_index*, *gt_type='segmentation'*)
Return the specified slice from (input, ground truth).

Parameters

- **slice_index** (*int*) – Slice number.
- **gt_type** (*str*) – Choice between segmentation or classification, returns mask (array) or label (int) resp. for the ground truth.

ivadomed.loader.loader.**load_dataset**(*data_list*, *bids_path*, *transforms_params*, *model_params*, *target_suffix*, *roi_params*, *contrast_params*, *slice_filter_params*, *slice_axis*, *multichannel*, *dataset_type='training'*, *requires_undo=False*, *metadata_type=None*, *object_detection_params=None*, *soft_gt=False*, ***kwargs*)

Get loader appropriate loader according to model type. Available loaders are Bids3DDataset for 3D data, BidsDataset for 2D data and HDF5Dataset for HeMIS.

Parameters

- **data_list** (*list*) – Subject names list.
- **bids_path** (*str*) – Path to the BIDS dataset.
- **transforms_params** (*dict*) – Dictionary containing transformations for "training", "validation", "testing" (keys), eg output of imed_transforms.get_subdatasets_transforms.
- **model_params** (*dict*) – Dictionary containing model parameters.
- **target_suffix** (*list of str*) – List of suffixes for target masks.
- **roi_params** (*dict*) – Contains ROI related parameters.
- **contrast_params** (*dict*) – Contains image contrasts related parameters.
- **slice_filter_params** (*dict*) – Contains slice_filter parameters, see *Configuration File* for more details.
- **slice_axis** (*string*) – Choice between "axial", "sagittal", "coronal" ; controls the axis used to extract the 2D data.
- **multichannel** (*bool*) – If True, the input contrasts are combined as input channels for the model. Otherwise, each contrast is processed individually (ie different sample / tensor).
- **metadata_type** (*str*) – Choice between None, "mri_params", "contrasts".
- **dataset_type** (*str*) – Choice between "training", "validation" or "testing".
- **requires_undo** (*bool*) – If True, the transformations without undo_transform will be discarded.
- **object_detection_params** (*dict*) – Object dection parameters.
- **soft_gt** (*bool*) – If True, ground truths will be converted to float32, otherwise to uint8 and binarized (to save memory).

Returns BidsDataset

Note: For more details on the parameters transform_params, target_suffix, roi_params, contrast_params, slice_filter_params and object_detection_params see *Configuration File*.

10.1.4 loader.utils

class ivadomed.loader.utils.**BalancedSampler**(*dataset*)

Bases: torch.utils.data.sampler.Sampler

Estimate sampling weights in order to rebalance the class distributions from an imbalanced dataset.

Parameters **dataset** (BidsDataset) – Dataset containing input, gt and metadata.

Attributes

- **indices** (*list*) – List from 0 to length of dataset (number of elements in the dataset).

- **nb_samples** (*int*) – Number of elements in the dataset.
- **weights** (*Tensor*) – Weight of each dataset element equal to 1 over the frequency of a given label (inverse of the frequency).

__init__(*dataset*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.loader.utils.**SampleMetadata**(*d=None*)
Bases: object

Metadata class to help update, get and set metadata values.

Parameters **d** (*dict*) – Initial metadata.

Attributes **metadata** (*dict*) – Image metadata.

__init__(*d=None*)
Initialize self. See help(type(self)) for accurate signature.

ivadomed.loader.utils.**clean_metadata**(*metadata_lst*)
Remove keys from metadata. The keys to be deleted are stored in a list.

Parameters **metadata_lst** (*list*) – List of SampleMetadata.

Returns List of SampleMetadata with removed keys.

Return type list

ivadomed.loader.utils.**filter_roi**(*ds*, *nb_nonzero_thr*)
Filter slices from dataset using ROI data.

This function loops across the dataset (ds) and discards slices where the number of non-zero voxels within the ROI slice (e.g. centerline, SC segmentation) is inferior or equal to a given threshold (nb_nonzero_thr).

Parameters

- **ds** (*mt_datasets.MRI2DSegmentationDataset*) – Dataset.
- **nb_nonzero_thr** (*int*) – Threshold.

Returns Dataset without filtered slices.

Return type mt_datasets.MRI2DSegmentationDataset

ivadomed.loader.utils.**get_new_subject_split**(*path_folder*, *center_test*, *split_method*, *random_seed*, *train_frac*, *test_frac*, *log_directory*)
Randomly split dataset between training / validation / testing.

Randomly split dataset between training / validation / testing and save it in log_directory + "/split_datasets.joblib".

Parameters

- **path_folder** (*string*) – Dataset folder.
- **center_test** (*list*) – List of centers to include in the testing set.
- **split_method** (*string*) – See imed_loader_utils.split_dataset.
- **random_seed** (*int*) – Random seed.
- **train_frac** (*float*) – Training dataset proportion, between 0 and 1.
- **test_frac** (*float*) – Testing dataset proportionm between 0 and 1.
- **log_directory** (*string*) – Output folder.

Returns Training, validation and testing subjects lists.

Return type list, list list

ivadomed.loader.utils.**get_subdatasets_subjects_list**(*split_params*, *bids_path*, *log_directory*)

Get lists of subjects for each sub-dataset between training / validation / testing.

Parameters

- **split_params** (*dict*) – Split parameters, see *Configuration File* for more details.
- **bids_path** (*str*) – Path to the BIDS dataset.
- **log_directory** (*str*) – Output folder.

Returns Training, validation and testing subjects lists.

Return type list, list list

ivadomed.loader.utils.**imed_collate**(*batch*)

Collates data to create batches

Parameters **batch** (*dict*) – Contains input and gt data with their corresponding metadata.

Returns Collated data.

Return type list or dict or str or tensor

ivadomed.loader.utils.**orient_img_hwd**(*data*, *slice_axis*)

Orient a given RAS image to height, width, depth according to slice axis.

Parameters

- **data** (*ndarray*) – RAS oriented data.
- **slice_axis** (*int*) – Indicates the axis used for the 2D slice extraction: Sagittal: 0, Coronal: 1, Axial: 2.

Returns Array oriented with the following dimensions: (height, width, depth).

Return type ndarray

ivadomed.loader.utils.**orient_img_ras**(*data*, *slice_axis*)

Orient a given array with dimensions (height, width, depth) to RAS orientation.

Parameters

- **data** (*ndarray*) – Data with following dimensions (Height, Width, Depth).
- **slice_axis** (*int*) – Indicates the axis used for the 2D slice extraction: Sagittal: 0, Coronal: 1, Axial: 2.

Returns Array oriented in RAS.

Return type ndarray

ivadomed.loader.utils.**orient_shapes_hwd**(*data*, *slice_axis*)

Swap dimensions according to match the height, width, depth orientation.

Parameters

- **data** (*list or tuple*) – Shape or numbers associated with each image dimension (e.i. image resolution).
- **slice_axis** (*int*) – Indicates the axis used for the 2D slice extraction: Sagittal: 0, Coronal: 1, Axial: 2.

Returns Reoriented vector.

Return type ndarray

ivadomed.loader.utils.**split_dataset**(*path_folder*, *center_test_lst*, *split_method*, *random_seed*, *train_frac=0.8*, *test_frac=0.1*)

Splits list of subject into training, validation and testing datasets either according to their center or per patient. In the 'per_center' option the centers associated the subjects are split according the train, test and validation fraction whereas in the 'per_patient', the patients are directly separated according to these fractions.

Parameters

- **path_folder** (*str*) – Path to BIDS folder.
- **center_test_lst** (*list*) – list of centers to include in the testing set.
- **split_method** (*str*) – Between 'per_center' or 'per_person'. If 'per_center' the separation fraction are applied to centers, if 'per_person' they are applied to the subject list.
- **random_seed** (*int*) – Random seed to ensure reproducible splits.
- **train_frac** (*float*) – Between 0 and 1. Represents the train set proportion.
- **test_frac** (*float*) – Between 0 and 1. Represents the test set proportion.

Returns Train, validation and test subjects list.

Return type list, list, list

ivadomed.loader.utils.**update_metadata**(*metadata_src_lst*, *metadata_dest_lst*)

Update metadata keys with a reference metadata. A given list of metadata keys will be changed and given the values of the reference metadata.

Parameters

- **metadata_src_lst** (*list*) – List of source metadata used as reference for the destination metadata.
- **metadata_dest_lst** (*list*) – List of metadate that needs to be updated.

Returns updated metadata list.

Return type list

10.2 Object Detection API

10.2.1 object_detection.utils

ivadomed.object_detection.utils.**adjust_bb_size**(*bounding_box*, *factor*, *resample=False*)

Modifies the bounding box dimensions according to a given factor.

Parameters

- **bounding_box** (*list or tuple*) – Coordinates of bounding box (x_min, x_max, y_min, y_max, z_min, z_max).
- **factor** (*list or tuple*) – Multiplicative factor for each dimension (list or tuple of length 3).
- **resample** (*bool*) – Boolean indicating if this resize is for resampling.

Returns New coordinates (x_min, x_max, y_min, y_max, z_min, z_max).

Return type list

ivadomed.object_detection.utils.**adjust_transforms**(*transforms*, *seg_pair*, *length=None*, *stride=None*)

This function adapts the transforms by adding the BoundingBoxCrop transform according the specific parameters of an image. The dimensions of the crop are also adapted to fit the length and stride parameters if the 3D loader is used.

Parameters

- **transforms** (Compose) – Prepreocessing transforms.
- **seg_pair** (*dict*) – Segmentation pair (input, gt and metadata).
- **length** (*list or tuple*) – Patch size of the 3D loader.
- **stride** (*list or tuple*) – Stride value of the 3D loader.

Returns Modified transforms.

Return type *Compose*

ivadomed.object_detection.utils.**adjust_undo_transforms**(*transforms*, *seg_pair*, *index=0*)

This function adapts the undo transforms by adding the BoundingBoxCrop to undo transform according the specific parameters of an image.

Parameters

- **transforms** (Compose) – Transforms.
- **seg_pair** (*dict*) – Segmentation pair (input, gt and metadata).
- **index** (*int*) – Batch index of the seg_pair.

ivadomed.object_detection.utils.**bounding_box_prior**(*fname_mask*, *metadata*, *slice_axis*)

Computes prior steps to a model requiring bounding box crop. This includes loading a mask of the ROI, orienting the given mask into the following dimensions: (height, width, depth), extracting the bounding boxes and storing the information in the metadata.

Parameters

- **fname_mask** (*str*) – Filename containing the mask of the ROI
- **metadata** (*dict*) – Dictionary containing the image metadata
- **slice_axis** (*int*) – Slice axis (0: sagittal, 1: coronal, 2: axial)

ivadomed.object_detection.utils.**compute_bb_statistics**(*bounding_box_path*)

Measures min, max and average, height, width, depth and volume of bounding boxes from a json file

Parameters **bounding_box_path** (*string*) – Path to json file.

ivadomed.object_detection.utils.**generate_bounding_box_file**(*subject_list*, *model_path*, *log_dir*, *gpu_number=0*, *slice_axis=0*, *contrast_lst=None*, *keep_largest_only=True*, *safety_factor=None*)

Creates json file containing the bounding box dimension for each images. The file has the following format: {"path/to/img.nii.gz": [[x1_min, x1_max, y1_min, y1_max, z1_min, z1_max], [x2_min, x2_max, y2_min, y2_max, z2_min, z2_max]]} where each list represents the coordinates of an object on the image (2 instance of a given object in this example).

Parameters

- **subject_list** (*list*) – List of all subjects in the BIDS directory.
- **model_path** (*string*) – Path to object detection model.
- **log_dir** (*string*) – Log directory.
- **gpu_number** (*int*) – If available, GPU number.
- **slice_axis** (*int*) – Slice axis (0: sagittal, 1: coronal, 2: axial).
- **contrast_lst** (*list*) – Contrasts.
- **keep_largest_only** (*bool*) – Boolean representing if only the largest object of the prediction is kept.
- **safety_factor** (*list or tuple*) – Factors to multiply each dimension of the bounding box.

Returns Dictionary containing bounding boxes related to their image.

Return type dict

ivadomed.object_detection.utils.**get_bounding_boxes**(*mask*)

Generates a 3D bounding box around a given mask. :param mask: Mask of the ROI. :type mask: Numpy array

Returns Bounding box coordinate (x_min, x_max, y_min, y_max, z_min, z_max).

Return type list

ivadomed.object_detection.utils.**load_bounding_boxes**(*object_detection_params*, *subjects*, *slice_axis*, *constrast_lst*)

Verifies if bounding_box.json exists in the log directory, if so loads the data, else generates the file if a valid detection model path exists.

Parameters

- **object_detection_params** (*dict*) – Object detection parameters.
- **subjects** (*list*) – List of all subjects in the BIDS directory.
- **slice_axis** (*int*) – Slice axis (0: sagittal, 1: coronal, 2: axial).
- **constrast_lst** (*list*) – Contrasts.

Returns bounding boxes for every subject in BIDS directory

Return type dict

ivadomed.object_detection.utils.**resample_bounding_box**(*metadata*, *transform*)

Resample bounding box.

Parameters

- **metadata** (*dict*) – Dictionary containing the metadata to be modified with the resampled coordinates.
- **transform** (Compose) – Transformations possibly containing the resample params.

ivadomed.object_detection.utils.**resize_to_multiple**(*shape*, *multiple*, *length*)

Modify a given shape so each dimension is a multiple of a given number. This is used to avoid dimension mismatch with patch training. The return shape is always larger then the initial shape (no cropping).

Parameters

- **shape** (*tuple or list*) – Initial shape to be modified.
- **multiple** (*tuple or list*) – Multiple for each dimension.
- **length** (*tuple or list*) – Patch length.

Returns New dimensions.

Return type list

ivadomed.object_detection.utils.**verify_metadata**(*metadata*, *has_bounding_box*)

Validates across all metadata that the 'bounding_box' param is present.

Parameters

- **metadata** (*dict*) – Image metadata.
- **has_bounding_box** (*bool*) – If 'bounding_box' is present across all metadata, True, else False.

Returns Boolean indicating if 'bounding_box' is present across all metadata.

Return type bool

10.3 Evaluation API

class ivadomed.evaluation.**Evaluation3DMetrics**(*data_pred*, *data_gt*, *dim_lst*, *params=None*)

Bases: object

Computes 3D evaluation metrics.

Parameters

- **data_pred** (*ndarray*) – Network prediction mask.
- **data_gt** (*ndarray*) – Ground-truth mask.
- **dim_lst** (*list*) – Resolution (mm) along each dimension.
- **params** (*dict*) – Evaluation parameters.

Attributes

- **data_pred** (*ndarray*) – Network prediction mask.
- **data_gt** (*ndarray*) – Ground-truth mask.
- **n_classes** (*int*) – Number of classes.
- **px** (*float*) – Resolution (mm) along the first axis.
- **py** (*float*) – Resolution (mm) along the second axis.
- **pz** (*float*) – Resolution (mm) along the third axis.
- **bin_struct** (*ndarray*) – Binary structure.
- **size_min** (*int*) – Minimum size of objects. Objects that are smaller than this limit can be removed if "removeSmall" is in params.
- **overlap_vox** (*int*) – A prediction and ground-truth are considered as overlapping if they overlap for at least this amount of voxels.
- **overlap_ratio** (*float*) – A prediction and ground-truth are considered as overlapping if they overlap for at least this portion of their volumes.
- **data_pred_label** (*ndarray*) – Network prediction mask that is labeled, ie each object is filled with a different value.

- **data_gt_label** (*ndarray*) – Ground-truth mask that is labeled, ie each object is filled with a different value.
- **n_pred** (*int*) – Number of objects in the network prediction mask.
- **n_gt** (*int*) – Number of objects in the ground-truth mask.
- **data_painted** (*ndarray*) – Mask where each predicted object is labeled depending on whether it is a TP or FP.

__init__ (*data_pred*, *data_gt*, *dim_lst*, *params=None*)
Initialize self. See help(type(self)) for accurate signature.

get_avd()
Absolute volume difference.

get_lfdr (*label_size=None*, *class_idx=0*)
Lesion False Detection Rate / 1 - Precision.

Parameters

- **label_size** (*int*) – Size of label.
- **class_idx** (*int*) – Label index. If monolabel 0, else ranges from 0 to number of output channels - 1.

Note: computed only if n_obj >= 1.

get_ltpr (*label_size=None*, *class_idx=0*)
Lesion True Positive Rate / Recall / Sensitivity.

Parameters

- **label_size** (*int*) – Size of label.
- **class_idx** (*int*) – Label index. If monolabel 0, else ranges from 0 to number of output channels - 1.

Note: computed only if n_obj >= 1.

get_rvd()
Relative volume difference.

get_vol (*data*)
Get volume.

label_per_size (*data*)
Get data with labels corresponding to label size.

Parameters **data** (*ndarray*) – Input data.

Returns ndarray

remove_small_objects (*data*)
Removes all unconnected objects smaller than the minimum specified size.

Parameters **data** (*ndarray*) – Input data.

Returns Array with small objects.

Return type ndarray

run_eval()
Stores evaluation results in dictionary

Returns dictionary containing evaluation results, data with each object painted a different color

Return type dict, ndarray

ivadomed.evaluation.**evaluate**(*bids_path*, *log_directory*, *path_preds*, *target_suffix*, *eval_params*)
Evaluate predictions from inference step.

Parameters

- **bids_path** (*str*) – Folder where raw data is stored.
- **log_directory** (*str*) – Folder where the output folder "results_eval" is be created.
- **path_preds** (*str*) – Folder where model predictions were saved
- **target_suffix** (*list*) – List of suffixes that indicates the target mask(s).
- **eval_params** (*dict*) – Evaluation parameters.

Returns results for each image.

Return type pd.Dataframe

10.4 Losses API

class ivadomed.losses.**AdapWingLoss**(*theta=0.5*, *alpha=2.1*, *omega=14*, *epsilon=1*)
Bases: torch.nn.modules.module.Module

Adaptive Wing loss Used for heatmap ground truth.

..seealso:: Wang, Xinyao, Liefeng Bo, and Li Fuxin. "Adaptive wing loss for robust face alignment via heatmap regression." Proceedings of the IEEE International Conference on Computer Vision. 2019.

Parameters

- **theta** (*float*) – Threshold between linear and non linear loss.
- **alpha** (*float*) – Used to adapt loss shape to input shape and make loss smooth at 0 (background).
- **needs to be slightly above 2 to maintain ideal properties.** (*It*) –
- **omega** (*float*) – Multiplicating factor for non linear part of the loss.
- **epsilon** (*float*) – factor to avoid gradient explosion. It must not be too small

__init__(*theta=0.5*, *alpha=2.1*, *omega=14*, *epsilon=1*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class ivadomed.losses.**BinaryCrossEntropyLoss**
Bases: torch.nn.modules.module.Module

(BinaryCrossEntropyLoss).

Attributes **loss_fct** (*BCELoss*) – Binary cross entropy loss function from torch library.

__init__()
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*prediction*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class ivadomed.losses.**DiceLoss**(*smooth=1.0*)
Bases: `torch.nn.modules.module.Module`

DiceLoss.

See also:

Milletari, Fausto, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-net: Fully convolutional neural networks for volumetric medical image segmentation." 2016 fourth international conference on 3D vision (3DV). IEEE, 2016.

Parameters **smooth** (*float*) – Value to avoid division by zero when images and predictions are empty.

Attributes **smooth** (*float*) – Value to avoid division by zero when images and predictions are empty.

__init__(*smooth=1.0*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*prediction*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class ivadomed.losses.**FocalDiceLoss**(*beta=1*, *gamma=2*, *alpha=0.25*)
Bases: `torch.nn.modules.module.Module`

FocalDiceLoss.

See also:

Wong, Ken CL, et al. "3D segmentation with exponential logarithmic loss for highly unbalanced object sizes." International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, Cham, 2018.

Parameters

- **beta** (*float*) – Value from 0 to 1, indicating the weight of the dice loss.
- **gamma** (*float*) – Value from 0 to 5, Control between easy background and hard ROI training examples. If set to 0, equivalent to cross-entropy.

- **alpha** (*float*) – Value from 0 to 1, usually corresponding to the inverse of class frequency to address class imbalance.

Attributes

- **beta** (*float*) – Value from 0 to 1, indicating the weight of the dice loss.
- **gamma** (*float*) – Value from 0 to 5, Control between easy background and hard ROI training examples. If set to 0, equivalent to cross-entropy.
- **alpha** (*float*) – Value from 0 to 1, usually corresponding to the inverse of class frequency to address class imbalance.

__init__(*beta=1*, *gamma=2*, *alpha=0.25*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class ivadomed.losses.**FocalLoss**(*gamma=2*, *alpha=0.25*, *eps=1e-07*)
Bases: `torch.nn.modules.module.Module`

FocalLoss.

See also:

Lin, Tsung-Yi, et al. "Focal loss for dense object detection." Proceedings of the IEEE international conference on computer vision. 2017.

Parameters

- **gamma** (*float*) – Value from 0 to 5, Control between easy background and hard ROI training examples. If set to 0, equivalent to cross-entropy.
- **alpha** (*float*) – Value from 0 to 1, usually corresponding to the inverse of class frequency to address class imbalance.
- **eps** (*float*) – Epsilon to avoid division by zero.

Attributes

- **gamma** (*float*) – Value from 0 to 5, Control between easy background and hard ROI training examples. If set to 0, equivalent to cross-entropy.
- **alpha** (*float*) – Value from 0 to 1, usually corresponding to the inverse of class frequency to address class imbalance.
- **eps** (*float*) – Epsilon to avoid division by zero.

__init__(*gamma=2*, *alpha=0.25*, *eps=1e-07*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

class ivadomed.losses.**FocalTverskyLoss**(*alpha=0.7*, *beta=0.3*, *gamma=1.33*, *smooth=1.0*)
Bases: *ivadomed.losses.TverskyLoss*

Focal Tversky Loss.

See also:

Abraham, Nabila, and Naimul Mefraz Khan. "A novel focal tversky loss function with improved attention u-net for lesion segmentation." 2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019). IEEE, 2019.

Parameters

- **alpha** (*float*) – Weight of false positive voxels.
- **beta** (*float*) – Weight of false negative voxels.
- **gamma** (*float*) – Typically between 1 and 3. Control between easy background and hard ROI training examples.
- **smooth** (*float*) – Epsilon to avoid division by zero, when both Numerator and Denominator of Tversky are zeros.

Attributes **gamma** (*float*) – Typically between 1 and 3. Control between easy background and hard ROI training examples.

Notes

- setting alpha=beta=0.5 and gamma=1: Equivalent to DiceLoss.
- default parameters were suggested by https://arxiv.org/pdf/1810.07842.pdf .

__init__(*alpha=0.7*, *beta=0.3*, *gamma=1.33*, *smooth=1.0*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

class ivadomed.losses.**GeneralizedDiceLoss**(*epsilon=1e-05*, *include_background=True*)
Bases: torch.nn.modules.module.Module

GeneralizedDiceLoss.

See also:

Sudre, Carole H., et al. "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations." Deep learning in medical image analysis and multimodal learning for clinical decision support. Springer, Cham, 2017. 240-248.

Parameters

- **epsilon** (*float*) – Epsilon to avoid division by zero.
- **include_background** (*float*) – If True, then an extra channel is added, which represents the background class.

Attributes

- **epsilon** (*float*) – Epsilon to avoid division by zero.
- **include_background** (*float*) – If True, then an extra channel is added, which represents the background class.

__init__(*epsilon=1e-05*, *include_background=True*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class ivadomed.losses.**L2loss**
Bases: torch.nn.modules.module.Module

Euclidean loss also known as L2 loss. Compute the sum of the squared difference between the two images.

__init__()
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class ivadomed.losses.**LossCombination**(*losses_list*, *params_list=None*)
Bases: torch.nn.modules.module.Module

Loss that sums other implemented losses.

Parameters

- **losses_list** (*list*) – list of losses that will be summed. Elements should be string.
- **params_list** (*list*) – list of params for the losses, contain None or dictionnary definition of params for the loss
- **same index. If no params list is given all default parameter will be used.** (*at*) –
- **losses_list = ["L2loss","DiceLoss"]** (*(e.g.,)* – params_list = [None,{"param1:0.5"}])

Returns sum of losses computed on (input,target) with the params

Return type tensor

__init__(*losses_list*, *params_list=None*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class ivadomed.losses.**MultiClassDiceLoss**(*classes_of_interest=None*)
Bases: `torch.nn.modules.module.Module`

Multi-class Dice Loss.

Inspired from https://arxiv.org/pdf/1802.10508.

Parameters **classes_of_interest** (*list*) – List containing the index of a class which its dice will be added to the loss. If is None all classes are considered.

Attributes

- **classes_of_interest** (*list*) – List containing the index of a class which its dice will be added to the loss. If is None all classes are considered.
- **dice_loss** (*DiceLoss*) – Class computing the Dice loss.

__init__(*classes_of_interest=None*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*prediction*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class ivadomed.losses.**TverskyLoss**(*alpha=0.7*, *beta=0.3*, *smooth=1.0*)
Bases: `torch.nn.modules.module.Module`

Tversky Loss.

See also:

Salehi, Seyed Sadegh Mohseni, Deniz Erdogmus, and Ali Gholipour. "Tversky loss function for image segmentation using 3D fully convolutional deep networks." International Workshop on Machine Learning in Medical Imaging. Springer, Cham, 2017.

Parameters

- **alpha** (*float*) – Weight of false positive voxels.
- **beta** (*float*) – Weight of false negative voxels.

- **smooth** (*float*) – Epsilon to avoid division by zero, when both Numerator and Denominator of Tversky are zeros.

Attributes

- **alpha** (*float*) – Weight of false positive voxels.
- **beta** (*float*) – Weight of false negative voxels.
- **smooth** (*float*) – Epsilon to avoid division by zero, when both Numerator and Denominator of Tversky are zeros.

Notes

- setting alpha=beta=0.5: Equivalent to DiceLoss.
- default parameters were suggested by https://arxiv.org/pdf/1706.05721.pdf .

__init__(*alpha=0.7*, *beta=0.3*, *smooth=1.0*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*, *target*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

tversky_index(*y_pred*, *y_true*)
Compute Tversky index.

Parameters

- **y_pred** (*torch Tensor*) – Prediction.
- **y_true** (*torch Tensor*) – Target.

Returns Tversky index.

Return type float

10.5 Main API

ivadomed.main.**run_main**(*config=None*)
Run main command.

This function is central in the ivadomed project as training / testing / evaluation commands are run via this function. All the process parameters are defined in the config.

Parameters **config** (*dict*) – Dictionary containing all parameters that are needed for a given process. See *Configuration File* for more details.

Returns Returns floats: best loss score for both training and validation. If "test" command: Returns dict: of averaged metrics computed on the testing sub dataset. If "eval" command: Returns a pandas Dataframe: of metrics computed for each subject of the testing sub dataset.

Return type If "train" command

10.6 Metrics API

class ivadomed.metrics.**MetricManager**(*metric_fns*)
Bases: object

Computes specified metrics and stores them in a dictionary.

Parameters **metric_fns** (*list*) – List of metric functions.

Attributes

- **metric_fns** (*list*) – List of metric functions.
- **result_dict** (*dict*) – Dictionary storing metrics.
- **num_samples** (*int*) – Number of samples.

__call__(*prediction*, *ground_truth*)
Call self as a function.

__init__(*metric_fns*)
Initialize self. See help(type(self)) for accurate signature.

ivadomed.metrics.**accuracy_score**(*prediction*, *groundtruth*)
Accuracy.

Parameters

- **prediction** (*ndarray*) – First array.
- **groundtruth** (*ndarray*) – Second array.

Returns Accuracy.

Return type float

ivadomed.metrics.**dice_score**(*im1*, *im2*, *empty_score=nan*)
Computes the Dice coefficient between im1 and im2.

Compute a soft Dice coefficient between im1 and im2. If both images are empty, then it returns empty_score.

Parameters

- **im1** (*ndarray*) – First array.
- **im2** (*ndarray*) – Second array.
- **empty_score** (*float*) – Returned value if both input array are empty.

Returns Dice coefficient.

Return type float

ivadomed.metrics.**hausdorff_score**(*prediction*, *groundtruth*)
Compute the directed Hausdorff distance between two N-D arrays.

Parameters

- **prediction** (*ndarray*) – First array.
- **groundtruth** (*ndarray*) – Second array.

Returns Hausdorff distance.

Return type float

ivadomed.metrics.**intersection_over_union**(*prediction*, *groundtruth*, *err_value=0.0*)
Intersection of two arrays over their union (IoU).

Parameters

- **prediction** (*ndarray*) – First array.
- **groundtruth** (*ndarray*) – Second array.
- **err_value** (*float*) – Value returned in case of error.

Returns IoU.

Return type float

ivadomed.metrics.**mse**(*im1*, *im2*)

Compute the Mean Squared Error.

Compute the Mean Squared Error between the two images, i.e. sum of the squared difference.

Parameters

- **im1** (*ndarray*) – First array.
- **im2** (*ndarray*) – Second array.

Returns Mean Squared Error.

Return type float

ivadomed.metrics.**multi_class_dice_score**(*im1*, *im2*)

Dice score for multi-label images.

Parameters

- **im1** (*ndarray*) – First array.
- **im2** (*ndarray*) – Second array.

Returns Multi-class dice.

Return type float

ivadomed.metrics.**numeric_score**(*prediction*, *groundtruth*)

Computation of statistical numerical scores:

- FP = False Positives
- FN = False Negatives
- TP = True Positives
- TN = True Negatives

Parameters

- **prediction** (*ndarray*) – Binary prediction.
- **groundtruth** (*ndarray*) – Binary groundtruth.

Returns FP, FN, TP, TN

Return type float, float, float, float

ivadomed.metrics.**precision_score**(*prediction*, *groundtruth*, *err_value=0.0*)

Positive predictive value (PPV).

Parameters

- **prediction** (*ndarray*) – First array.
- **groundtruth** (*ndarray*) – Second array.

- **err_value** (*float*) – Value returned in case of error.

Returns Precision score.

Return type float

ivadomed.metrics.**recall_score**(*prediction*, *groundtruth*, *err_value=0.0*)
True positive rate (TPR).

Parameters

- **prediction** (*ndarray*) – First array.
- **groundtruth** (*ndarray*) – Second array.
- **err_value** (*float*) – Value returned in case of error.

Returns Recall score.

Return type float

ivadomed.metrics.**specificity_score**(*prediction*, *groundtruth*, *err_value=0.0*)
True negative rate (TNR).

Parameters

- **prediction** (*ndarray*) – First array.
- **groundtruth** (*ndarray*) – Second array.
- **err_value** (*float*) – Value returned in case of error.

Returns Specificity score.

Return type float

10.7 Postprocessing API

ivadomed.postprocessing.**binarize_with_low_threshold**(*wrapped*)
Decorator to set low values (< 0.001) to 0.

Parameters **wrapped** – Given function.

Returns Functions' return.

ivadomed.postprocessing.**coordinate_from_heatmap**(*nifti_image*, *thresh=0.3*)
Retrieve coordinates of local maxima in a soft segmentation. :param nifti_image: nifti image of the soft segmentation. :type nifti_image: nibabel object :param thresh: Relative threshold for local maxima, i.e., after normalizing :type thresh: float :param the min and max between 0 and 1, respectively.:

Returns A list of computed coordinates found by local maximum. each element will be a list composed of [x, y, z]

Return type list

ivadomed.postprocessing.**fill_holes**(*predictions*, *structure=(3, 3, 3)*)
Fill holes in the predictions using a given structuring element. Note: This function only works for binary segmentation.

Parameters

- **predictions** (*ndarray or nibabel object*) – Input binary segmentation. Image could be 2D or 3D.

- **structure** (*tuple of integers*) – Structuring element, number of ints equals number of dimensions in the input array.

Returns ndrray or nibabel (same object as the input). Output type is int.

ivadomed.postprocessing.**keep_largest_object**(*predictions*)
Keep the largest connected object from the input array (2D or 3D).

Parameters **predictions** (*ndarray or nibabel object*) – Input segmentation. Image could be 2D or 3D.

Returns ndarray or nibabel (same object as the input).

ivadomed.postprocessing.**keep_largest_object_per_slice**(*predictions*, *axis=2*)
Keep the largest connected object for each 2D slice, along a specified axis.

Parameters

- **predictions** (*ndarray or nibabel object*) – Input segmentation. Image could be 2D or 3D.
- **axis** (*int*) – 2D slices are extracted along this axis.

Returns ndarray or nibabel (same object as the input).

ivadomed.postprocessing.**label_file_from_coordinates**(*nifti_image*, *coord_list*)
Creates a nifti object with single-voxel labels. Each label has a value of 1. The nifti object as the same orientation as the input. :param nifti_image: Path to the image which affine matrix will be used to generate a new image with :type nifti_image: nibabel object :param labels.: :param coord_list: list of coordinates. Each element is [x, y, z]. Orientation should be the same as the image :type coord_list: list

Returns A nifti object containing the singe-voxel label of value 1. The matrix will be the same size as *nifti_image*.

Return type nib_pred

ivadomed.postprocessing.**mask_predictions**(*predictions*, *mask_binary*)
Mask predictions using a binary mask: sets everything outside the mask to zero.

Parameters

- **predictions** (*ndarray or nibabel object*) – Input binary segmentation. Image could be 2D or 3D.
- **mask_binary** (*ndarray*) – Numpy array with the same shape as predictions, containing only zeros or ones.

Returns ndarray or nibabel (same object as the input).

ivadomed.postprocessing.**multilabel_capable**(*wrapped*)
Decorator to make a given function compatible multilabel images.

Parameters **wrapped** – Given function.

Returns Functions' return.

ivadomed.postprocessing.**nifti_capable**(*wrapped*)
Decorator to make a given function compatible with input being Nifti objects.

Parameters **wrapped** – Given function.

Returns Functions' return.

ivadomed.postprocessing.**threshold_predictions**(*predictions*, *thr=0.5*)
Threshold a soft (i.e. not binary) array of predictions given a threshold value, and returns a binary array.

Parameters

- **predictions** (*ndarray or nibabel object*) – Image to binarize.
- **thr** (*float*) – Threshold value: voxels with a value < to thr are assigned 0 as value, 1 otherwise.

Returns ndarray or nibabel (same object as the input) containing only zeros or ones. Output type is int.

Return type ndarray

10.8 Testing API

ivadomed.testing.**run_inference**(*test_loader*, *model*, *model_params*, *testing_params*, *ofolder*, *cuda_available*, *i_monte_carlo=None*)

Run inference on the test data and save results as nibabel files.

Parameters

- **test_loader** (*torch DataLoader*) –
- **model** (*nn.Module*) –
- **model_params** (*dict*) –
- **testing_params** (*dict*) –
- **ofolder** (*str*) – Folder where predictions are saved.
- **cuda_available** (*bool*) – If True, CUDA is available.
- **i_monte_carlo** (*int*) – i_th Monte Carlo iteration.

Returns Prediction, Ground-truth of shape n_sample, n_label, h, w, d.

Return type ndarray, ndarray

ivadomed.testing.**test**(*model_params*, *dataset_test*, *testing_params*, *log_directory*, *device*, *cuda_available=True*, *metric_fns=None*)

Main command to test the network.

Parameters

- **model_params** (*dict*) – Model's parameters.
- **dataset_test** (*imed_loader*) – Testing dataset.
- **testing_params** (*dict*) – Testing parameters.
- **log_directory** (*str*) – Folder where predictions are saved.
- **device** (*torch.device*) – Indicates the CPU or GPU ID.
- **cuda_available** (*bool*) – If True, CUDA is available.
- **metric_fns** (*list*) – List of metrics, see *ivadomed.metrics*.

Returns result metrics.

Return type dict

10.9 Training API

ivadomed.training.**get_loss_function**(*params*)
Get Loss function.

Parameters **params** (*dict*) – See *ivadomed.losses*.

Returns imed_losses object.

ivadomed.training.**get_metadata**(*metadata*, *model_params*)
Get metadata during batch loop.

Parameters

- **metadata** (*batch*) –
- **model_params** (*dict*) –

Returns If FiLMedUnet, Returns a list of metadata, that have been transformed by the One Hot Encoder. If HeMISUnet, Returns a numpy array where each row represents a sample and each column represents a contrast.

ivadomed.training.**get_sampler**(*ds*, *balance_bool*)
Get sampler.

Parameters

- **ds** (BidsDataset) – BidsDataset object.
- **balance_bool** (*bool*) – If True, a sampler is generated that balance positive and negative samples.

Returns Returns BalancedSampler, Bool: Sampler and boolean for shuffling (set to False). Otherwise: Returns None and True.

Return type If balance_bool is True

ivadomed.training.**get_scheduler**(*params*, *optimizer*, *num_epochs=0*)
Get scheduler.

Parameters

- **params** (*dict*) – scheduler parameters, see PyTorch documentation
- **optimizer** (*torch optim*) –
- **num_epochs** (*int*) – number of epochs.

Returns torch.optim, bool, which indicates if the scheduler is updated for each batch (True), or for each epoch (False).

ivadomed.training.**save_film_params**(*gammas*, *betas*, *contrasts*, *depth*, *ofolder*)
Save FiLM params as npy files.

These parameters can be further used for visualisation purposes. They are saved in the *ofolder* with *.npy* format.

Parameters

- **gammas** (*dict*) –
- **betas** (*dict*) –
- **contrasts** (*list*) – list of the batch sample's contrasts (eg T2w, T1w)
- **depth** (*int*) –
- **ofolder** (*str*) –

ivadomed.training.**store_film_params**(*gammas*, *betas*, *contrasts*, *metadata*, *model*, *film_layers*, *depth*)

Store FiLM params.

Parameters

- **gammas** (*dict*) –
- **betas** (*dict*) –
- **contrasts** (*list*) – list of the batch sample's contrasts (eg T2w, T1w)
- **metadata** (*list*) –
- **model** (*nn.Module*) –
- **film_layers** (*list*) –
- **depth** (*int*) –

Returns gammas, betas

Return type dict, dict

ivadomed.training.**train**(*model_params*, *dataset_train*, *dataset_val*, *training_params*, *log_directory*, *device*, *cuda_available=True*, *metric_fns=None*, *debugging=False*)

Main command to train the network.

Parameters

- **model_params** (*dict*) – Model's parameters.
- **dataset_train** (*imed_loader*) – Training dataset.
- **dataset_val** (*imed_loader*) – Validation dataset.
- **training_params** (*dict*) –
- **log_directory** (*str*) – Folder where log files, best and final models are saved.
- **device** (*str*) – Indicates the CPU or GPU ID.
- **cuda_available** (*bool*) – If True, CUDA is available.
- **metric_fns** (*list*) – List of metrics, see *ivadomed.metrics*.
- **debugging** (*bool*) – If True, extended verbosity and intermediate outputs.

Returns best_training_dice, best_training_loss, best_validation_dice, best_validation_loss.

Return type float, float, float, float

10.10 Transformations API

class ivadomed.transforms.**AdditiveGaussianNoise**(*mean=0.0*, *std=0.01*)

Bases: *ivadomed.transforms.ImedTransform*

Adds Gaussian Noise to images.

Parameters

- **mean** (*float*) – Gaussian noise mean.
- **std** (*float*) – Gaussian noise standard deviation.

__call__(*sample*, *metadata=None*)

Call self as a function.

__init__(*mean=0.0*, *std=0.01*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.transforms.**BoundingBoxCrop**(*size*)
Bases: *ivadomed.transforms.Crop*

Crops image according to given bounding box.

__call__(*sample*, *metadata*)
Call self as a function.

class ivadomed.transforms.**CenterCrop**(*size*)
Bases: *ivadomed.transforms.Crop*

Make a centered crop of a specified size.

__call__(*sample*, *metadata=None*)
Call self as a function.

class ivadomed.transforms.**Clahe**(*clip_limit=3.0*, *kernel_size=(8, 8)*)
Bases: *ivadomed.transforms.ImedTransform*

Applies Contrast Limited Adaptive Histogram Equalization for enhancing the local image contrast.

See also:

Zuiderveld, Karel. "Contrast limited adaptive histogram equalization." Graphics gems (1994): 474-485.

Default values are based on:: .. seealso:

```
Zheng, Qiao, et al. "3-D consistent and robust segmentation of cardiac images by␣
↪deep learning with spatial
propagation." IEEE transactions on medical imaging 37.9 (2018): 2137-2148.
```

Parameters

- **clip_limit** (*float*) – Clipping limit, normalized between 0 and 1.
- **kernel_size** (*tuple of int*) – Defines the shape of contextual regions used in the algorithm. Length equals image
- **dimension** (*ie 2 or 3 for 2D or 3D, respectively*) –

__call__(*sample*, *metadata=None*)
Call self as a function.

__init__(*clip_limit=3.0*, *kernel_size=(8, 8)*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.transforms.**Compose**(*dict_transforms*, *requires_undo=False*)
Bases: object

Composes transforms together.

Composes transforms together and split between images, GT and ROI.

self.transform is a dict:

- keys: "im", "gt" and "roi"
- values torchvision_transform.Compose objects.

Attributes

- **dict_transforms** (*dict*) – Dictionary where the keys are the transform names and the value their parameters.
- **requires_undo** (*bool*) – If True, does not include transforms which do not have an undo_transform implemented yet.

Parameters **transform** (*dict*) – Keys are "im", "gt", "roi" and values are torchvision_transforms.Compose of the transformations of interest.

__call__(*sample*, *metadata*, *data_type='im'*)
Call self as a function.

__init__(*dict_transforms*, *requires_undo=False*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.transforms.**Crop**(*size*)
Bases: *ivadomed.transforms.ImedTransform*

Crop data.

Parameters **size** (*tuple of int*) – Size of the output sample. Tuple of size 2 if dealing with 2D samples, 3 with 3D samples.

Attributes **size** (*tuple of int*) – Size of the output sample. Tuple of size 3.

__call__(*sample*, *metadata*)
Call self as a function.

__init__(*size*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.transforms.**CroppableArray**
Bases: numpy.ndarray

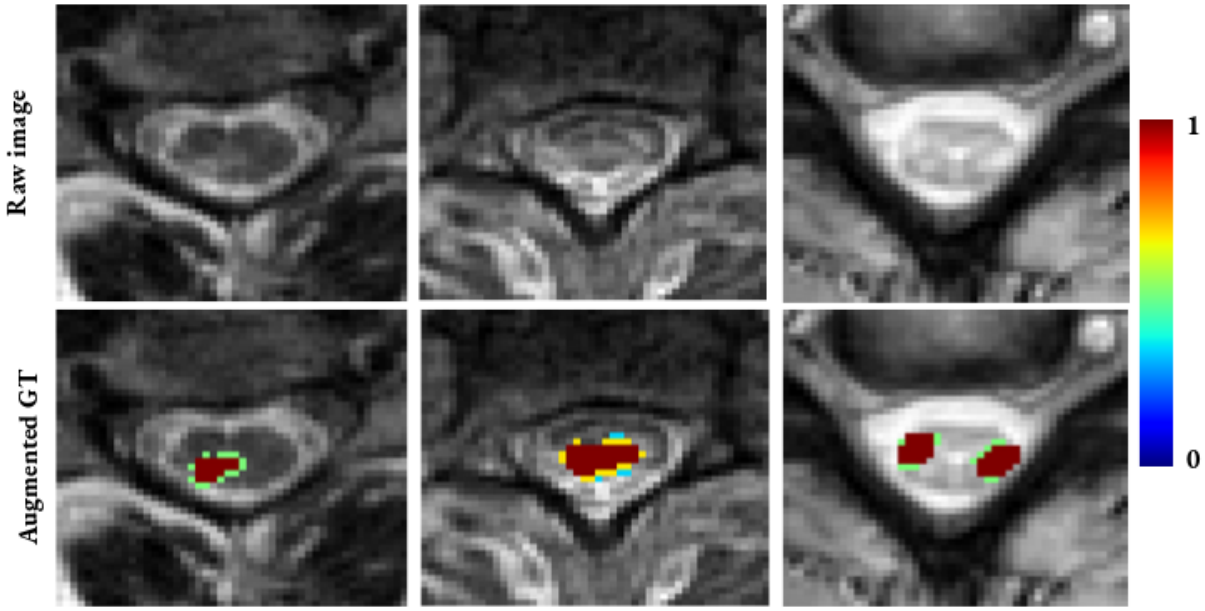
Zero padding slice past end of array in numpy.

Adapted From: https://stackoverflow.com/a/41155020/13306686

__getitem__(*item*)
Return self[key].

class ivadomed.transforms.**DilateGT**(*dilation_factor*)
Bases: *ivadomed.transforms.ImedTransform*

Randomly dilate a ground-truth tensor.



Parameters **dilation_factor** (*float*) – Controls the number of dilation iterations. For each individual lesion, the number of dilation iterations is computed as follows:

nb_it = int(round(dilation_factor * sqrt(lesion_area)))

If dilation_factor <= 0, then no dilation will be performed.

__call__(*sample*, *metadata=None*)
Call self as a function.

__init__(*dilation_factor*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.transforms.**ElasticTransform**(*alpha_range*, *sigma_range*, *p=0.1*)
Bases: *ivadomed.transforms.ImedTransform*

Applies elastic transformation.

See also:

Simard, Patrice Y., David Steinkraus, and John C. Platt. "Best practices for convolutional neural networks applied to visual document analysis." Icdar. Vol. 3. No. 2003. 2003.

Parameters

- **alpha_range** (*tuple of floats*) – Deformation coefficient. Length equals 2.
- **sigma_range** (*tuple of floats*) – Standard deviation. Length equals 2.

__call__(*sample*, *metadata=None*)
Call self as a function.

__init__(*alpha_range*, *sigma_range*, *p=0.1*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.transforms.**HistogramClipping**(*min_percentile=5.0*, *max_percentile=95.0*)
Bases: *ivadomed.transforms.ImedTransform*

Performs intensity clipping based on percentiles.

Parameters

- **min_percentile** (*float*) – Between 0 and 100. Lower clipping limit.
- **max_percentile** (*float*) – Between 0 and 100. Higher clipping limit.

__call__(*sample*, *metadata=None*)
Call self as a function.

__init__(*min_percentile=5.0*, *max_percentile=95.0*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.transforms.**ImedTransform**
Bases: object

Base class for transforamtions.

__call__(*sample*, *metadata=None*)
Call self as a function.

class ivadomed.transforms.**NormalizeInstance**
Bases: *ivadomed.transforms.ImedTransform*

Normalize a tensor or an array image with mean and standard deviation estimated from the sample itself.

__call__(*sample*, *metadata=None*)
Call self as a function.

class ivadomed.transforms.**NumpyToTensor**
Bases: *ivadomed.transforms.ImedTransform*

Converts nd array to tensor object.

__call__(*sample*, *metadata=None*)
Converts nd array to Tensor.

undo_transform(*sample*, *metadata=None*)
Converts Tensor to nd array.

class ivadomed.transforms.**ROICrop**(*size*)
Bases: *ivadomed.transforms.Crop*

Make a crop of a specified size around a Region of Interest (ROI).

__call__(*sample*, *metadata=None*)
Call self as a function.

class ivadomed.transforms.**RandomAffine**(*degrees=0*, *translate=None*, *scale=None*)
Bases: *ivadomed.transforms.ImedTransform*

Apply Random Affine transformation.

Parameters

- **degrees** (*float*) – Positive float or list (or tuple) of length two. Angles in degrees. If only a float is provided, then rotation angle is selected within the range [-degrees, degrees]. Otherwise, the list / tuple defines this range.
- **translate** (*list of float*) – List of floats between 0 and 1, of length 2 or 3 depending on the sample shape (2D or 3D). These floats defines the maximum range of translation along each axis.
- **scale** (*list of float*) – List of floats between 0 and 1, of length 2 or 3 depending on the sample shape (2D or 3D). These floats defines the maximum range of scaling along each axis.

Attributes

- **degrees** (*tuple of floats*)
- **translate** (*list of float*)
- **scale** (*list of float*)

__call__(*sample*, *metadata=None*)
Call self as a function.

__init__(*degrees=0*, *translate=None*, *scale=None*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.transforms.**RandomReverse**
Bases: *ivadomed.transforms.ImedTransform*

Make a randomized symmetric inversion of the different values of each dimensions.

__call__(*sample*, *metadata=None*)
Call self as a function.

class ivadomed.transforms.**RandomShiftIntensity**(*shift_range*, *prob=0.1*)
Bases: *ivadomed.transforms.ImedTransform*

Add a random intensity offset.

Parameters

- **shift_range** (*tuple of floats*) – Tuple of length two. Specifies the range where the offset that is applied is randomly selected from.
- **prob** (*float*) – Between 0 and 1. Probability of occurence of this transformation.

__call__(*sample*, *metadata=None*)
Call self as a function.

__init__(*shift_range*, *prob=0.1*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.transforms.**Resample**(*hspace*, *wspace*, *dspace=1.0*)
Bases: *ivadomed.transforms.ImedTransform*

Resample image to a given resolution.

Parameters

- **hspace** (*float*) – Resolution along the first axis, in mm.
- **wspace** (*float*) – Resolution along the second axis, in mm.
- **dspace** (*float*) – Resolution along the third axis, in mm.
- **interpolation_order** (*int*) – Order of spline interpolation. Set to 0 for label data. Default=2.

__call__(*sample*, *metadata=None*)
Resample to a given resolution, in millimeters.

__init__(*hspace*, *wspace*, *dspace=1.0*)
Initialize self. See help(type(self)) for accurate signature.

undo_transform(*sample*, *metadata=None*)
Resample to original resolution.

class ivadomed.transforms.**UndoCompose**(*compose*)
Bases: object

Undo the Compose transformations.

Call the undo transformations in the inverse order than the "do transformations".

Attributes **compose** (*torchvision_transforms.Compose*)

Parameters **transforms** (*torchvision_transforms.Compose*) –

__call__ (*sample*, *metadata*, *data_type='gt'*)
Call self as a function.

__init__ (*compose*)
Initialize self. See help(type(self)) for accurate signature.

class ivadomed.transforms.**UndoTransform**(*transform*)
Bases: object

Call undo transformation.

Attributes **transform** (*ImedTransform*)

Parameters **transform** (ImedTransform) –

__call__ (*sample*)
Call self as a function.

__init__ (*transform*)
Initialize self. See help(type(self)) for accurate signature.

ivadomed.transforms.**apply_preprocessing_transforms**(*transforms*, *seg_pair*, *roi_pair=None*)
Applies preprocessing transforms to segmentation pair (input, gt and metadata).

Parameters

- **transforms** (Compose) – Preprocessing transforms.
- **seg_pair** (*dict*) – Segmentation pair containing input and gt.
- **roi_pair** (*dict*) – Segementation pair containing input and roi.

Returns Segmentation pair and roi pair.

Return type tuple

ivadomed.transforms.**get_preprocessing_transforms**(*transforms*)
Checks the transformations parameters and selects the transformations which are done during preprocessing only.

Parameters **transforms** (*dict*) – Transformation dictionary.

Returns Preprocessing transforms.

Return type dict

ivadomed.transforms.**get_subdatasets_transforms**(*transform_params*)
Get transformation parameters for each subdataset: training, validation and testing.

Parameters **transform_params** (*dict*) –

Returns Training, Validation and Testing transformations.

Return type dict, dict, dict

ivadomed.transforms.**multichannel_capable**(*wrapped*)
Decorator to make a given function compatible multichannel images.

Parameters **wrapped** – Given function.

Returns Functions' return.

ivadomed.transforms.**preprare_transforms**(*transform_dict*, *requires_undo=True*)

This function separates the preprocessing transforms from the others and generates the undo transforms related.

Parameters

- **transform_dict** (*dict*) – Dictionary containing the transforms and there parameters.
- **requires_undo** (*bool*) – Boolean indicating if transforms can be undone.

Returns

transform lst containing the preprocessing transforms and regular transforms, UndoCompose object containing the transform to undo.

Return type list, *UndoCompose*

ivadomed.transforms.**rescale_values_array**(*arr*, *minv=0.0*, *maxv=1.0*, *dtype=<class 'numpy.float32'>*)

Rescale the values of numpy array *arr* to be from *minv* to *maxv*.

Parameters

- **arr** (*ndarry*) – Array whose values will be rescaled.
- **minv** (*float*) – Minimum value of the output array.
- **maxv** (*float*) – Maximum value of the output array.
- **dtype** (*type*) – Cast array to this type before performing the rescaling.

ivadomed.transforms.**two_dim_compatible**(*wrapped*)

Decorator to make a given function compatible 2D or 3D images.

Parameters **wrapped** – Given function.

Returns Functions' return.

10.11 Utils API

class ivadomed.utils.**HookBasedFeatureExtractor**(*submodule*, *layername*, *upscale=False*)

Bases: torch.nn.modules.module.Module

This function extracts feature maps from given layer. Helpful to observe where the attention of the network is focused.

https://github.com/ozan-oktay/Attention-Gated-Networks/tree/a96edb72622274f6705097d70cfaa7f2bf818a5a

Parameters

- **submodule** (*nn.Module*) – Trained model.
- **layername** (*str*) – Name of the layer where features need to be extracted (layer of interest).
- **upscale** (*bool*) – If True output is rescaled to initial size.

Attributes

- **submodule** (*nn.Module*) – Trained model.
- **layername** (*str*) – Name of the layer where features need to be extracted (layer of interest).
- **outputs_size** (*list*) – List of output sizes.
- **outputs** (*list*) – List of outputs containing the features of the given layer.

- **inputs** (*list*) – List of inputs.
- **inputs_size** (*list*) – List of input sizes.

__init__(*submodule*, *layername*, *upscale=False*)
Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

class ivadomed.utils.**SliceFilter**(*filter_empty_mask=True*, *filter_empty_input=True*)
Bases: `object`

Filter 2D slices from dataset.

If a sample does not meet certain conditions, it is discarded from the dataset.

Parameters

- **filter_empty_mask** (*bool*) – If True, samples where all voxel labels are zeros are discarded.
- **filter_empty_input** (*bool*) – If True, samples where all voxel intensities are zeros are discarded.

Attributes

- **filter_empty_mask** (*bool*) – If True, samples where all voxel labels are zeros are discarded.
- **filter_empty_input** (*bool*) – If True, samples where all voxel intensities are zeros are discarded.

__call__(*sample*)
Call self as a function.

__init__(*filter_empty_mask=True*, *filter_empty_input=True*)
Initialize self. See help(type(self)) for accurate signature.

ivadomed.utils.**combine_predictions**(*fname_lst*, *fname_hard*, *fname_prob*, *thr=0.5*)
Combine predictions from Monte Carlo simulations.

Combine predictions from Monte Carlo simulations and save the resulting as:

(1) *fname_prob*, a soft segmentation obtained by averaging the Monte Carlo samples.

(2) *fname_hard*, a hard segmentation obtained thresholding with *thr*.

Parameters

- **fname_lst** (*list of str*) – List of the Monte Carlo samples.
- **fname_hard** (*str*) – Filename for the output hard segmentation.
- **fname_prob** (*str*) – Filename for the output soft segmentation.
- **thr** (*float*) – Between 0 and 1. Used to threshold the soft segmentation and generate the hard segmentation.

ivadomed.utils.**convert_labels_to_RGB**(*grid_img*)
Converts 2D images to RGB encoded images for display in tensorboard.

Parameters **grid_img** (*Tensor*) – GT or prediction tensor with dimensions (batch size, number of classes, height, width).

Returns RGB image with shape (height, width, 3).

Return type tensor

ivadomed.utils.**cuda**(*input_var*, *cuda_available=True*, *non_blocking=False*)
Passes input_var to GPU.

Parameters

- **input_var** (*Tensor*) – either a tensor or a list of tensors.
- **cuda_available** (*bool*) – If False, then return identity
- **non_blocking** (*bool*) –

Returns Tensor

ivadomed.utils.**define_device**(*gpu_id*)
Define the device used for the process of interest.

Parameters **gpu_id** (*int*) – GPU ID.

Returns True if cuda is available.

Return type Bool, device

ivadomed.utils.**display_selected_model_spec**(*params*)
Display in terminal the selected model and its parameters.

Parameters **params** (*dict*) – Keys are param names and values are param values.

ivadomed.utils.**display_selected_transfoms**(*params*, *dataset_type*)
Display in terminal the selected transforms for a given dataset.

Parameters

- **params** (*dict*) –
- **dataset_type** (*list*) – e.g. ['testing'] or ['training', 'validation']

ivadomed.utils.**mixup**(*data*, *targets*, *alpha*, *debugging=False*, *ofolder=None*)
Compute the mixup data.

See also:

Zhang, Hongyi, et al. "mixup: Beyond empirical risk minimization." arXiv preprint arXiv:1710.09412 (2017).

Parameters

- **data** (*Tensor*) – Input images.
- **targets** (*Tensor*) – Input masks.
- **alpha** (*float*) – MixUp parameter.
- **debugging** (*Bool*) – If True, then samples of mixup are saved as png files.
- **ofolder** (*str*) – If debugging, output folder where "mixup" folder is created and samples are saved.

Returns Mixed image, Mixed mask.

Return type Tensor, Tensor

ivadomed.utils.**onnx_inference**(*model_path*, *inputs*)
Run ONNX inference

Parameters

- **model_path** (*str*) – Path to the ONNX model.
- **inputs** (*Tensor*) – Batch of input image.

Returns Network output.

Return type Tensor

ivadomed.utils.**plot_transformed_sample**(*before*, *after*, *list_title=[]*, *fname_out=''*, *cmap='jet'*)
Utils tool to plot sample before and after transform, for debugging.

Parameters

- **before** (*ndarray*) – Sample before transform.
- **after** (*ndarray*) – Sample after transform.
- **list_title** (*list of str*) – Sub titles of before and after, resp.
- **fname_out** (*str*) – Output filename where the plot is saved if provided.
- **cmap** (*str*) – Matplotlib colour map.

ivadomed.utils.**pred_to_nib**(*data_lst*, *z_lst*, *fname_ref*, *fname_out*, *slice_axis*, *debug=False*, *kernel_dim='2d'*, *bin_thr=0.5*, *discard_noise=True*)
Save the network predictions as nibabel object.

Based on the header of *fname_ref* image, it creates a nibabel object from the Network predictions (*data_lst*).

Parameters

- **data_lst** (*list of np arrays*) – Predictions, either 2D slices either 3D patches.
- **z_lst** (*list of ints*) – Slice indexes to reconstruct a 3D volume for 2D slices.
- **fname_ref** (*str*) – Filename of the input image: its header is copied to the output nibabel object.
- **fname_out** (*str*) – If not None, then the generated nibabel object is saved with this filename.
- **slice_axis** (*int*) – Indicates the axis used for the 2D slice extraction: Sagittal: 0, Coronal: 1, Axial: 2.
- **debug** (*bool*) – If True, extended verbosity and intermediate outputs.
- **kernel_dim** (*str*) – Indicates whether the predictions were done on 2D or 3D patches. Choices: '2d', '3d'.
- **bin_thr** (*float*) – If positive, then the segmentation is binarized with this given threshold. Otherwise, a soft segmentation is output.
- **discard_noise** (*bool*) – If True, predictions that are lower than 0.01 are set to zero.

Returns Object containing the Network prediction.

Return type NibabelObject

ivadomed.utils.**reorient_image**(*arr*, *slice_axis*, *nib_ref*, *nib_ref_canonical*)
Reorient an image to match a reference image orientation.

It reorients a array to a given orientation and convert it to a nibabel object using the reference nibabel header.

Parameters

- **arr** (*ndarray*) – Input array, array to re orient.
- **slice_axis** (*int*) – Indicates the axis used for the 2D slice extraction: Sagittal: 0, Coronal: 1, Axial: 2.
- **nib_ref** (*nibabel*) – Reference nibabel object, whose header is used.
- **nib_ref_canonical** (*nibabel*) – *nib_ref* that has been reoriented to canonical orientation (RAS).

ivadomed.utils.**run_uncertainty**(*ifolder*)
Compute uncertainty from model prediction.

This function loops across the model predictions (nifti masks) and estimates the uncertainty from the Monte Carlo samples. Both voxel-wise and structure-wise uncertainty are estimates.

Parameters **ifolder** (*str*) – Folder containing the Monte Carlo samples.

ivadomed.utils.**save_color_labels**(*gt_data*, *binarize*, *gt_filename*, *output_filename*, *slice_axis*)
Saves labels encoded in RGB in specified output file.

Parameters

- **gt_data** (*ndarray*) – Input image with dimensions (Number of classes, height, width, depth).
- **binarize** (*bool*) – If True binarizes gt_data to 0 and 1 values, else soft values are kept.
- **gt_filename** (*str*) – GT path and filename.
- **output_filename** (*str*) – Name of the output file where the colored labels are saved.
- **slice_axis** (*int*) – Indicates the axis used to extract slices: "axial": 2, "sagittal": 0, "coronal": 1.

Returns RGB labels.

Return type ndarray

ivadomed.utils.**save_feature_map**(*batch*, *layer_name*, *log_directory*, *model*, *test_input*, *slice_axis*)
Save model feature maps.

Parameters

- **batch** (*dict*) –
- **layer_name** (*str*) –
- **log_directory** (*str*) – Output folder.
- **model** (*nn.Module*) – Network.
- **test_input** (*Tensor*) –
- **slice_axis** (*int*) – Indicates the axis used for the 2D slice extraction: Sagittal: 0, Coronal: 1, Axial: 2.

ivadomed.utils.**save_mixup_sample**(*ofolder*, *input_data*, *labeled_data*, *lambda_tensor*)
Save mixup samples as png files in a "mixup" folder.

Parameters

- **ofolder** (*str*) – Output folder where "mixup" folder is created and samples are saved.
- **input_data** (*Tensor*) – Input image.
- **labeled_data** (*Tensor*) – Input masks.
- **lambda_tensor** (*Tensor*) –

ivadomed.utils.**save_onnx_model**(*model*, *inputs*, *model_path*)

Convert PyTorch model to ONNX model and save it as *model_path*.

Parameters

- **model** (*nn.Module*) – PyTorch model.
- **inputs** (*Tensor*) – Tensor, used to inform shape and axes.
- **model_path** (*str*) – Output filename for the ONNX model.

ivadomed.utils.**save_tensorboard_img**(*writer*, *epoch*, *dataset_type*, *input_samples*, *gt_samples*, *preds*, *is_three_dim=False*)

Saves input images, gt and predictions in tensorboard.

Parameters

- **writer** (*SummaryWriter*) – Tensorboard's summary writer.
- **epoch** (*int*) – Epoch number.
- **dataset_type** (*str*) – Choice between Training or Validation.
- **input_samples** (*Tensor*) – Input images with shape (batch size, number of channel, height, width, depth) if 3D else (batch size, number of channel, height, width)
- **gt_samples** (*Tensor*) – GT images with shape (batch size, number of channel, height, width, depth) if 3D else (batch size, number of channel, height, width)
- **preds** (*Tensor*) – Model's prediction with shape (batch size, number of channel, height, width, depth) if 3D else (batch size, number of channel, height, width)
- **is_three_dim** (*bool*) – True if 3D input, else False.

ivadomed.utils.**segment_volume**(*folder_model*, *fname_image*, *fname_prior=None*, *gpu_number=0*)

Segment an image.

Segment an image (*fname_image*) using a pre-trained model (*folder_model*). If provided, a region of interest (*fname_roi*) is used to crop the image prior to segment it.

Parameters

- **folder_model** (*str*) – foldername which contains (1) the model ('folder_model/folder_model.pt') to use (2) its configuration file ('folder_model/folder_model.json') used for the training,

  see https://github.com/neuropoly/ivadomed/wiki/configuration-file
- **fname_image** (*str*) – image filename (e.g. .nii.gz) to segment.
- **fname_prior** (*str*) – Image filename (e.g. .nii.gz) containing processing information (e.i. spinal cord segmentation, spinal location or MS lesion classification)

  e.g. spinal cord centerline, used to crop the image prior to segment it if provided. The segmentation is not performed on the slices that are empty in this image.
- **gpu_number** (*int*) – Number representing gpu number if available.

Returns Object containing the soft segmentation.

Return type nibabelObject

ivadomed.utils.**structurewise_uncertainty**(*fname_lst*, *fname_hard*, *fname_unc_vox*, *fname_out*)

Estimate structure wise uncertainty.

Structure-wise uncertainty from N MC probability maps (*fname_lst*) and saved in *fname_out* with the following suffixes:

- '-cv.nii.gz': coefficient of variation
- '-iou.nii.gz': intersection over union
- '-avgUnc.nii.gz': average voxel-wise uncertainty within the structure.

Parameters

- **fname_lst** (*list of str*) – List of the Monte Carlo samples.
- **fname_hard** (*str*) – Filename of the hard segmentation, which is used to compute the *avgUnc* by providing a mask of the structures.
- **fname_unc_vox** (*str*) – Filename of the voxel-wise uncertainty, which is used to compute the *avgUnc*.
- **fname_out** (*str*) – Output filename.

ivadomed.utils.**unstack_tensors**(*sample*)

Unstack tensors.

Parameters **sample** (*Tensor*) –

Returns list of Tensors.

Return type list

ivadomed.utils.**volume_reconstruction**(*batch*, *pred*, *undo_transforms*, *smp_idx*, *volume=None*, *weight_matrix=None*)

Reconstructs volume prediction from subvolumes used during training :param batch: Dictionary containing input, gt and metadata :type batch: dict :param pred: Subvolume prediction :type pred: tensor :param undo_transforms: Undo transforms so prediction match original image resolution and shap :type undo_transforms: UndoCompose :param smp_idx: Batch index :type smp_idx: int :param volume: Reconstructed volume :type volume: tensor :param weight_matrix: Weights containing the number of predictions for each voxel :type weight_matrix: tensor

Returns undone subvolume, metadata, boolean representing if its the last sample to process, reconstructed volume, weight matrix

Return type tensor, dict, bool, tensor, tensor

ivadomed.utils.**voxelwise_uncertainty**(*fname_lst*, *fname_out*, *eps=1e-05*)

Estimate voxel wise uncertainty.

Voxel-wise uncertainty is estimated as entropy over all N MC probability maps, and saved in *fname_out*.

Parameters

- **fname_lst** (*list of str*) – List of the Monte Carlo samples.
- **fname_out** (*str*) – Output filename.
- **eps** (*float*) – Epsilon value to deal with np.log(0).

CHAPTER 11

Contributors

This project results from a collaboration between the NeuroPoly Lab and Mila.

A list of contributors is available here.

CHAPTER 12

Sponsors

If you wish to sponsor this project, please consider donating.

CHAPTER 13

License

MIT License

Python Module Index

i

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V