mdgru Documentation

Release 0.2

Simon Andermatt

Nov 25, 2019

Guides

This repository contains the code used to generate the result in the paper Automated Segmentation of Multiple Sclerosis Lesions using Multi-Dimensional Gated Recurrent Units. It is implemented in **Python** using the deep learning libraries **PyTorch** and **TensorFlow** each, modified versions were also used to reach *1st place* in the ISBI 2015 longitudinal lesion segmentation challenge, *2nd place* in the white matter hyperintensities challenge of MICCAI 2017 (and its previous implementation using Caffe made *3rd place* in the MrBrainS13 Segmentation challenge). It was also applied in the BraTS 2017 competition, where the information on the exact rank are still unknown.

Since being published the first time using a Caffe implementation, the code has been improved on quite a bit, especially to facilitate handling training and testing runs. The reported results should still be reproducible though using this new implementation with TensorFlow and PyTorch. (The former Caffe code is not maintained anymore (there are probably breaking changes in CuDNN, not tested), but a snapshot of it is included in this release in the folder tensorflow_extra_ops as additional operation for TensorFlow.)

The code has been developed in Python==3.5.2. It is best to set up a **virtual environment** (e.g. with conda) with the mentioned properties in order to develop the deep learning model. For this purpose, follow the instructions in the docs, and install mdgru (together with mvloader) using pip. In addition, make sure you have CUDA/cuDNN installed.

```
pip install git+https://github.com/gtancev/MD-GRU.git
pip install git+https://github.com/spezold/mvloader.git
```

CHAPTER 1

Papers

Reference implementation (and based on former Caffe version):

```
@inproceedings{andermatt2016multi,
  title={Multi-dimensional gated recurrent units for the segmentation of biomedical 3D-data},
  author={Andermatt, Simon and Pezold, Simon and Cattin, Philippe},
  booktitle={International Workshop on Large-Scale Annotation of Biomedical Data and Expert Label Synthesis},
  pages={142--151},
  year={2016},
  organization={Springer}
}
```

Code also used for (with modifications):

```
@inproceedings{andermatt2017a,
  title = {{{Automated Segmentation of Multiple Sclerosis Lesions}} using {{Multi-Dimensional Gated Recurrent Units}}},
  timestamp = {2017-08-09T07:27:10Z},
  journal = {Lecture Notes in Computer Science},
  author = {Andermatt, Simon and Pezold, Simon and Cattin, Philippe},
  year = {2017},
  booktitle={International Workshop on Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries},
  note = {{{[accepted]}}},
  organization={Springer}
}

@article{andermatt2017b,
  title={Multi-dimensional Gated Recurrent Units for Automated Anatomical Landmark Localization},
  author={Andermatt, Simon and Pezold, Simon and Amann, Michael and Cattin, Philippe C},
  journal={arXiv preprint arXiv:1708.02766},
```

(continues on next page)

(continued from previous page)

```
  year={2017}
}

@article{andermatt2017wmh,
  title={Multi-dimensional Gated Recurrent Units for the Segmentation of White Matter␣
↪Hyperintensites},
  author={Andermatt, Simon and Pezold, Simon and Cattin, Philippe}
}

@inproceedings{andermatt2017brats,
title = {Multi-dimensional Gated Recurrent Units for
Brain Tumor Segmentation},
author = {Simon Andermatt and Simon Pezold and Philippe C. Cattin},
year = 2017,
booktitle = {2017 International {{MICCAI}} BraTS Challenge}
}
```

When using this code, please cite at least *andermatt2016multi*, since it is the foundation of this work. Furthermore, feel free to cite the publication matching your use-case from above. E.g. if you're using the code for pathology segmentation, it would be adequate to cite *andermatt2017a* as well.

CHAPTER 2

Acknowledgements

We thank the Medical Image Analysis Center for funding this work.

2.1 How to install

The code has been developed in Python==3.5.2. It is best to set up a **virtual environment** (e.g. with conda) with the mentioned properties in order to develop the deep learning model. For this purpose, install mdgru (together with mvloader) using pip. In addition, make sure you have CUDA/cuDNN installed.

```
pip install git+https://github.com/gtancev/MD-GRU.git
pip install git+https://github.com/spezold/mvloader.git
```

2.2 How to use (Tensorflow backend)

The file *RUN_mdgru.py* is used for basically all segmentation tasks. For now, please refer to it's help message by calling *python3 RUN_mdgru.py* and the documentation in the code.

As the RUN_mdgru.py file contains a overly large number of parameters, a sample train & test, individual train, and individual test run are detailed in the following:

First, the data have to be prepared and have to have a certain format. Each sample should be contained in one folder, with the label and feature (e.g. different sequences) files consistently named after a certain scheme. Furthermore, all the samples belonging to test, train and validation set should be located in respective folders. The following shows an example, where we have training, testing and validation folders train_data, test_data and val_data respectively, containing each some samples. **Each sample consists of two featurefiles in this case (seq1.nii(.gz) and seq2.nii(.gz), e.g. t2.nii(.gz), flair.nii(.gz), . . .) and one labelfile (lab.nii(.gz), e.g. mask1.nii(.gz)), as shown in the following example.**

```
path/to/samplestructure
├── test_data
│   ├── SAMPLE_27535
│   │   ├── lab.nii.gz
│   │   ├── seq1.nii.gz
│   │   └── seq2.nii.gz
│   └── SAMPLE_6971
│       ├── lab.nii.gz
│       ├── seq1.nii.gz
│       └── seq2.nii.gz
├── train_data
│   ├── SAMPLE_11571
│   │   ├── lab.nii.gz
│   │   ├── seq1.nii.gz
│   │   └── seq2.nii.gz
│   ├── SAMPLE_13289
│   │   ├── lab.nii.gz
│   │   ├── seq1.nii.gz
│   │   └── seq2.nii.gz
│   ├── SAMPLE_16158
│   │   ├── lab.nii.gz
│   │   ├── seq1.nii.gz
│   │   └── seq2.nii.gz
│   ├── SAMPLE_18429
│   │   ├── lab.nii.gz
│   │   ├── seq1.nii.gz
│   │   └── seq2.nii.gz
│   ├── SAMPLE_19438
│   │   ├── lab.nii.gz
│   │   ├── seq1.nii.gz
│   │   └── seq2.nii.gz
│   └── SAMPLE_2458
│       ├── lab.nii.gz
│       ├── seq1.nii.gz
│       └── seq2.nii.gz
└── val_data
    ├── SAMPLE_26639
    │   ├── lab.nii.gz
    │   ├── seq1.nii.gz
    │   └── seq2.nii.gz
    └── SAMPLE_27319
        ├── lab.nii.gz
        ├── seq1.nii.gz
        └── seq2.nii.gz
```

The label files need to be consistent with increasing class numbers. Eg. if we model the background, white matter, gray matter and csf for instance, we have 4 classes and hence distribute them to the numbers 0, 1, 2 and 3. Furthermore, the label files should also be encoded as integer files (e.g. nifti uint8), and the **feature and label files need to have matching dimensions**.

2.2.1 Train & test

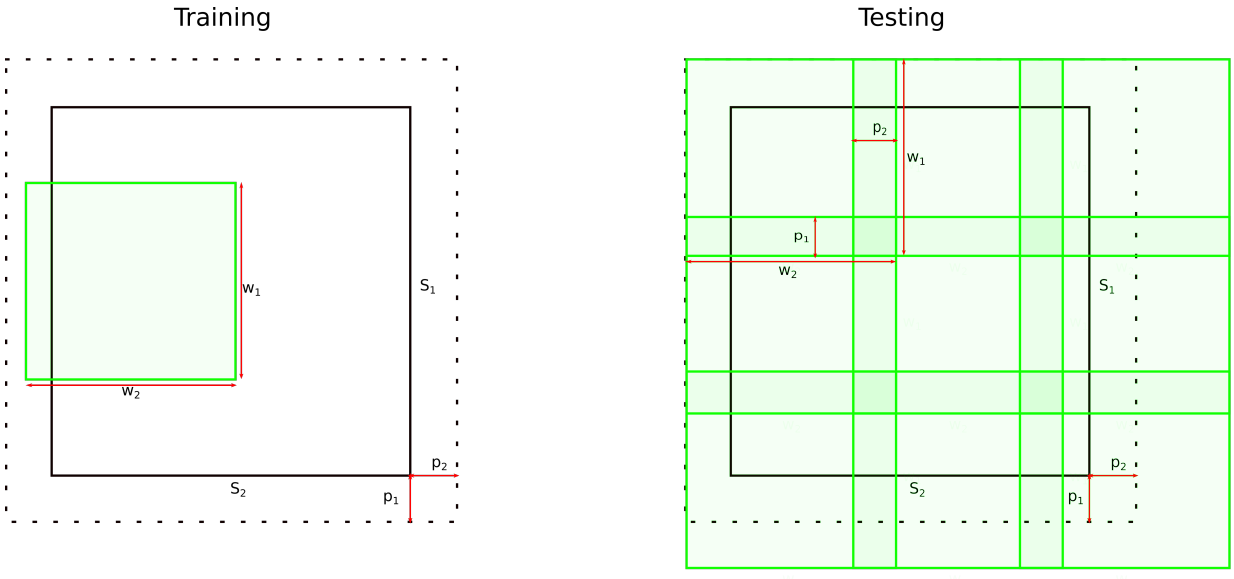
In the following, we show the case, where we train a model on the above data and also immediately evaluate our model on the last training state (rarely a good idea in general) to explain the individual parameters:

```
python3 RUN_mdgru.py --datapath path/to/samplestructure --locationtraining train_data␣
↪\
--locationvalidation val_data --locationtesting test_data \
--optionname defaultsettings --modelname mdgrudef48 -w 64 64 64 -p 5 5 5 \
-f seq1.nii.gz seq2.nii.gz -m lab.nii.gz --iterations 100000 \
--nclasses 4 --num_threads 4
```

Read more about the start script options by selecting **start script** in the navigation bar. The above first four parameters tell the script, where our different data can be found. Furthermore, it will create a folder experiments in "path/to/samplestructure". Inside this experiments folder, a folder for the current setting is created. The name of this folder can be determined with "–optionname". For each individual train/test/train&test run, a folder with logging data is created using the latest timestamp in seconds inside this settings folder. Any log data for the experiment can then in turn be found inside the cache subfolder. (e.g. /path/to/samplestructure/defaultsettings/1524126169/cache). Inside this cache folder, there will be a log file, logging all relevant information to the current run, all validation files will be saved here as well as the checkpoints and tensorboard data.

Expecially for 2-D data - and if a large number of samples is available - the whole image can be processed. There, we set the subvolume (patchsize) parameter to the size of the images (-w 200 200 200) and the padding parameters to 0. This has the effect that we only sample inside the image with a padding of 0 (-p 0 0 0), and hence just take the full image. As current hardware can rarely support the full volume for volumetric data though, a subvolume needs to be specified. Imagine we are using volumetric data with dimensions 256x256x192. Since this will not fit, we decide to sample patches of 64^3, and hence set the subvolume parameter -w to 64 64 64. Furthermore, we decide that we do want to sample a bit outside of the full volume as well, as interesting data is close to the border. we hence set the -p parameter to 5 5 5, allowing for a random sampling of patches of 5 voxels outside along each axis of the full volume. During testing, patches are sampled from a regular grid to fully cover the full volume (or image). There, the p parameter is used to also specify the amount of overlap of the patches. In our example, we would only specify an overlap of 5 voxels along each dimension.

The following image shows the influence of the w and p parameters when sampling images during the training and testing phase:



The remaining options given above are the –modelname, which is a optional, userspecified name for the model we

are creating in the tensorflow graph. -f and -m specify feature and mask files to be used. –nclasses specifies how many classes are in the label files (e.g. 4 for background, white matter, grey matter and csf). –iterations specifies the maximum number of iterations to train. If we cancel the training process at any time, the current state is saved in a checkpoint called *interrupt*. Finally, –num_threads is a parameter which defines how many threads should be used to load data concurrently. This can initially be set to a low value such as 4. If during training, in the log file or stdout on the console, values larger than 0.1 seconds are used for "io", it might be advisable to increase this value, as valuable time is wasted on waiting for the data loading routine.

2.2.2 Only train

Usually, we want to use the validation set to determine, which state of the network works best for our data and then evaluate our testset on that data. We can do this by using the following command:

```
python3 RUN_mdgru.py --datapath path/to/samplestructure --locationtraining train_data␣
↪\
--locationvalidation val_data \
--optionname onlytrainrun --modelname mdgrudef48 -w 64 64 64 -p 5 5 5 \
-f seq1.nii.gz seq2.nii.gz -m lab.nii.gz --iterations 100000 \
--nclasses 4 --num_threads 4 --only_train
```

In this setup, we can omit the '–locationtesting' and append '–only_train' in its place, to specify, that we want to stop the procedure after the training process.

Furthermore, it is in most cases advisable to use a certain amount of data augmentation, since rarely enough labelled training data is available. For this, the following set of parameters can be optionally added for the training procedure:

```
--rotate ANGLE --scale scale1 scale2...
--deformation gridspacing1 gridspacing2...
--deformSigma samplingstdev1 samplingstdev2...
```

The first parameter is a scalar in radians which allows for random rotation around a random vector for 3d data, and around the center point for 2d data between [-ANGLE,+ANGLE] radians. The parameter is sampled uniformly. The scaling parameter allows for random scaling between [1/scale,scale], where we sample form an exponential distribution and each axis has its own scaling parameter. The last two parameters have to be used together and specify a random deformation grid which is applied to the subvolumes. The first parameters specify the grid spacing, and the second set of parameters the standard deviation of a zero mean Gaussian which is used at each grid point to sample a random vector. This low resolution grid is then interpolated quadratically and used to deform the sampling of the subvolumes or patches.

2.2.3 Only test

```
python3 RUN_mdgru.py --datapath path/to/samplestructure --locationtraining train_data␣
↪\
--locationtesting test_data\
--optionname defaultsettings --modelname mdgrudef48 -w 64 64 64 -p 5 5 5 \
-f seq1.nii.gz seq2.nii.gz -m lab.nii.gz \
--nclasses 4 --only_test --ckpt path/to/samplestructure/experiments/onlytrainrun/
↪1524126169/cache/temp-22500 --notestingmask
```

Usually, after conducting a training run, it is the best idea to simply copy the training parameters, remove the "only_test", add the locationtesting and the checkpointfile with "–ckpt". Some other parameters can also be left out as shown above, since they do not have an impact on the testing process. The training process before, when completed, creates at the specified saving interval checkpoint files, which are named temp-$i, where $i is the iteration number, if no epochs are specified or temp-epoch$epoch-$i otherwise. On the file system, the files also have appendices like

".data-00000-of-00001" or ".meta" or ".index", but these can be ignored and should not be specified when specifying a checkpoint. **After the whole training procedure, a final checkpoint is created, which saves the final state of the network.** If the training process is interrupted, a "interrupt-$i" checkpoint is created, where $i is again the iteration number. All of these three types of checkpoints can be used to evaluate the model. During testing, the optionname also defines the name of the probability maps that are saved in the test_data sample folders as results. If multiple checkpoints are used for evaluation, either none, one or the same number of optionnames can be provided. Finally, –notestingmask has to be used, if for the testing samples, no mask files are available. Otherwise, it will not find testing samples, as it uses the mask file as a requirement for each folder to be accepted as valid sample. If there are labelmaps for the test samples, this flag can be omitted, leading to an automatic evaluation using predefined metrics during the evaluation.

2.2.4 Fine tuning

In order to fine tune the model and increase the performance even more, several data augmentation options are available, and these can be found in the **data loader module**. In addition, dice loss is implemented for cases with skewed classes, e.g. when only few pixels in a subvolume are part of the positive class such as in tumor or lesion segmentation. Read more about dice loss in the **start script** options.

2.2.5 Usage on a high performance computing (HPC) cluster

When using the code on a HPC cluster, make sure to set the GPU IDs. Copy the RUN_mdgru.py script to your working directory. The slurm submission file should look like this:

```
#!/bin/bash

#SBATCH --job-name=mdgru
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=8G
#Total memory reserved: 8GB
#SBATCH --partition=pascal      # pascal / titanx
#SBATCH --gres=gpu:6            # --gres=gpu:2 for two GPU, aso.

#SBATCH --time=00:30:00
#SBATCH --qos=30min

# Paths to STDOUT or STDERR files should be absolute or relative to current working␣
↪directory
#SBATCH --output=stdout
#SBATCH --mail-type=END,FAIL,TIME_LIMIT
#SBATCH --mail-user=your.email@adress.com

#This job runs from the current working directory

#Remember:
#The variable $TMPDIR points to the local hard disks in the computing nodes.
#The variable $HOME points to your home directory.
#The variable $JOB_ID stores the ID number of your task.

#load your required modules below
################################
ml Python/3.5.2-goolf-1.7.20
ml CUDA/9.0.176
ml cuDNN/7.3.1.20-CUDA-9.0.176
```

(continues on next page)

(continued from previous page)

```
#export your required environment variables below
################################################
source "/pathtoyourfolderbeforeanaconda3/anaconda3/bin/activate"␣
↪nameofvirtualenvironment

#add your command lines below
#############################
python3 RUN_mdgru.py --datapath files --locationtraining train \
--locationvalidation val --locationtesting test \
--optionname defaultsettings --modelname mdgrudef48 -w 64 64 64 -p 5 5 5 \
-f pd_pp.nii t2_pp.nii flair_pp.nii mprage_pp.nii -m mask.nii --iterations 10000 \
--nclasses 2 --num_threads 4  --gpu 0
```

2.2.6 Localization code

The code for the landmark localization task is also included in this release except for an appropriate *RUN*-file. Since it would need some code updates due to recent changes in the code, it has not been included. If you're anyhow interested in the localization code, please get in touch, and I could provide you with the (now outdated) *RUN*-files we used and information on what needs to be updated to make it work again.

2.3 Start script

Below is a sample start script with minimum inputs.

```
python3 RUN_mdgru.py --datapath path/to/samplestructure --locationtraining train_data␣
↪\
--locationvalidation val_data --locationtesting test_data \
--optionname bestsettingsever --modelname bestmodelever -w 64 64 64 -p 5 5 5 \
-f t2.nii.gz flair.nii.gz -m mask1.nii.gz --iterations 100000 \
--nclasses 2
```

2.3.1 Mandatory inputs

These are mandatory inputs for which no default values are set.

path to folders with train/val/test data (dtype=str)

```
--datapath path
```

name of folder with data for respective purpose (dtype=str)

```
--locationtraining foldernametrain --locationvalidation foldernameval --
↪locationtesting foldernametest
```

name of options/settings (dtype=str)

```
--optionname bestoptionsever
```

name of model (dtype=str)

```
--modelname bestmodelever
```

subvolume size (dtype=int) of shape (1, n_dims)

```
-w 64 64 64
```

padding size (dtype=int) of shape (1, n_dims)

```
-p 5 5 5
```

sequences to include (dtype=str)

```
-f t2.nii flair.nii
```

masks to include (dtype=str)

```
-m mask1.nii
```

iterations of training to perform (dtype=int); only possible if –epochs 1

```
--interations 100000
```

number of classes (dtype=int)

```
--nclasses 2
```

2.3.2 Optional inputs

These are optional inputs for which the default values (listed in the commands below) can be changed manually. In the data loader module, you can learn about additional optional data augmentation methods.

epochs to perform (dtype=int); does not work together with iterations

```
--epochs 1
```

number of threads in data collection for data prefetching (dtype=int)

```
--num_threads 3
```

use non-threaded data collection (dtype=bool)

```
--nonthreaded
```

use PyTorch version (dtype=bool)

```
--use_pytorch
```

set GPU IDs for usage, e.g. using GPUs with ID 0 (in total using 1 GPU) (dtype=int)

```
--gpu 0
```

use only CPU (dtype=bool)

```
--only_cpu
```

fraction of GPU memory to use (dtype=float) in [0,1]

```
--gpubound 1.0
```

probability for dropout (dtype=float) in [0,1]

```
--dropout_rate 0.5
```

use batch normalization (dtype=bool)

```
--add_e_bn False
```

use skip connections/residual learning; add a residual connection around a MDGRU block (dtype=bool)

```
--resmdgru False
```

perform only training without testing (dtype=bool)

```
--only_train
```

perform only testing without training, make sure to add a checkpoint (dtype=bool)

```
--only_test --ckpt path/to/checkpoint/file
```

application of dice loss to label i, by default turned off (dtype=int)

```
--dice_loss_label 0 1
```

weights for dice loss, if label(s) are provided (dtype=float)

```
--dice_loss_weight 0.0
```

weights for dice loss, automatized instead of manual (dtype=bool)

```
--dice_autoweighted # weights the label dices with the squared inverse gold standard
↪area/volume - specify which labels with dice_loss_label, sum(dice_loss_weight) is
↪used as a weighting between crossentropy and diceloss
--dice_generalized # total intersections of all labels over total sums of all labels,
↪instead of linearly combined class dices
--dice_cc # dice loss for binary segmentation per true component
```

2.4 Model (Tensorflow backend)

These are optional controllable parameters for the CGRUnits inside a MDGRU block.

periodic convolution on input x (dtype=bool)

```
--periodic_convolution_x False
```

periodic convolution on input h (dtype=bool)

```
--periodic_convolution_h False
```

use Bernoulli distribution (insted of Gaussian) for dropconnect (dtype=bool)

```
--use_bernoulli False
```

stride (dtype=int)

```
--strides None
```

use dropconnect on input x (dtype=bool)

```
--use_dropconnect_x True
```

use dropconnect on input h (dtype=bool)

```
--use_dropconnect_h True
```

don't use average pooling (dtype=bool)

```
--no_avg_pooling True
```

filter size for input x (dtype=int)

```
--filter_size_x 7
```

filter size for input h (dtype=int)

```
--filter_size_h 7
```

use static RNN (dtype=bool)

```
--use_static_rnn False
```

add batch normalization at the input x in gate (dtype=bool)

```
--add_x_bn False
```

add batch normalization at the input h in candidate (dtype=bool)

```
--add_h_bn False
```

add batch normalization at the candidates input and state (dtype=bool)

```
--add_e_bn False
```

add residual learning to the input x of each cgru (dtype=bool)

```
--resgrux False
```

add residual learning to the input h of each cgru (dtype=bool)

```
--resgruh False
```

move the reset gate to the location the original GRU applies it at (dtype=bool)

```
--put_r_back False
```

apply dropconnect on the candidate weights as well (dtype=bool)

```
--use_dropconnect_on_state False
```

2.4.1 Subpackages

2.4.1.1 Convolutional RNN

2.4.1.1.1 CGRU

2.4.1.1.2 Module contents

2.4.1.2 MDRNN

2.4.1.2.1 MDGRU

2.4.1.2.2 Module contents

2.4.2 MDGRUClassification

2.4.3 Module contents

2.5 Model (PyTorch backend)

2.5.1 Subpackages

2.5.1.1 Convolutional GRU

2.5.1.1.1 CGRU

These are optional parameters for the CGRUnits inside a MDGRU block.

periodic convolution on input x (dtype=bool)

```
--periodic_convolution_x False
```

periodic convolution on input h (dtype=bool)

```
--periodic_convolution_h False
```

use Bernoulli distribution (insted of Gaussian) for dropconnect (dtype=bool)

```
--use_bernoulli False
```

use dropconnect on input x (dtype=bool)

```
--dropconnectx False
```

use dropconnect on input h (dtype=bool)

```
--dropconnecth False
```

add batch normalization at the input x in gate (dtype=bool)

```
--add_x_bn
```

add batch normalization at the input h in candidate (dtype=bool)

```
--add_h_bn False
```

add batch normalization at the candidates input and state (dtype=bool)

```
--add_a_bn False
```

add residual learning to the input x of each cgru (dtype=bool)

```
--resgrux False
```

add residual learning to the input h of each cgru (dtype=bool)

```
--resgruh False
```

move the reset gate to the location the original GRU applies it at (dtype=bool)

```
--put_r_back False
```

apply dropconnect on the candidate weights as well (dtype=bool)

```
--use_dropconnect_on_state False
```

2.5.1.1.2 Module contents

2.5.1.2 Multi-dimensional GRU

2.5.1.2.1 MDGRU

These are optional model inputs.

stride (dtype=int)

```
--strides None
```

use dropconnect on input x (dtype=bool)

```
--use_dropconnect_x True
```

use dropconnect on input h (dtype=bool)

```
--use_dropconnect_h True
```

don't use average pooling (dtype=bool)

```
--no_avg_pooling True
```

filter size for input x (dtype=int)

```
--filter_size_x 7
```

filter size for input h (dtype=int)

```
--filter_size_h 7
```

use static RNN (dtype=bool)

```
--use_static_rnn False
```

2.5.1.2.2 Module contents

2.5.2 MDGRUClassification

2.5.3 Module contents

2.6 Evaluation module

iterations after which to evaluate on validation set (dtype=int)

```
--test_each 2500
```

iterations after which to create a checkpoint (dtype=int)

```
--save_each None
```

iteration after which to create a plot (dtype=int)

```
--plot_each 2500
```

test size (dtype=int)

```
--test_size 1
```

whether to perform validation on the full images (dtype=bool)

```
--perform_full_image_validation True
```

save only labels and no probability distributions (dtype=bool)

```
--only_save_label False
```

always pick other random samples for validation (dtype=bool)

```
--validate_same True
```

number times we want to evaluate one volume; this only makes sense using a keep rate of less than 1 during evaluation (dropout_during_evaluation less than 1) (dtype=int)

```
--evaluate_uncertainty_times 1
```

keep rate of weights during evaluation; useful to visualize uncertainty in conjunction with a number of samples per volume (dtype=float)

```
--evaluate_uncertainty_dropout 1.0
```

Save each evaluation sample per volume. Without this flag, only the standard deviation and mean over all samples is kept. (dtype=bool)

```
--evaluate_uncertainty_saveall False
```

2.6.1 Supervised evaluation

class mdgru.eval.**SupervisedEvaluation**(*modelcls*, *datacls*, *kw*)
Bases: object

Handler for the evaluation of model defined in modelcls using data coming from datacls.

Parameters

- **kw** (*dict containing the following options.*) –
  - **dropout_rate** [default: 0.5] "keep rate" for weights using dropconnect. The higher the value, the closer the sampled models to the full model.
  - **namespace** [default: default] override default model name (if no ckpt is provided). Probably not a good idea!
  - **only_save_labels** [default: False] save only labels and no probability distributions
  - **validate_same** [default: True] always pick other random samples for validation!
  - **evaluate_uncertainty_times** [default: 1] Number times we want to evaluate one volume. This only makes sense using a keep rate of less than 1 during evaluation (dropout_during_evaluation less than 1)
  - **evaluate_uncertainty_dropout** [default: 1.0] Keeprate of weights during evaluation. Useful to visualize uncertainty in conjunction with a number of samples per volume
  - **evaluate_uncertainty_saveall** [default: False] Save each evaluation sample per volume. Without this flag, only the standard deviation and mean over all samples is kept.
  - **show_f05** [default: True]
  - **show_f1** [default: True]
  - **show_f2** [default: True]
  - **show_l2** [default: True]
  - **show_cross_entropy** [default: True]
  - **print_each** [default: 1] print execution time and losses each # iterations
  - **batch_size** [default: 1] Minibatchsize
  - **datapath** path where training, validation and testing folders lie. Can also be some other path, as long as the other locations are provided as absolute paths. An experimentsfolder will be created in this folder, where all runs and checkpoint files will be saved.
  - **locationtraining** [default: None] absolute or relative path to datapath to the training data. Either a list of paths to the sample folders or one path to a folder where samples should be automatically determined.
  - **locationtesting** [default: None] absolute or relative path to datapath to the testing data. Either a list of paths to the sample folders or one path to a folder where samples should be automatically determined.
  - **locationvalidation** [default: None] absolute or relative path to datapath to the validation data. Either a list of paths to the sample folders or one path to a folder where samples should be automatically determined.

- **output_dims** number of output channels, e.g. number of classes the model needs to create a probability distribution over.
- **windowsize** window size to be used during training, validation and testing, if not specified otherwise
- **padding** [default: [0]] padding to be used during training, validation and testing, if not specified otherwise. During training, the padding specifies the amount a patch is allowed to reach outside of the image along all dimensions, during testing, it specifies also the amount of overlap needed between patches.
- **windowsizetesting** [default: None] override windowsize for testing
- **windowsizevalidation** [default: None]
- **paddingtesting** [default: None] override padding for testing
- **paddingvalidation** [default: None]
- **testbatchsize** [default: 1] batchsize for testing

- **modelcls** (*cls*) – Python class defining the model to evaluate
- **datacls** (*cls*) – Python class implementing the data loading and storing

_defaults = {'batch_size': {'help': 'Minibatchsize', 'name': 'batchsize', 'short':

_load(*f*)

Load model in current framework from f

Parameters **f** (*location of stored model*) –

_predict(*batch*, *dropout*, *testing*)

Predict given batch and keeprate dropout.

Parameters

- **batch** (*ndarray*) –
- **dropout** (*float*) – Keeprate for dropconnect
- **testing** –

Returns **ndarray** (*Prediction based on data batch*)

_predict_with_loss(*batch*, *batchlabs*)

Predict for given batch and return loss compared to labels in batchlabs

Parameters

- **batch** (*image data*) –
- **batchlabs** (*corresponding label data*) –

Returns *tuple of ndarray prediction and losses*

_save(*f*)

Save to file f in current framework

Parameters **f** (*location to save model at*) –

_set_session(*sess*, *cachefolder*)

_train()

Performs one training iteration in respective framework and returns loss(es)

add_summary_simple_value(*text*, *value*)

get_globalstep()
Return number of iterations this model has been trained in

Returns **int** (*iteration count*)

load(*f*)
loads model at location f from disk

Parameters **f** (*str*) – location of stored model

save(*f*)
saves model to disk at location f

Parameters **f** (*str*) – location to save model to

set_session(*sess*, *cachefolder*, *train=False*)

test_all_available(*batch_size=None*, *dc=None*, *return_results=False*, *dropout=None*, *testing=False*)
Completely evaluates each full image in tps using grid sampling.

Parameters

- **batch_size** (*int*) – minibatch size to compute on
- **dc** (*datacollection instance, optional*) – datacollection to sample from
- **return_results** (*bool*) – should results be returned or stored right away?
- **dropout** (*float*) – keeprate of dropconnect for inference
- **testing** –

Returns *either tuple of predictions and errors or only errors, depending on return_results flag*

test_all_random(*batch_size=None*, *dc=None*, *resample=True*)
Test random samples

Parameters

- **batch_size** (*int*) – minibatch size to compute on
- **dc** (*datacollection instance, optional*) – datacollection to sample from
- **resample** (*bool*) – indicates if we need to sample before evaluating

Returns *tuple of loss and prediction ndarray*

test_scores(*pred*, *ref*)
Evaluates all selected scores between reference data ref and prediction pred.

Parameters

- **pred** (*ndarray*) – prediction, as probability distributions per pixel / voxel
- **ref** (*ndarray*) – labelmap, either as probability distributions per pixel / voxel or as label map

train()
Measures and logs time when performing data sampling and training iteration.

2.6.2 TensorFlow backend

2.6.3 PyTorch backend

2.7 Data loader module

data augmentation: standard deviation to use for Gaussian filtered images during high pass filtering (dtype=int)

```
--SubtractGaussSigma 5
```

data augmentation: use only Gauss-Sigma filtered images (dtype=bool)

```
--nooriginal False
```

data augmentation: deformation grid spacing in pixels (dtype=int); if zero: no deformation will be applied

```
--deform 0
```

data augmentation: given a deformation grid spacing, this determines the standard deviations for each dimension of the random deformation vectors (dtype=float)

```
--deformSigma 0.0
```

data augmentation: activate random mirroring along the specified axes during training (dtype=bool)

```
--mirror [0]
```

data augmentation: random multiplicative Gaussian noise with unit mean, unit variance (dtype=bool)

```
--gaussiannoise False
```

data augmentation: amount of randomly scaling images per dimension as a factor (dtype=float)

```
--scaling 0.0
```

data augmentation: amount in radians to randomly rotate the input around a randomly drawn vector (dtype=float)

```
--rotation 0.0
```

sampling outside of discrete coordinates (dtype=float)

```
--shift 0.0
```

interpolation when using no deformation grids (dtype=bool)

```
--interpolate_always False
```

define random seed for deformation variables (dtype=int)

```
--deformseed 1234
```

spline order interpolation_order in 0 (constant), 1 (linear), 2 (cubic) (dtype=int)

```
--interpolation_order 3
```

rule on how to add values outside image boundaries (“constant”, “nearest”, “reflect”, “wrap”) (dtype=str)

```
--padding_rule constant
```

whiten image data to mean 0 and unit variance (dtype=bool)

```
--whiten True
```

force each n-th sample to contain labelled data (dtype=int)

```
--each_with_labels 0
```

whether channels appear first (PyTorch) or last (TensorFlow) (dtype=bool)

```
--channels_first False
```

if multiple masks are provided, we select one at random for each sample (dtype=bool)

```
--choose_mask_at_random False
```

perform one-hot-encoding from probability distributions (dtype=bool)

```
--perform_one_hot_encoding True
```

ignore missing masks (dtype=bool)

```
--ignore_missing_mask False
```

correct nifti orientation (dtype=bool)

```
--correct_orientation True
```

2.7.1 DataCollection

class mdgru.data.**DataCollection**(*kw*)

Bases: object

Abstract class for all data handling classes.

Parameters **kw** (*dict containing the following options.*) –

- **seed** [default: 1234] Seed to be used for deterministic random sampling, given no threading is used
- **nclasses** [default: None]

_defaults = {'nclasses': None, 'seed': {'help': 'Seed to be used for deterministic

_one_hot_vectorize(*indexlabels*, *nclasses=None*, *zero_out_label=None*)

simplified onehotlabels method. we discourage using interpolated labels anyways, hence this only allows integer values in indexlabels

Parameters

- **indexlabels** (*ndarray*) – array containing labels or indices for each class, starting at 0 until nclasses-1
- **nclasses** (*int*) – number of classes

- **zero_out_label** (*int*) – label to assign probability of zero for the whole probability distribution

Returns *ndarray* – Probabilitydistributions per pixel where at position indexlabels the value is set to 1, otherwise to 0

static get_all_tps(*folder*, *featurefiles*, *maskfiles*)
computes list of all folders that are subfolders of folder and contain all provided featurefiles and maskfiles.

Parameters

- **folder** (*str*) – location at which timepoints are searched
- **featurefiles** (*list of str*) – necessary featurefiles to be contained in a timepoint
- **maskfiles** (*list of str*) – necessary maskfiles to be contained in a timepoint

Returns *sorted list* – valid timepoints in string format

get_data_dims()
Returns the dimensionality of the whole collection (even if samples are returned/computed on the fly, the theoretical size is returned). Has between two and three entries (Depending on the type of data. A dataset with sequence of vectors has 3, a dataset with sequences of indices has two, etc)

Returns *list* – A shape array of the dimensionality of the data.

get_shape()

get_states()
Get states of this data collection

random_sample(***kw*)
Randomly samples from our dataset. If the implementation knows different datasets, the dataset string can be used to choose one, if not, it will be ignored.

Parameters ****kw** (*keyword args*) – batch_size can be set, amongst other parameters. See implementing methods for more detail.

Returns *array* – A random sample of length batch_size.

reset_seed(*seed=12345678*)
reset main random number generator with given seed

set_states(*state*)
reset random state generators given the states in "states"

Parameters **states** (*object*) – Random generator state

2.7.2 GridDataCollection

2.8 Helper routines and functions

Those functions are not accessible and only needed for the software.

mdgru.helper.**_initializer_Q**(*k1*, *k2*)
Computes a block circulant k1xk1 matrix, consisting of circulant k2xk2 blocks.

Parameters

- **k1** – Outer convolution filter size
- **k2** – Inner convolution filter size

Returns block circulant matrix with circulant blocks

mdgru.helper.**argget**(*dt*, *key*, *default=None*, *keep=False*, *ifset=None*)
Evaluates function arguments.

It takes a dictionary dt and a key "key" to evaluate, if this key is contained in the dictionary. If yes, return its value. If not, return default. By default, the key and value pair are deleted from the dictionary, except if keep is set to True. ifset can be used to override the value, and it is returned instead (except if None). :param dt: dictionary to be searched :param key: location in dictionary :param default: default value if key not found in dictionary :param keep: bool, indicating if key shall remain in dictionary :param ifset: value override. :return: chosen value for key if available. Else default or ifset override.

mdgru.helper.**check_if_kw_empty**(*class_name*, *kw*, *module_name*)
Prints standardised warning if an unsupported keyword argument is used for a given class

mdgru.helper.**collect_parameters**(*cls*, *kw_args={}*)
helper function to collect all parameters defined in cls' _defaults needed for both compile arguments and define arguments :param cls: :param kw_args: :return:

mdgru.helper.**compile_arguments**(*cls*, *kw*, *transitive=False*, *override_static=False*, *keep_entries=True*)
Extracts valid keywords for cls from given keywords and returns the resulting two dicts.

Parameters

- **cls** – instance or class having property or attribute "_defaults", which is a dict of default parameters.
- **transitive** – determines if parent classes should also be consulted
- **kw** – the keyword dictionary to separate into valid arguments and rest

Returns tuple of dicts, one with extracted/copied relevant keywords and one with the rest

mdgru.helper.**counter_generator**(*maxim*)
Generates indices over multidimensional ranges.

Parameters **maxim** – Number of iterations per dimension

Returns Generator yielding next iteration

mdgru.helper.**define_arguments**(*cls*, *parser*)
Requires cls to have field _defaults! Parses all fields defined in _defaults and creates a argparse compatible structure from them. These are then appended to the parser structure.

Parameters

- **cls** – cls which contains (at least an empty) _defaults dict.
- **parser** – parser to add parameters to

Returns parser with added parameters

mdgru.helper.**deprecated**(*func*)
Decorator function to indicate through our logger that the decorated function should not be used anymore :param func: function to decorate :return: decorated function

mdgru.helper.**force_symlink**(*file1*, *file2*)
Tries to create symlink. If it fails, it tries to remove the folder obstructing its way to create one. :param file1: path :param file2: symlink name

mdgru.helper.**generate_defaults_info**(*cls*)
Adds description for keyword dict to docstring of class cls

Parameters **cls** (*Class to extract _defaults field from to generate additional docstring info from.*) –

mdgru.helper.**harmonize_filter_size**(*fs*, *ndim*)
Used in both model classes to set filters to their default values, given either no or incomplete input. :param fs: filters as specified :param ndim: number of dimensions :return: corrected list for filter sizes

mdgru.helper.**initializer_W**(*n*, *k1*, *k2*)
Computes kronecker product between an orthogonal matrix T and a circulant orthogonal matrix Q.

Creates a block circulant matrix using the Kronecker product of a orthogonal square matrix T and a circulant orthogonal square matrix Q.

Parameters

- **n** – Number of channels
- **k1** – Outer convolution filter size
- **k2** – Inner convolution filter size

Returns

mdgru.helper.**lazy_property**(*function*)
This function computes a property or simply returns it if already computed.

mdgru.helper.**notify_user**(*chat_id*, *token*, *message='no message'*)
Sends a notification when training is completed or the process was killed.

Given that a telegram bot has been created, and it's api token is known, a chat_id has been opened, and the corresponding chat_id is known, this method can be used to be informed if something happens to our process. More on telegram bots at https://telegram.org/blog/bot-revolution. :param chat_id: chat_id which is used by telegram to communicate with your bot :param token: token generated when creating your bot through the BotFather :param message: The message to be sent

mdgru.helper.**np_arr_backward**(*matrix*, *n*, *k1*, *k2*)
Transforms from block block circulant matrix to filter representation using indices. :param matrix: matrix representation of filter :param n: number of channels :param k1: filter dim 1 :param k2: filter dim 2 :return: filter representation

mdgru.helper.**np_arr_forward**(*filt*, *n*, *k1*, *k2*)
Transforms from filter to block block circulant matrix representation using indices. :param filt: filter variable :param n: number of channels :param k1: filter dimension 1 :param k2: filter dimension 2 :return: matrix representation of filter

2.9 Runner module

The optional parameters from the runner module are documentend in the evaluation module for simplicity.

class mdgru.runner.**Runner**(*evaluationinstance*, ***kw*)
Bases: object

Parameters

- **kw** (*dict containing the following options.*) –
  - **test_each** [default: 2500] validate after # training iterations
  - **save_each** [default: None] save after # training iterations
  - **plot_each** [default: 2500]

- **test_size** [default: 1]
- **test_iters** [default: 1] number of validations to perform on random samples. Only makes sense if full_image_validation is not set
- **test_first** [default: False] Perform validation on the untrained model as well.
- **perform_full_image_validation** [default: True] Use random samples instead of the complete validation images
- **save_validation_results** [default: True] Do not save validation results on the disk
- **notifyme** [default: None] Experimental feature that when something goes amiss, this telegram chat id will be used to inform the respective user of the error. This requires a file called config.json in the same folder as this file, containing a simple dict structure as follows: {"chat_id": CHATID, "token": TOKEN}, where CHATID and TOKEN have to be created with Telegrams BotFather. The chatid from config can be overriden using a parameter together with this option.
- **results_to_csv** [default: True] Do not create csv with validation results
- **checkpointfiles** [default: None] provide checkpointfile for this template. If no modelname is provided, we will infer one from this file. Multiple files are only allowed if only_test is set. If the same number of optionnames are provided, they will be used to name the different results. If only one optionname is provided, they will be numbered in order and the checkpoint filename will be included in the result file.
- **epochs** [default: 1] Number of times through the training dataset. Cant be used together with "iterations"
- **iterations** [default: None] Number of iterations to perform. Can only be set and makes sense if epochs is 1
- **only_test** [default: False] Only perform testing. Requires at least one ckpt
- **only_train** [default: False] Only perform training and validation.
- **experimentloc** [default: /home/docs/experiments]
- **optionname** [default: None] name for chosen set of options, if multiple checkpoints provided, there needs to be 1 or the same number of names here
- **fullparameters** [default: None]

• **evaluationinstance** (*instance of an evaluation class*) – Will be used to call train and test routines on.

_defaults = {'checkpointfiles': {'help': 'provide checkpointfile for this template.

_finish(*signal*)

calc_min_mean_median_max_errors(*errors*)

ignore_signal = True

run(***kw*)

save(*filename*)

test()

train()

validation(*showIt=True*, *name=1574669911.1845152*)

write_error_to_csv(*errors*, *filename*, *minerrors*, *avgerrors*, *medianerrors*, *maxerrors*)

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

Symbols

A

C

D

F

G

H

I

L

M