FreeSurfer Multiprocessing Pipeline

Contents

Module `freesurfer_wrapper`

freesurfer_wrapper

freesurfer_wrapper aims to facilitate the creation of a multiprocessing pipeline using FreeSurfer. It is a Python wrapper to execute parallel runs of recon-all and some pial edits algorithms.

Requirements

- Docker[1]
- FreeSurfer license key[2]

Preparation

1) Place the license in a license.txt file in the same folder as the Dockerfile.
2) Place your dataset folder in the same folder as the Dockerfile.
3) Build the docker image:

```
docker build -t fs_wrapper .
```

Overview run

The main script has 3 commands, each of them is explained below.

```
python run.py -h
usage: run.py [-h] {recon,edit,recon_edit} ...

Command-line wrapper tool to execute parallel runs of FreeSurfer recon-all and
some pial edits algorithms.

positional arguments:
  {recon,edit,recon_edit}
    recon               Run FreeSurfer recon-all.
    edit                Run mri_gcut and mri_binarize for pial edits.
    recon_edit          Re-run recon-all for pial edits.

optional arguments:
  -h, --help            show this help message and exit
```

recon

Run FreeSurfer recon-all.

```
python run.py recon -h
usage: run.py recon [-h] -i INPUT [-p PARALLEL]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Tab separated file. First column: unique ID. Second
                        column: path to dcm/nii file.
  -p PARALLEL, --parallel PARALLEL
                        Number of parallel runs (default: number of CPUs).
```

[1] https://www.docker.com/
[2] https://surfer.nmr.mgh.harvard.edu/registration.html

edit

Run mri_gcut and mri_binarize for pial edits.

```
python run.py edit -h
usage: run.py edit [-h] -i INPUT [-p PARALLEL]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Tab separated file. First column: unique ID. Second
                        column: path to dcm/nii file. Third column: tissue
                        ratio
  -p PARALLEL, --parallel PARALLEL
                        Number of parallel FS runs (default: number of CPUs).
```

recon_edit

Re-run recon-all for pial edits.

```
python run.py recon_edit -h
usage: run.py recon_edit [-h] -i INPUT [-p PARALLEL]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Subject id list file.
  -p PARALLEL, --parallel PARALLEL
                        Number of parallel FS runs (default: number of CPUs).
```

How to run recon

Create the input file

A tab-separated file is needed as input with the following characteristics:

- Each line must represent a single scan.
- First column: unique ID.
- Second column: path to dcm/nii file.

For the ADNI dataset, you can create this file using the create_recon_input.py script. This will create a recon_input.txt file. The script will combine the subject ID and the session ID to create a unique ID.

```
docker run --rm -it -v $(pwd):/root/freesurfer_wrapper fs_wrapper \
python scripts/create_recon_input.py
```

As an example, a recon_input.txt file for some ADNI records will look like this:

```
137_S_1414_S46193   ADNI/137_S_1414/MP-RAGE/2008-02-26_11_57_53.0/S46193/ADNI_137_S_1414_MR_MP-RAGE_
137_S_1414_S72806   ADNI/137_S_1414/MP-RAGE/2009-08-26_11_06_33.0/S72806/ADNI_137_S_1414_MR_MP-RAGE_
```

For other datasets you can try to edit the PATH_PATTERN variable in `scripts/create_recon_input.py`.

Run recon

Now you can run the recon command using Docker. This can take several hours.

```
docker run --rm -it -v $(pwd):/root/freesurfer_wrapper fs_wrapper \
python run.py recon -i recon_input.txt
```

This command will use the maximum number of CPUs. You can append the `-p <INT>` flag where `<INT>` is the number of parallel runs you want.

How to run edit

Create the input file

After running recon you can check your results using freeview. It is not possible to run freeview using Docker, the graphical user interface cannot be displayed. Therefore, use your host machine freeview command to check the results.

```
SUBJECTS_DIR=$(pwd)/FS_OUTPUTS
freeview -recon <UNIQUE_ID>
```

If any skull edits are necessary, you need to create an input table to run edit with the following characteristics:

- Each line must represent a single scan.
- First column: unique ID.
- Second column: path to dcm/nii file.
- Third column: the tissue ratio for WM edits.

Tissue ratio is the threshold to value (%) of WM intensity. The value should be >0 and <1; larger values would correspond to cleaner skull-strip but higher chance of brain erosion.

You can copy the recon input table, keep only the lines for the scans that need pial edits, and add the column with the tissue ratio values.

As an example, a edit_input.txt file for some ADNI records will look like this:

```
137_S_1414_S46193    ADNI/137_S_1414/MP-RAGE/2008-02-26_11_57_53.0/S46193/ADNI_137_S_1414_MR_MP-RAGE_
137_S_1414_S72806    ADNI/137_S_1414/MP-RAGE/2009-08-26_11_06_33.0/S72806/ADNI_137_S_1414_MR_MP-RAGE_
```

Run edit

Now you can run the edit command using Docker.

```
docker run --rm -it -v $(pwd):/root/freesurfer_wrapper fs_wrapper \
python run.py edit -i edit_input.txt
```

This command will use the maximum number of CPUs. You can append the `-p <INT>` flag where `<INT>` is the number of parallel runs you want.

You can check the resulting edited mask using freeview:

```
SUBJECTS_DIR=$(pwd)/FS_OUTPUTS
freeview -recon <UNIQUE_ID> -v brainmask.gcutsT$<TISSUE_RATIO>.mgz:colormap=heat:opacity=0.5
```

If still not good, change the tissue ratio value in the input file and run edit again. When all masks are OK, proceed to recon_edit command.

How to run recon_edit

Create the input file

The input file is the table used for edit with the final values for tissue ratio.

Run recon_edit

recon_edit will re-run parts of FS recon-all using the edited masks. This can take several hours.

```
docker run --rm -it -v $(pwd):/root/freesurfer_wrapper fs_wrapper python run.py recon_edit -i edit_i
```

This command will use the maximum number of CPUs. You can append the `-p <INT>` flag where `<INT>` is the number of parallel runs you want.

How to check for completed runs and hard errors

FreeSurfer's recon-all command creates different logs while running. The `recon-all.done` log is created only for completed runs. The `recon-all.error` is created for hard failures.

You can check these logs using a custom script. The script was written to work on ADNI folder structure. For other datasets you can try to edit the PATH_PATTERN variable in scripts/check_logs.py.

Done

```
docker run --rm -it -v $(pwd):/root/freesurfer_wrapper fs_wrapper \
python scripts/check_logs.py done
```

You can also pipe the output to bash word count command to get a quick count:

```
docker run --rm -it -v $(pwd):/root/freesurfer_wrapper fs_wrapper \
python scripts/check_logs.py done | wc -l
```

Error

```
docker run --rm -it -v $(pwd):/root/freesurfer_wrapper fs_wrapper \
python scripts/check_logs.py error
```

How to restart after a computer failure

If the execution of the pipeline is halted by a computer failure or system restart then you have to update the input file of the recon command.

```
bash scripts/update_recon_input.sh
```

This will remove all "done" samples from the original recon_input. It will also delete the folders from the samples that were running when the failure happened. These samples will run again from scratch.

Now run again the `recon` command but using the new input file (`<YYYY-MM-DD>_recon_input.txt`).

Quality control

The tool is packaged with **qatools-python**[3] version 1.2 for quality control measurements. This script was developed by Reuter DeepMI Lab[4] as a revision, extension, and translation to the Python language of the Freesurfer QA Tools.

```
docker run --rm -it -v $(pwd):/root/freesurfer_wrapper fs_wrapper \
python scripts/qatools-python/qatools.py --subjects_dir FS_OUTPUTS --output_dir QC \
--screenshots --outlier --fornix
```

This will create `qatools-results.csv` file; screenshots, outliers and fornix folders inside the QC folder. Please consult qatools-python docs[5] for a full explanation of each QC measurement.

Sub-modules

- freesurfer_wrapper.run
- freesurfer_wrapper.scripts

Module `freesurfer_wrapper.run`

Command-line wrapper tool to execute parallel runs of FreeSurfer recon-all and some pial edits algorithms.

This file can also be imported as a module and contains the following functions:

```
* argument_parser -  parser for command-line options, arguments and sub-commands.
* run_command -
* handle_workers - creates a pool of parallel worker processes running commands.
* worker - invokes a subprocess running the command.
* recon - formats recon-all command string.
* edit - formats mri_gcut and mri_binarize command string.
* recon_edit - formats a cp and recon-all command string.
```

[3] https://github.com/Deep-MI/qatools-python
[4] https://deep-mi.org/
[5] scripts/qatools-python/README.md#description

* parse_input_file - parses the input tables.

Functions

Function `argument_parser`

```
def argument_parser(
    args: list
) -> ArgumentParser.parse_args
```

Parser for command-line options, arguments and sub-commands.

Parameters

`args : list` Command-line arguments list

Returns

Parser

Function `edit`

```
def edit(
    edit_args: list
) -> str
```

Formats mri_gcut and mri_binarize command string. mri_gcut performs skull stripping algorithm based on graph cut. mri_binarize binarizes the edited mask.

Parameters

`edit_args : list` mri_gcut and mri_binarize arguments list

Returns

`mri_gcut [args] && mri_binarize [args]`

Function `handle_workers`

```
def handle_workers(
    p: int,
    command: function,
    input_file: str
)
```

Creates a pool of parallel worker processes running commands. Workers will be called until all lines from the input file are processed.

Parameters

`p : int` The number of parallel processes.
`command : function` Function returning the command-line string to pass the worker.
`input_file : str` Tab-separated .txt file.

Returns

None

Function `parse_input_file`

```
def parse_input_file(
    input_file: str
) -> List[List[str]]
```

Parses the input tables.

Parameters

`input_file : str` Tab-separated .txt file.

Returns

File lines and columns parsed as a list of lists.

Function `recon`

```
def recon(
    recon_args: list
) -> str
```

Formats recon-all command string.

Parameters

`recon_args` **: list** recon-all arguments list

Returns

`recon-all [args]`

Function `recon_edit`

```
def recon_edit(
    recon_edit_args: list
) -> str
```

Formats a cp and recon-all command string. cp replaces the original brainmask with the edited brainmask.gcutsT{tissue_ratio}.mgz. recon-all re-runs -autorecon2-wm -autorecon3 stream with the new mask.

Parameters

`recon_edit_args` **: list** cp and recon-all arguments list

Returns

`cp [args] && recon-all [args]`

Function `run_command`

```
def run_command(
    args
)
```

Pass the appropriate command function to the worker handler.

Parameters

`args` **: list** Command-line arguments list

Returns

None

Function `worker`

```
def worker(
    cmd: str
) -> <function run at 0x7f6a967f5670>
```

Invokes a subprocess running the command.

Parameters

`cmd` **: str** Command-line string

Returns

subprocess.run()

Namespace `freesurfer_wrapper.scripts`

Sub-modules

- freesurfer_wrapper.scripts.check_logs
- freesurfer_wrapper.scripts.create_recon_input

Module `freesurfer_wrapper.scripts.check_logs`

Script to check recon-all logs for each run

usage: python check_logs.py <done|error>

Please edit the **PATH_PATTERN** variable with the appropriate pathname pattern to find each file.

This file can also be imported as a module and contains the following functions:

```
* get_logs - get the path for each log based on pathname pattern.
* print_id_from_logs - prints the IDs from a list of logs.
```

Functions

Function `get_logs`

```
def get_logs(
    path_pattern: str
) -> list
```

Get the path for each log based on pathname pattern.

Parameters

`path_pattern` : str Glob pathname pattern to find each log.

Returns

List of log paths

Function `print_id_from_logs`

```
def print_id_from_logs(
    logs: list
)
```

Prints the IDs from a list of logs.

Parameters

`logs` : list List of log paths

Returns

None

Module `freesurfer_wrapper.scripts.create_recon_input`

Script to create recon input table

This script creates an input table based on the directory organization of the image files.

Please edit the **PATH_PATTERN** variable with the appropriate pathname pattern to find each file.

This file can also be imported as a module and contains the following functions:

```
* create_input_file - creates the input table.
```

Functions

Function `create_input_file`

```
def create_input_file(
    path_pattern: str
)
```

Creates a two column text file to be used as input for the main script recon command. First column: unique ID (combines SUBJECT ID and SESSION ID). Second column: path to DICOM file.

Parameters

`path_pattern` : str Glob pathname pattern to find each DICOM file.

Returns

None

Generated by *pdoc* 0.10.0 (https://pdoc3.github.io).