Pipeline to process (f)MRI data on a remote server in a BIDS-compatible fashion using Docker and Singularity

Björn C. Schiffler

Jensen lab, Department of Clinical Neuroscience, Karolinska Institutet, Stockholm, Sweden

Motivation

Reproducibility is a common problem in modern science. This is not only true for the replication of studies across labs - even within labs, it is often difficult to pick up with a data analysis where other people have left off because decisions about the flow of data are often arbitrary and non-standardized. Further, data sharing is often complicated by the fact that data can be stored in various different ways.

For applications in neuroscience, the Brain Imaging Data Structure BIDS (Figure 1) has been developed to standardize the process of storing and analysing MRI-derived data.

This pipeline document will guide a standardized way of organizing (f)MRI data by using tools which allow automatized structuring into BIDS format, doing quality control via MRIQC and then preprocessing them using the fmriprep pipeline.

We are going to be using two tools called Docker and Singularity throughout this pipeline which make standardized processing much easier. The idea of a Docker/Singularity container is similar to a virtual machine: The container has prespecified versions of certain software modules (e.g. Python, Freesurfer, FSL etc.) so that the end-user does not have to collect all of the software libraries by herself.

Further, a particular version of the container can be specified to ensure reproducibility (this is particularly important for the preprocessing pipeline using fmriprep). In contrast to full virtual machines, the container relies on more subprocesses of the operating system it is being run on (but this also makes it more lightweight).

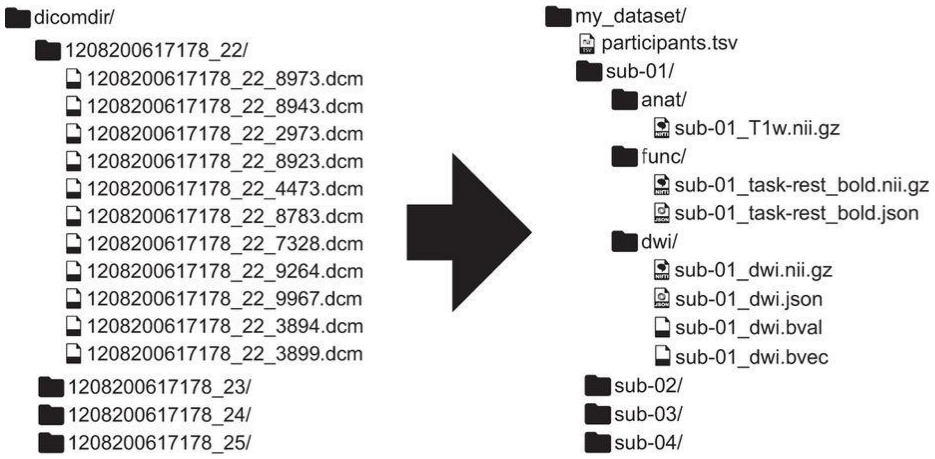

Figure 1: BIDS pipeline by Gorgolewski et al. 2016 Scientific Data

Prerequisites

- Account on the server if data should be processed remotely
- Singularity needs to be installed on the server
- To make your own Singularity containers, e.g. to update the pipeline, you need a local Docker installation, this is easiest within a Linux distribution, but a virtual machine is sufficient for this conversion process NB: It is possible this might not be necessary in the future if containers can be directly pulled from SingularityHub

If your data is already in validated BIDS format, you can skip to Part 3 in this pipeline.

The Pipeline

Part 1 - Setting up directories and files

1. Move to the main study directory (create this if necessary):
   ```
   cd bids_examples
   ```
2. Locate the subject IDs of the DICOMs that you want to process
3. Create a simple textfile which contains subject IDs located in step 2:
   ```
   atom participants_in.txt
   ```
   After putting the unique subject IDs in, your textfile should look something like this:
   ```
   6548
   4398
   2204
   ```

Part 2 - Conversion of DICOMs to Nifti in BIDS structure via heudiconv

We are going to use the heuristic conversion tool heudiconv to get from raw DICOM files to BIDS format. This step can be performed with other converter tools as well.

To set up a Singularity container, see these instructions. In short, you need to use Docker (this can be on another machine than the server) to convert the Docker container of choice (i.e., a specific version of fmriprep) into a Singularity one. Usually, a list of available container versions can be found on Docker Hub, see e.g., for fmriprep.

Execute this command on the machine you have installed Docker on:

```
sudo docker run --privileged -t --rm \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /media/sf_VMShared/heudiconv:/output \
    singularityware/docker2singularity \
    nipy/heudiconv:latest
```

where "/media/sf_VMShared/heudiconv/" is the output directory, this needs to be adapted to your machine.

Then copy the created Singularity container .img to the server. The following commands will link to already created Singularity containers. If other containers should be used, the links need to be changed.

Running the heudiconv Singularity container:

First we will run the pipeline with the standard heuristic and only on one subject: (don't worry if this looks intimidating, the function will be explained in detail below). Replace the number 6548 in the function call by any other subject ID that you want to process and call this command from the newly created folder.

```
singularity run -B $PWD:/curdir -B \
/data/dicom:/dcminput /data/bjorn/singularity/nipy_heudiconv_latest-2017-12-15-a6ab851284c6.img \
-d /dcminput/{subject}_*/*/*dcm \
-s 6548 -f /src/heudiconv/heuristics/convertall.py \
-c none -b -o /curdir/output/
```

Now there will be a directory ".heudiconv" in the output folder which can be accessed: `cd .heudiconv`. Inside will be the subject and subfolder "info", e.g. `cd 6548/info`. Here are several files. We are particularly interested in "dicominfo.tsv" as it holds the information for all the sequences and can be accessed e.g. like this: `libreoffice dicominfo.tsv`

For heudiconv to know which sequences belong to which type of data, a heuristic (Python file) needs to be created:

```
atom heuristic.py
```

An example for a heuristic file is shown below, but this needs to be adapted according to the study/sequences used:

```
import os

def create_key(template, outtype=('nii.gz',), annotation_classes=None):

    if template is None or not template:
        raise ValueError('Template must be a valid format string')
    return template, outtype, annotation_classes

def infotodict(seqinfo):

    print(seqinfo)

    t1w = create_key('sub-{subject}/anat/sub-{subject}_T1w')
    dwi = create_key('sub-{subject}/dwi/sub-{subject}_run-{item:01d}_dwi')
    pressure1 = create_key('sub-{subject}/func/sub-{subject}_task-pressure_rec-{rec}_run-1_bold')
    pressure2 = create_key('sub-{subject}/func/sub-{subject}_task-pressure_rec-{rec}_run-2_bold')

    info = {t1w: [], dwi: [], rest: []}

    for s in seqinfo:
        if (s.dim3 == 176) and ('t1' in s.series_description or 'T1' in s.series_description) and not s
            info[t1w] = [s.series_id]  # assign if a single series meets criteria
        if (s.dim3 == 3900) and (s.dim4 == 1) and ('dti' in s.series_description or 'DTI' in s.series_d
            info[dwi].append(s.series_id)  # append if multiple series meet criteria
        if (s.dim3 == 13650) and ('fmri_1' in s.series_description):
            if s.is_motion_corrected:  # exclude non motion corrected series
                info[pressure1].append({'item': s.series_id, 'rec': 'corrected'})
            else:
                info[pressure1].append({'item': s.series_id, 'rec': 'uncorrected'})
    if (s.dim3 == 16044) and ('fmri_2' in s.series_description):
        if s.is_motion_corrected:  # exclude non motion corrected series
```

```
                info[pressure2].append({'item': s.series_id, 'rec': 'corrected'})
            else:
                info[pressure2].append({'item': s.series_id, 'rec': 'uncorrected'})
    return info
```

Once this heuristic has been created, the full heudiconv call can be performed on all participants in the textfile participants_in.txt:

```
singularity run -B $PWD:/curdir -B /data/dicom:/dcminput \
/data/bjorn/singularity/nipy_heudiconv_latest-2017-12-15-a6ab851284c6.img \
-d /dcminput/{subject}_*/*/*dcm \
-s `cat participants_in.txt` -f /curdir/heuristic.py \
-c dcm2niix -b -o /curdir/output/
```

In the following paragraphs I will explain the components of this quite dense function call in detail:

```
singularity run
```

The main call to the Singularity container.

```
-B $PWD:/curdir -B /data/dicom:/dcminput
```

This maps the present working directory to the /curdir directory inside the container and our main directory for storing dicoms (/data/dicom/) to the container folder dcminput. This can be changed if the dicoms are stored in some other place. And mappings can be done to already existing directories inside the container if necessary (e.g. to "scratch" instead of "curdir" and "data" instead of "dcminput").

```
/data/bjorn/singularity/nipy_heudiconv_latest-2017-12-15-a6ab851284c6.img
```

Specifies the location of the Singularity container for heudiconv.

```
-d /dcminput/{subject}*/*/*dcm
```

Points to the main dicom directory and uses wildcards to allow to find the dicoms in the subdirectories. Here, for subject "6548", the data is stored in `/data/dicom/6548_*/*/*dcm` (because the main dir naming convention on the server is "6548_20…", there are several dicom folders, and the dicoms all have different names).

```
-s `cat participants_in.txt`
```

-s identifies the subjects to be processed. The Linux `cat` command puts the subject IDs defined in participants.txt in a list. To try out the converter, it makes sense to put in only one subject ID here instead of a list, e.g. `-s 6548`.

```
-f /curdir/heuristics/heuristic.py
```

Points to the heuristic to be used to extract the data (remember, the "/curdir/" directory is inside the container and we have mapped it to the present working directory above).

```
-c dcm2niix -b -o /curdir/output/
```

Specifies the transformation (from DICOM format to Nifti) and the output directory.

Sidenote: With `singularity shell container.img`, we can look into the container and whether folders are correctly mapped.

BIDS validation

At the end of this step you should validate whether the directories you created fulfill the BIDS standard, for example using the Online BIDS validator or by using a Singularity container:

Making the Singularity container (locally with Docker) if it is not available on the server yet:

```
sudo docker run --privileged -t --rm \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /media/sf_VMshared/bids_validator:/output \
    singularityware/docker2singularity \
    bids/validator:0.24.2
```

Call the container:

```
singularity run /data/bjorn/singularity/bids_validator_0.24.2-2017-12-20-5336325e7600.img \
/path/to/data
```

Part 3 - (f)MRI quality control using MRIQC

Again, we are first making a Singularity container out of a Docker container:

```
sudo docker run --privileged -t --rm \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /media/sf_VMshared/mriqc_0_10_0:/output \
    singularityware/docker2singularity \
    poldracklab/mriqc:0.10.0
```

where "/media/sf_VMshared/mriqc_0_10_0" is the output folder for the Singularity image

and then

```
singularity run \
/data/bjorn/singularity/poldracklab_mriqc_0.10.0-2017-11-19-90032168ea7b.img \
$PWD $PWD/output participant --no-sub --participant_label 6548 \
-w $PWD/work --n_procs 3 --ants-nthreads 2 --mem_gb 10 --verbose-reports
```

or for all participants in the folder:

```
singularity run \
/data/bjorn/singularity/poldracklab_mriqc_0.10.0-2017-11-19-90032168ea7b.img \
$PWD $PWD/output participant --no-sub  \
-w $PWD/work --n_procs 3 --ants-nthreads 2 --mem_gb 10 --verbose-reports
```

to run MRIQC on data.

To compare the obtained QC metrics to other datasets, there are many fMRI datasets openly available, some of which have MRIQC reports attached to them, for example on openfMRI.

Part 4 - fMRI preprocessing using fmriprep

Same as in the previous steps, pull the fmriprep Docker container of choice and convert it to a Singularity container:

```
sudo docker run --privileged -t --rm \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /media/sf_VMShared/fmriprep_1_0_0:/output \
    singularityware/docker2singularity \
    poldracklab/fmriprep:1.0.0
```

where "/media/sf_VMShared/fmriprep_1_0_0" is the output folder for the Singularity image.

Then you would copy the created Singularity image to a folder on the server and execute it e.g. like this if you want to run the pipeline without Freesurfer (the `nice` command is, of course, optional):

```
nice singularity run /data/bjorn/singularity/poldracklab_fmriprep_1.0.0-2017-12-07-e1d56047aab0.img \
$PWD $PWD/output participant --nthreads 10 -w $PWD/work --fs-no-reconall --ignore slicetiming \
--fs-license-file /data/bjorn/license.txt
```

or with the Freesurfer recon-all pipeline:

```
nice singularity run /data/bjorn/singularity/poldracklab_fmriprep_1.0.0-2017-12-07-e1d56047aab0.img \
$PWD $PWD/output participant --nthreads 10 -w $PWD/work --ignore slicetiming \
--fs-license-file /data/bjorn/license.txt
```

The "–ignore slicetiming" option is not strictly necessary and will only be active if there is a "SliceTiming" field present in the input dataset metadata. Point the "–fs-license-file" option to the location of a Freesurfer license file (important that this is a recent one, the format of the older files is not recognized by new Freesurfer versions). Relevant optional parameters to speed up performance are `--nthreads`, `--omp-nthreads` and `--mem_mb`.

Quality check of fmriprep processed data

While the fmriprep pipeline is almost entirely automatised, checking the quality of the data in the end is an important part of all data processing and should not be neglected. Fmriprep makes this convenient by providing .html outputs in the main output folder for every participant (although it should be said that if for example cortical thickness analyses are planned, it can make sense to take a more in-depth look at the quality of the Freesurfer output than is provided in the html files).

If the output images need to be viewed somewhere else, they can be copied (or directly scp'd) from the output directory via:

```
cp *.html /reports_directory/
cp --parents ./*/figures/* /reports_directory/
```