

# BIDS converter tool

All the code can be found here:

<https://github.com/free-dino/dcm2bids>

## 1. Installing the Dependencies

### 1.1 Python environment setup

For simplicity, I will use Conda to create and manage my Python environment. Create a new environment (named `bids`) with Python 3.12:

```
conda create -n bids python=3.12
```

Activate the environment

```
conda activate bids
```

### 1.2 Python package installation

Install the required Python packages:

- **dcm2bids**: Converts DICOM files to BIDS format.
- **pandas**: Reads and processes tabular data, including Excel and CSV files.
- **openpyxl**: Enables pandas to read `.xlsx` Excel files.
- **pydicom**: Inspects and manipulates DICOM files.
- **Gooley** : Transform the Command Line Applications into user friendly GUIs
- **pyinstaller**: Packages Python application into standalone executables for cross-platform distribution

```
pip install dcm2bids  
pip install pandas  
pip install openpyxl  
pip install pydicom
```

```
pip install Goocy
pip install pyinstaller
```

## 2, Inspect the given data:

Before converting the DICOM dataset to the BIDS format, I first inspect the data to understand its structure, available modalities, and metadata.

This step helps in writing a correct configuration file for `dcm2bids`.

### 2.1 Inspecting the data

#### 2.1.1 Loading datasets

```
ds = pydicom.dcmread(file, force=True)
```

Reads each file with `pydicom`, skipping invalid ones

#### 2.1.2 Metadata extraction

```
modality = ds0.get("Modality", "Unknown")
protocol_name = ds0.get("ProtocolName", "Unknown")
series_desc = ds0.get("SeriesDescription", "Unknown")
```

#### 2.1.3 Sorting slices

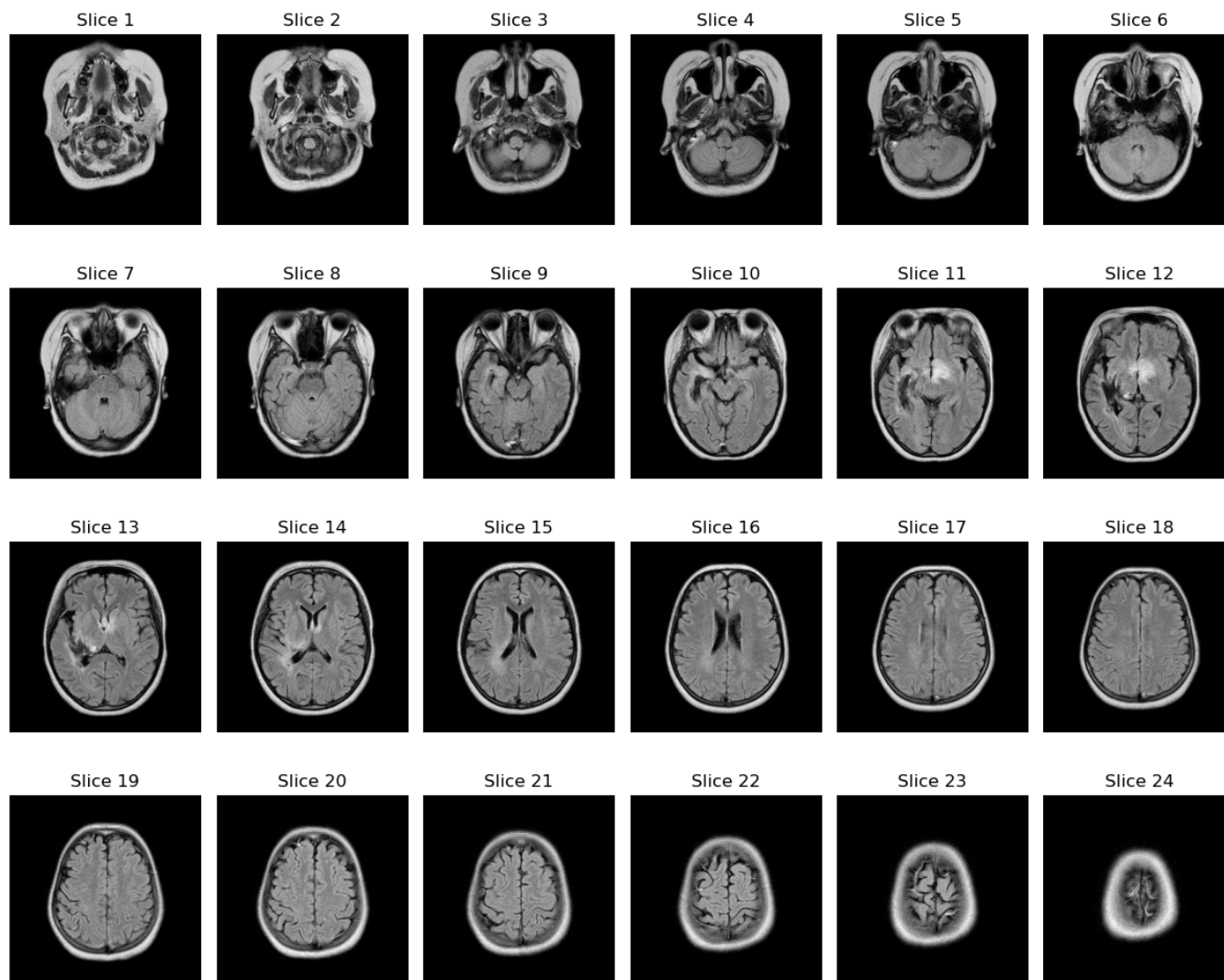
```
def get_sort_key(ds):
    return getattr(ds, "InstanceNumber",
                   getattr(ds, "SliceLocation",
                           ds.ImagePositionPatient[2] if
hasattr(ds, "ImagePositionPatient") else 0))
image_datasets.sort(key=get_sort_key)
```

Ensures slices are ordered properly for visualization

#### 2.1.4 Slice visualization:

```
ax.imshow(image_datasets[i].pixel_array, cmap="gray")
ax.set_title(f"Slice {i+1}")
```

Displays slices in a grid layout using Matplotlib, like this:



## 2.2 Purpose

This inspection process allows us to:

- Identify **SeriesDescription** and **ProtocolName** values for each scan.
- Determine the number of slices, image dimensions, and modality.
- Visualize a subset of slices for quality control.
- Collect the metadata needed for writing the `dcm2bids` configuration file.

## 2.3 Inspection results

After running the inspection on each folder, the following series were identified:

#	Series Description	Protocol Name
1	Servey	Servey
2	sT2W_FLAIR	sT2W_FLAIR SENSE
3	esT2W_FLAIR_SENSE	esT2W_FLAIR_SENSE
4	sT1W_FFE	sT1W_FFE SENSE
5	esT1W_FFE SENSE	esT1W_FFE SENSE
6	DWI	DWI SENSE
7	sDWI SENSE	sDWI SENSE
8	s3DI_TOF_BM	s3DI_TOF_BM SENSE
9	s3DI_TOF_BM	MIP s3DI_TOF_BM SENSE
10	RL	RL
11	FH	FH
12	sT2W_TSE	sT2W_TSE_SENSE
13	esT2W_TSE SENSE	esT2W_TSE SENSE
14	sT1W_3D_FFE	sT1W_3D_FFE SENSE

These values will be referenced when defining the `criteria` section in the `dcm2bids` configuration file.

## 4. Creating the Configuration File for `dcm2bids`

After inspecting the DICOM data (Section 2), we can now create a configuration file for `dcm2bids`.

This configuration file maps each DICOM series to its corresponding BIDS modality, suffix, and metadata changes.

### 4.1 Notes on excluded series

Some Protocol ( `Servey` , `RL` , `FH` , `MIP s3DI_TOF_BM` ) are excluded because:

- Their purpose is unclear from the metadata.
- They may be localizer or positioning scans, which are not needed for most BIDS datasets.
- `sDWI` is not supported

## 4.2 Configuration file

The configuration file is written in JSON format, but for clarity I'll present it as a table. Each row specifies the BIDS datatype, suffix, the matching criteria (based on DICOM `SeriesDescription` ), and the optional sidecar changes applied during conversion.

Datatype	Suffix	Criteria	Sidecar changes	Note
anat	FLAIR	sT2W_FLAIR	sT2W_FLAIR SENSE	
anat	FLAIR	esT2W_FLAIR_SENSE	esT2W_FLAIR SENSE	
anat	T1w	sT1W_FFE	sT1W_FFE SENSE	
anat	T1w	esT1W_FFE SENSE	esT1W_FFE SENSE	
dwi	dwi	DWI	DWI SENSE	
dwi	dwi	sDWI SENSE	sDWI SENSE	lack of .bvec files, may violated the BIDS format
anat	angio	s3DI_TOF_BM	s3DI_TOF_BM SENSE	
anat	T2w	sT2W_TSE	sT2W_TSE SENSE	
anat	T2w	esT2W_TSE SENSE	esT2W_TSE SENSE	
anat	T1w	sT1W_3D_FFE	sT1W_3D_FFE SENSE	

The file should be saved as `dcm2bids.json` in your working directory.  
It will be passed to `dcm2bids` when running the conversion process in the next step.

## 5. Converting DICOM files to BIDS with `dcm2bids`

I use the Python API of `dcm2bids` directly instead of the recommended CLI commands:

```
from dcm2bids import dcm2bids_gen

app = dcm2bids_gen.Dcm2BidsGen(
    dicom_dir=[str(patient_dir)],
    participant=subject_id,
    config=str(config_path),
    output=str(bids_output),
    session="",
    clobber=True,
    force_dcm2bids=True,
    bids_validate=False,
    log_level="INFO"
)

app.run()
```

## 6. Packages the GUI application with `pyinstaller` :

### 6.1 Handling resource paths for `pyinstaller`

When packaging with `pyinstaller`, resources (e.g., images, icons, configs) are bundled inside a temporary directory.

To make sure the program can still find these files, we define a helper function `resource_path` that resolves paths correctly both during development and after packaging:

```
def resource_path(relative_path):  
    try:  
        base_path = sys._MEIPASS  
    except Exception:  
        base_path = os.path.abspath(".")  
    return os.path.join(base_path, relative_path)  
  
resource_path(path)
```

This ensures that `pyinstaller` can locate bundled files at runtime.

## 6.2 Building with `pyinstaller`

First, run `pyinstaller` with just the script name to generate an initial `.spec` file (which stores build options and can be reused/edited):

```
pyinstaller main.py
```

This creates a `build/`, `dist/`, and `main.spec`.

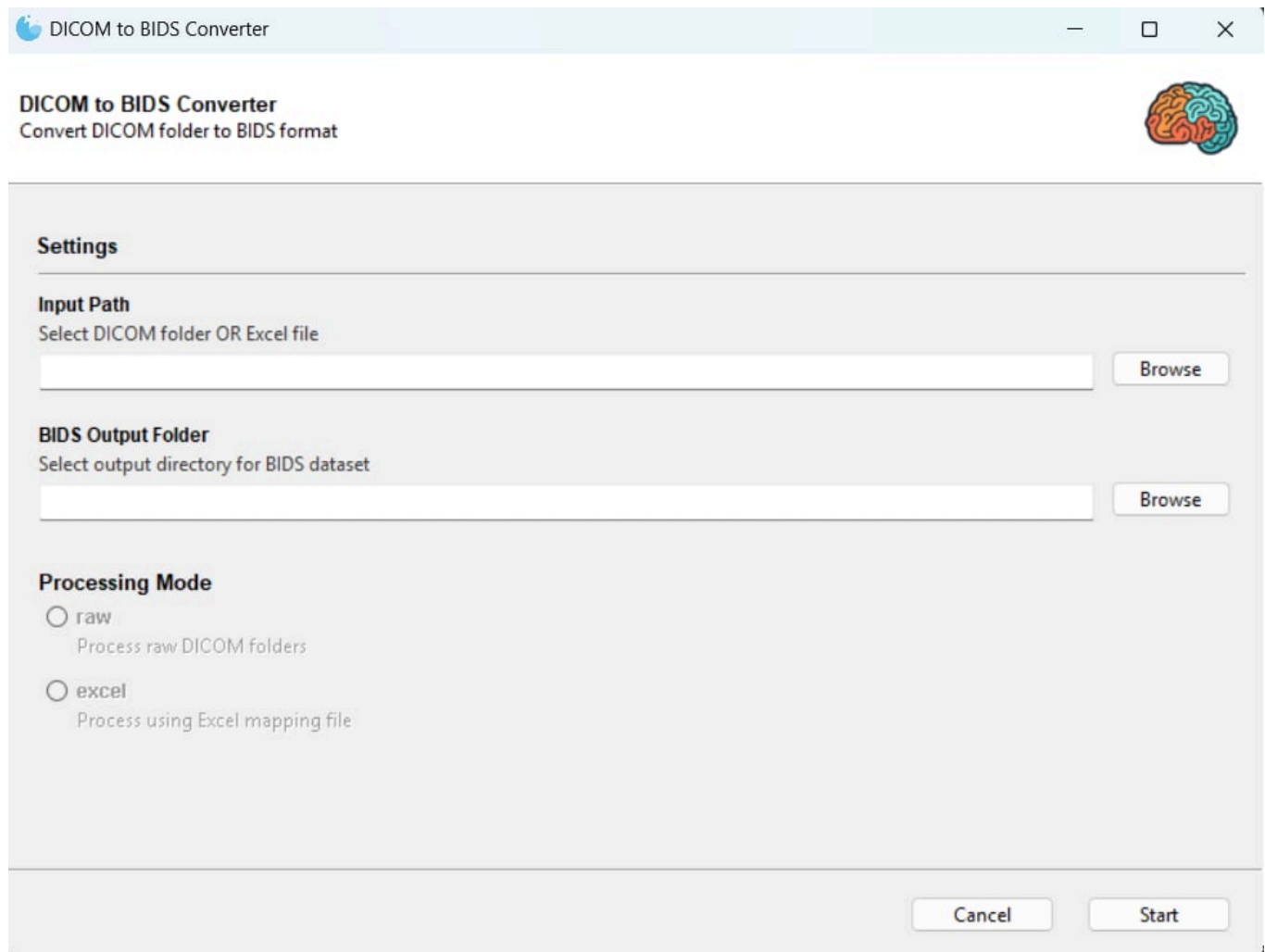
And then, to build a standalone, user-friendly executable with an icon and additional resources, use:

```
pyinstaller --name dcm2bids  
            --onefile --windowed  
            --icon=assets/brain.ico  
            --add-data "assets:assets"  
            --add-data "dcm2bids.json:."  
            main.py
```

- `--name dcm2bids` → Sets the output executable name.
- `--onefile` → Packages everything into a single `.exe` (or binary on Linux/macOS).
- `--windowed` → Prevents a console window from appearing (for GUI apps).
- `--icon=...` → Adds a custom application icon
- `--add-data` → Ensures external files/folders are bundled.

After the build finishes, the ready-to-use executable will be available inside the `dist/` directory.

Here is the UI:



The screenshot shows the 'DICOM to BIDS Converter' application window. The title bar reads 'DICOM to BIDS Converter'. The main window has a header with the application name and a brain icon. Below the header is a 'Settings' section. The 'Input Path' section has a text field and a 'Browse' button. The 'BIDS Output Folder' section has a text field and a 'Browse' button. The 'Processing Mode' section has two radio buttons: 'raw' (selected) and 'excel'. At the bottom right are 'Cancel' and 'Start' buttons.

**DICOM to BIDS Converter**  
Convert DICOM folder to BIDS format

**Settings**

**Input Path**  
Select DICOM folder OR Excel file

**Browse**

**BIDS Output Folder**  
Select output directory for BIDS dataset

**Browse**

**Processing Mode**


☒ raw  
Process raw DICOM folders

☐ excel  
Process using Excel mapping file

**Cancel** **Start**



# Choose the processing mode:

DICOM to BIDS Converter


—

□

×

DICOM to BIDS Converter

Convert DICOM folder to BIDS format



Settings

Input Path

Select DICOM folder OR Excel file

Browse

BIDS Output Folder

Select output directory for BIDS dataset

Browse

Processing Mode

☐ raw

Process raw DICOM folders

☒ excel

Process using Excel mapping file

Cancel

Start

Finish:

