Instructions for practical 1: Brain structure segmentation

N. Boutry J. Chazalon O. Ricou

Revised Sept. 2023

Introduction

This session is about segmenting core brain structures in 3D MRI brains scans of infants.

We will reuse the problem statement and data from the iSeg challenge held during the MICCAI 2017 conference: iSeg.

Quoting the authors of the original competition:

> The first year of life is the most dynamic phase of the postnatal human brain development, along with rapid tissue growth and development of a wide range of cognitive and motor functions. This early period is critical in many neurodevelopmental and neuropsychiatric disorders, such as schizophrenia and autism. More and more attention has been paid to this critical period.
>
> Accurate segmentation of infant brain MR images into white matter (WM), gray matter (GM), and cerebrospinal fluid (CSF) in this critical period is of fundamental importance in studying both normal and abnormal early brain development. Of note, there are three distinct phases in the first-year brain MRI, including (1) infantile phase ($\leq$ 5 months), (2) isointense phase (6-8 months), and (3) early adult-like phase ($\geq$ 9 months). In the isointense phase, the intensity range of voxels in GM and WM are largely overlapping (especially in the cortical regions), thus leading to the lowest tissue contrast and creating the most significant challenge for tissue segmentation, in comparison to images acquired at other phases of brain development.
>
> For example, [the figure below] shows longitudinal MR images for an infant scanned every 3 months during the first year, starting from the second week after birth. **At around 6 months of age, MR images show the lowest tissue contrast and create the most significant challenge for tissue segmentation.**

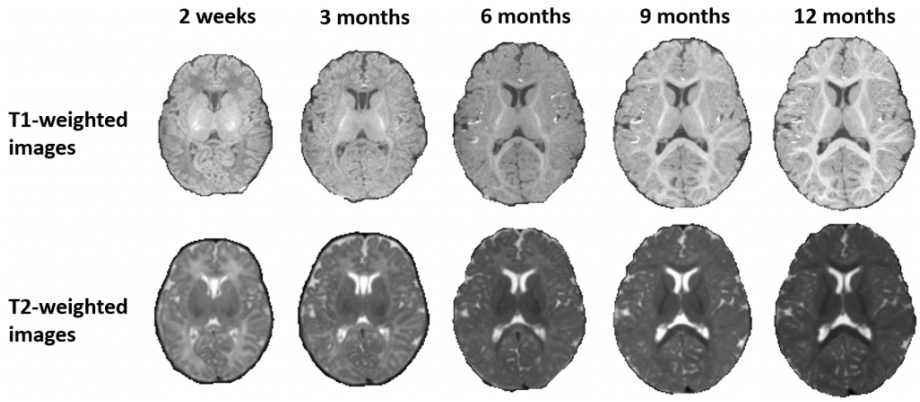

Figure 1: T1- and T2-weighted MR images of an infant scanned at 2 weeks, 3, 6, 9 and 12 months of age.

Dataset presentation and input data format

Training set

We provide you with a single ZIP file containing training images and manual labels. **Check Moodle for files/links.**

There are only 10 3D volumes available. This is both very little data and also a lot since these are 3D volumes. You need to exploit this data at best.

The zip file contains T1- and T2-weighted MR images of 10 infant subjects (named as `subject-1` to `subject-10`):

- `subject-?-T1`: T1-weighted image
- `subject-?-T2`: T2-weighted image
- `subject-?-label`: manual segmentation

Notes on the manual segmentation labels:

- `0`: Background (everything outside the brain)
- `10`: Cerebrospinal fluid (CSF)
- `150`: Gray matter (GM)
- `250`: White matter (WM)

Test set

The test set is composed of one zip file with testing images **but no associated labels**. The zip file contains T1- and T2-weighted MR images of 13 infant subjects (named as `subject-11` to `subject-23`).

You need to produce segmentation files saved in numpy array format:

```
# TEST is the path to the directory
```

```
# where you saved your outputs for the test set
predictions = {
    f'subject-{ii}':np.array(
        nib.load(f"{TEST}/subject-{ii}-label.img").get_fdata(),
        dtype="uint8")
        for ii in range(11,24)}
np.savez("/tmp/brain_labels_OR.npz", **predictions)
```

You MUST produce a file `brain_labels_XY.npz' where XY are your initials. The predictions for each subject must have the shape`(slice, row, col,)`, and`uint8‘ as data type. For each voxel, we expect one of the following values, exactly as in the training set:

- `0`: Background (everything outside the brain)
- `10`: Cerebrospinal fluid (CSF)
- `150`: Gray matter (GM)
- `250`: White matter (WM)

The previous code assumes you already saved your predictions for subject-{11..23} in files `subject-{11..23}-label.{hdr,img}`. This is not required but convenient. To do so, given the test set inputs, you need to be able to produce segmentation outputs in the same format as the label files in the training set. https://stackoverflow.com/a/41993044 contains some hints about how to write `.hdr` and `.img` file pairs using `nibabel`. You can copy the header and affine parameters from either the input volume or from other ones. You can also export the predictions directly, but debugging may be harder.

You can install `nibabel` with `pip install nibabel`.

Using `keras`, some useful imports may be:

```
from keras.layers import *
from keras.models import *
from keras.optimizers import *
```

What you have to do

You goal is to:

- understand and prepare a training and a validation set
- produce a system which produces the best possible segmentation on this data
- process the testing set
- upload your results to Moodle

Understand data

Open files (use the `nibabel` Python package) and print graphics to understand what we have.

```
# https://nipy.org/nibabel/gettingstarted.html

gt = nib.load(DATA+"subject-10-label.hdr").get_fdata()
t1 = nib.load(DATA+"subject-10-T1.hdr").get_fdata()
t2 = nib.load(DATA+"subject-10-T2.hdr").get_fdata()
```

- What kind of input are you going to provide to your network ?
- What kind of output do you expect from your network ?

If you cannot give a correct answer to these questions, don't expect to have an efficient network.

Prepare data

The most efficient here is to make a generator to provide slices to the model since 3D processing would be too heavy.

```
class SlicesGenerator(keras.utils.Sequence):

    def __init__(self, data, gt, batch_size=8):
        self.data = data              # input
        self.gt = gt                  # ground true
        self.batch_size = batch_size

    def __len__(self):        # how many batches for each period ?
        shape = data.shape
        return ... // self.batch_size

    def __getitem__(self, idx):  # return a batch of data/gound true
        batch_x, batch_y = [], []
        ...
        return np.array(batch_x), np.array(batch_y)
```

Do not forget to normalize your data.

Define your model

Use UNet segmentation architecture as describe in course (beware, some model on the Internet are not correct). You are going to need :

- `Input`
- `SeparableConv2D` (or Conv2D but let's try separable convolutions)
- `BatchNormalization`
- `MaxPooling2D`
- `Conv2DTranspose`
- `UpSampling2D`
- the `add` function to add former layer to current layer (or `Concatenate`)
- eventually, you may add some `Dropout` layers

- Use an adequate loss functions (Binary cross-entropy? Categorical cross-entropy? DICE loss?).

You can use the `Adam` optimizer with a moderate learning rate (`1e-2`).

To check you network is correct, there are plenty of options, like for instance:

- start with a network with few parameters (adjusting the filters in your convolution layers), which is fast to train
- make sure your network can overfit on a few training samples (this is suitable for larger network as well)

Hint: it is easier to train the network with a simple sigmoidal output a binary cross-entropy, and perform the softmax (or any aggregation then voting function) as a post-processing.

Train your model

Make sure you define training and validation(s) sets.

```
BATCH_SIZE = 16

train_gen = SlicesGenerator(..., ..., batch_size = BATCH_SIZE)
val_gen = SlicesGenerator(..., ..., batch_size = BATCH_SIZE)

nb_iter = 50
model.fit(train_gen, validation_data=val_gen, epochs=nb_iter, callbacks = callbacks)
```

- Make sure your architecture is capable to fit your data (no underfitting) but also generalizes correctly (no overfitting).
- Try to play with the number of layers, weights, regularization functions, etc. in your architecture.

Compure and optimize your results

- Always check your results visually.
- Compute DICE error
- Suggest a postprocessing method to improve results

Most important: start as simple as you can to get a baseline implementation as quickly as possible. Do not try to optimize before. Delivering something is always better that nothing.

Deliverables and grading

You need to submit your final predictions (assigning for each voxel of the test set a class, among the 4 possible ones).

The submission form is on Moodle. You only need to submit your `brain_labels_XY.npz` file, where XY are your initials.