BE 537 Homework Assignment 2 (2021)

Due on Canvas **November 15, 2021 at 3pm**

October 11, 2021

Contents

1 Overview

This assignment consists of a base component and one extension. In the base component, you will use Python and PyTorch to implement diffeomorphic deformable registration algorithm for 3D images based on the *Log-Domain Demons* algorithm by Vercauteren et al. (2008). After competing the base component, extend it by implementing **one** of the three extensions below:

- Construct an *unbiased population template*
- Automatically segment brain structures using *multi-atlas segmentation*
- Implement deformable registration using deep learning and variational auto-encoders (advanced)

1.1 Data

- The `images` directory contains 20 anonymized MRI images of the brain, which have been resampled with voxel size of $2.0 \times 2.0 \times 2.0\text{mm}^3$ and dimensions of $96 \times 116 \times 128$ voxels.
  - The images are named **atlas_2mm_[ID].nii.gz** and **target_2mm_[ID].nii.gz**. The distinction between the 15 “atlas” images and 5 “target” images will be relevant when you implement some of the extensions.
  - Each image is accompanied by a multi-label segmentation image, similarly to the first assignment. These segmentations assign different integer labels to about 140 different brain regions. The segmentations are named **atlasseg_2mm_[ID].nii.gz** and **targetseg_2mm_[ID].nii.gz**.
  - The images and segmentations are from the 2012 article *101 labeled brain images and a consistent human cortical labeling protocol* by Klein and Tourville.
- The `testing` directory contains some images to help you test functions in your code.

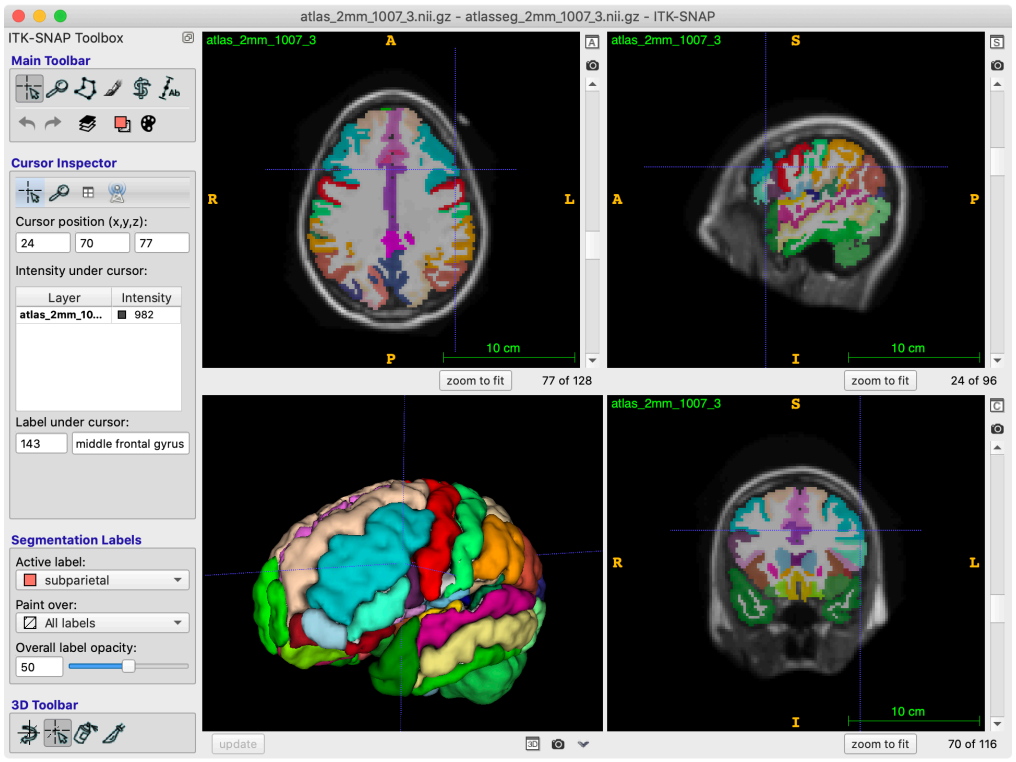

Figure 1: Example segmentation from the dataset

1.2 How to Format Your Assignment

- In the base assignment, you will fill in missing code and answer some theoretical questions in a **Jupyter Notebook** that is provided to you. You can use the free **Google Colab** service or your local Python environment.
- The extension is more open-ended and should be added as a new section in the above notebook.
  - The quality of your report for the extension (clarity of the writing, consistent formatting, clear labeling of the figures) will be factored in to the grading of the assignment. When generating plots and figures, be sure to use figure titles, axis labels, and legends, as you would in a scientific paper. The quality of the figures and plots is part of the grading criteria.
  - You don't need to write a lot of text in the extension, but include enough so that someone outside of our class could get a basic idea of what the notebook is doing. In other words, don't just include section headings with figures and code boxes, but also a few sentences explaining what you have done in each step.
- *Comment your code well.* The quality of your comments (clarity, relevance to the actual code) will be factored in to the grading of the assignment (both base component and extensions).
- Turn in the PDF of notebook, as well as the `.ipynb` file.

2 Base Component: Diffeomorphic Image Registration (70 points)

You will mplementing the non-symmetric version of the *Log-Domain Demons* algorithm by Vercauteren et al. (2008). Before the advent of deep learning image registration, the approach was arguably the most widely used deformable image registration approach because of its speed and ability to produce diffeomorphic transformations. The algorithm is described in Section 4.2.3 (Algorithm 4) of my **Deformable Image Registration** book chapter (on Canvas).

Implementing this algorithm will require the following components:

- **Scaling and squaring**, a technique that converts a smooth and bounded velocity field into a diffeomorphic transformation
- **Image gradient computation**, needed to compute optical flow between fixed and moving images
- **Gaussian smoothing**, used to regularize velocity fields and displacement fields

- **Optical flow computation**, used to compute residual displacement between the fixed image and the transformed moving image at each iteration of the registration algorithm.

To implement the base component, follow along in the provided notebook `Assignment2_base_template.ipynb`. All the instructions are in that notebook. You will need to answer several theoretical questions and fill in the code for several Python functions.

3 Extension (30 points)

This portion of the assignment is more open-ended. You have three options: - Use your registration code to build an unbiased population template - Use your registration code to perform multi-atlas segmentation - Implement a deep learning registration pipeline (challenging)

3.1 Option 1: Unbiased Population Template

Build a template from the first 10 "atlas_XXX" input images (you can use more, but it will take longer). Recall from class that building such a template involves iterating between registration and intensity averaging. The overall algorithm is quite simple:

1. Average the intensities of all untransformed input images to obtain the initial template
2. Register all input images to the current template (with template as fixed, input images as moving)
3. Apply resulting transformations to the input images and average their intensities to obtain the updated template
4. Repeat steps 2-3 for some fixed number of iterations (e.g. 5)

Since the affine registrations between images are given to you, you can apply them to all the atlas images before starting the template building process. For example, you can affinely transform all your images into the space of subject `1000`, and then build the template using deformable registration.

In addition, after each iteration, apply the transformations computed in step 2 to warp the anatomical segmentations of the atlas images into the template space and combine them into a **consensus segmentation** of the template. A function for combining multiple segmentations using majority voting is provided to you: `my_majority_vote`. This function assigns to each voxel the label that appears at that voxel in the largest number of input segmentation images. For example, if there are 7 inputs and they assigned voxel x the labels 1,2,2,7,9,2,9 then majority voting will assign the voxel the label 2.

Hints:

- Use `RegistrationExperiment` to help with loading image data. You can use subject `1000` as "fixed" and each of your input images as "moving".

3.1.1 Deliverables

In your report include the following:

- Plot the template using `my_view` after each iteration. Also plot the template segmentation.
- Plot the average mean squared intensity difference metric between the template and the input images vs. iteration. Does the metric go down as you iterate?
- Plot the average GDSC between the template segmentation and transformed input segmentations vs. iteration (use `my_generalized_dice`). Does the GDSC increase as you iterate?

3.2 Option 2: Multi-Atlas Segmentation

Multi-atlas segmentation is a powerful segmentation technique that is a simple extension of deformable registration. We have a set of images for which segmentations exist. We call these images "atlases". Suppose we are given a new image, which needs to be segmented. Call this image the "target".

In multi-atlas segmentation, we

- Separately register each of the atlas images to the target image (target used as fixed, atlas as moving)
- Apply the resulting (affine & deformable) transformations to the atlas segmentation images
  - Think of each transformed atlas segmentation as a "weak" guess at the segmentation of the target image.

- Combine the "weak" segmentations into a single "strong" or "consensus" segmentation
- Many techniques for combining segmentations have been developed. We will discuss some in later class. For now we will use the simplest possible technique, called majority voting. It is implemented in `my_majority_vote` function.

Implement multi-atlas segmentation using 10 images as atlases (`atlas_XXX`) and test it on 5 target images (`target_XXX`). Since the target images also have "ground truth" segmentations, you can test how accurate your automatic segmentation is relative to the gold standard of manual segmentation.

Hints:

- Use `RegistrationExperiment` to help with loading image data.

3.2.1 Deliverables

Include the following in your report:

- Plot "strong" and "ground truth" segmentations that you obtain using `my_view` with a colorful colormap (e.g., `jet`).
- Compute the average GDSC between "weak" segmentations and the known "ground truth" segmentations of the target images. Report the average GDSC across all atlas/target pairs. Compare this with the GDSC between the "strong" segmentations and corresponding "ground truth" segmentations. How much accuracy do you gain from using multi-atlas segmentation?
- Compute the average GDSC ("strong" vs. "ground truth") if only the affine part of the registration is performed. How much do you gain from using deformable registration?

3.3 Option 3: Deep Learning Registration Network

This extension is more advanced and assumes that you have some familiarity with training neural networks in PyTorch. In this extension, you will train a deep learning algorithm to generate spatial transformations given a pair of fixed/moving images as input. It will be similar to the VoxelMorph algorithm, but will utilize a variational autoencoder (VAE) just to make things more interesting and to make it possible to sample from the distribution of spatial transformations.

- Note: you need to have access to a GPU (e.g., Google Colab) to complete this assignment

3.3.1 Network Architecture

Below is a diagram of the network that we are asking you to implement. At training time, pairs of fixed/moving images are input into the network. This is implemented as a 3D image with two channels, the first channel containing the fixed and the second containing the moving image. A 3D deep convolutional decoder is used to non-linearly map the image data into a mean vector μ and a standard deviation vector σ. These vectors are of much lower dimension d than the images themselves, e.g., $d = 256$ or $d = 512$. A vector of latent variables z is sampled from the multivariate normal distribution $N(\mu, \Sigma)$ where the covariance matrix Σ is a $d \times d$ diagonal matrix with diagonal entries σ_i^2. The vector z is input to a deep convolutional decoder network that contains multiple upsampling layers and generates as an input an image of the same dimensions as the fixed and moving images and containing 3 components. This 3-component image is smoothed and scaled-and-squared (as in your log-Demons implementation), producing a displacement field ϕ. The moving image is deformed using ϕ and compared to the fixed image using the standard mean square error loss. This loss captures how well the ϕ generated by the network matches the fixed and moving images. Additionally, a Kullback-Liebler divergence (KL) loss is used to make sure that the outputs of the decoder are normally distributed in the latent space. The KL loss measures the KL divergence between $N(\mu, \Sigma)$ and the unit normal distribution $N(\mathbf{0}_d, I_{d\times d})$. Overall, this network architecture is similar to a standard VAE, but the difference is than in stadard VAE the objective of the decoder is to generate an image similar to the input image, and here the objective of the decoder is to derive a deformation field that matches the fixed and moving images.

At test time, we will give our network a fixed/moving image pair and it will generate a deformation field ϕ. Due to the stochastic nature of the network, running the network multiple times with the same inputs should generate different deformation fields. It will be interesting to explore if these deformation fields are consistently good at matching the fixed and moving images.

3.3.2 Implementation

To help you implement this network, we provided some helper code in the python module `be537hw2_dl.py`. This includes a data loader for the dataset in our assignment. This takes care of generating fixed/moving image pairs downsampled to dimensions of $64 \times 64 \times 64$ with optional data augmentation (in the form of affine transformations applied to both images). To use this data loader use the code below:

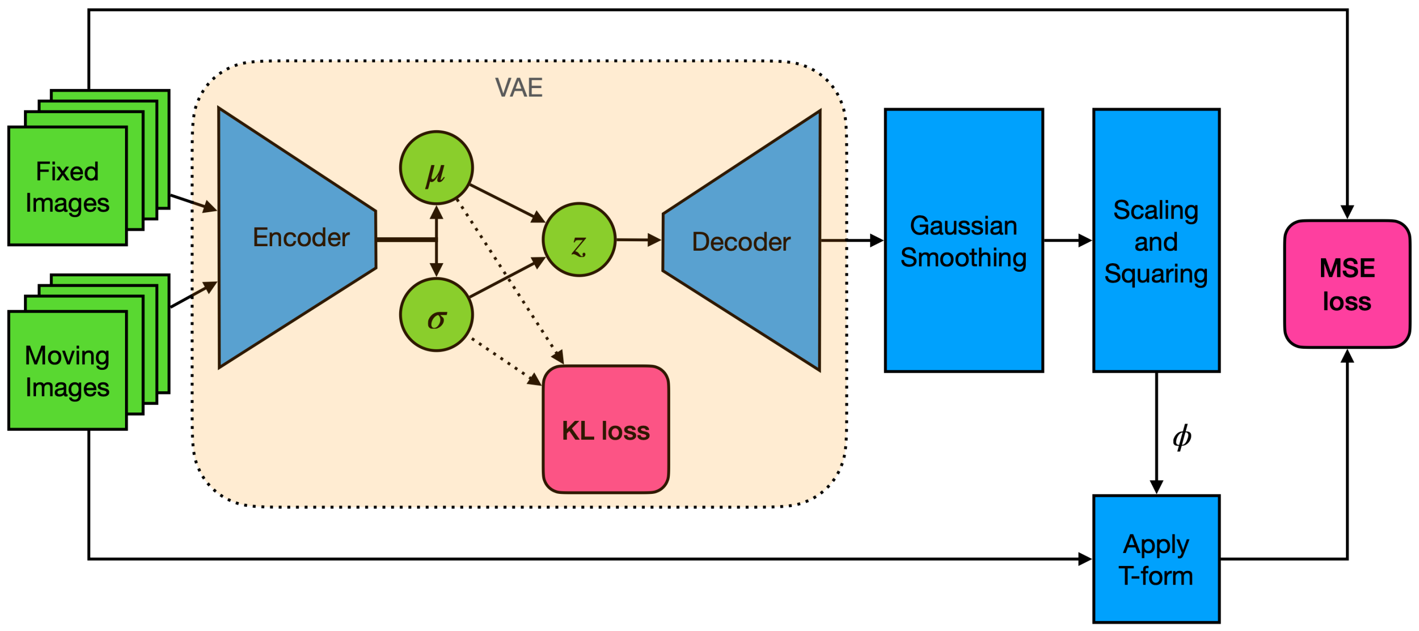

Figure 2: Network architecture

```
# Load the module for the assignment
from be537hw2_dl import *

# Generate a dataset for training. Augmentation is performed by applying random affine
# transformations to the fixed and moving images. Image intensity is normalized to 0-1 range.
ds_train = AssignmentTwoDataset(os.path.join(data_path,'images/atlas_*.nii.gz'),
                                transform=transforms.Compose([
                                    PowerOfTwoCenterPad(6),
                                    RandomAffineImage3D(sigma_log_scale=0.03, sigma_skew=0.05),
                                    NormalizeIntensity()]),
                                dtype=torch.float32, device=cuda)

# Generate a dataset for testing. No augmentation is performed.
ds_test = AssignmentTwoDataset(os.path.join(data_path,'images/target_*.nii.gz'),
                               transform=transforms.Compose([
                                    PowerOfTwoCenterPad(6),
                                    NormalizeIntensity()]),
                               dtype=torch.float32, device=cuda)

# Create training and test dataloaders
dl_train = torch.utils.data.DataLoader(ds_train, batch_size=10,shuffle=True, num_workers=0, collate_fn=my_collate_fn)
dl_test = torch.utils.data.DataLoader(ds_train, batch_size=4,shuffle=True, num_workers=0, collate_fn=my_collate_fn)

# Sample from the training dataloader
q=next(iter(dl_train))
my_view_multi(q['image'], q['hdr'][0], layout='C', figwidth=16, axis_visible=False)
```

In addition, we provide a VAE implementation, adapted from the CNN-VAE project. You can create a VAE with zero weights and apply it to a mini-batch of fixed/moving images as follows:

```
vae = VAE(channel_in=2, channel_out=3, ch=16, z=64).to(cuda)
decoded, mu, logvar = vae.forward(q['image'])

my_view_multi(decoded, q['hdr'][0], layout='A', axis_visible=False, figwidth=16, cmap='jet',
              title='Output from the VAE')
```

Note that the parameter z affects the number of dimensions d of the latent space. The output of `vae.forward` consists of the decoder output `decoded`, of dimension $[N, 3, 64, 64, 64]$, where N is the mini-batch size, the mean vector `mu` corresponding to μ above, and the vector `logvar` (logartihm of the variance), which is related to σ above as `logvar[i]` $= 2 \ln \sigma_i$. In this VAE implementation, these tensors are of dimensions $[N, z, 2, 2, 2]$ and the last four dimensions should be flattened to create vectors of size $[N, 256]$ or $[N, 512]$, depending on z.

What you will need to do is:

1. Create a new PyTorch network class (a class that inherits from `torch.nn.Module`) and that implements the full network (with Gaussian smoothing and scaling and squaring) as illustrated above. The class should have methods `__init__` and `forward`.
2. Create the KL and MSE losses. The MSE loss is available in PyTorch (`mse = torch.nn.MSELoss(reduction='mean')`) and the KL loss simply implements the formula

$$\frac{1}{2} \sum_{i=1}^{d} \sigma_i^2 + \mu_i^2 - 1 - \ln(\sigma_i^2)$$

3. Train the network using the Adam optimizer. Suggested hypeparameters are:
   - Learning rate for Adam: 0.001
   - Relative weights of the MSE and KL losses: 1.0 and 0.001
   - Number of epochs: 200 (for mini-batch of size 10)
4. Test the network on test data

3.3.3 Deliverables

- Plot the average KL, MSE and total losses over training epochs
- Report the average KL, MSE and total loss on the test set after training
- Plot example registration results for test data
- Compare the deformation fields generated when applying the network multiple times to the same fixed/moving image pair. Does the network achieve our goal of generating a stochastic family of deformations between the fixed and moving images? Are the stochastically sampled deformations equally good at matching the images or are some better than others?
- Do images cluster in the latent space? Compute the average Euclidean distance between the latent space representations (i.e., μ vectors) of two fixed/moving pairs that share the same fixed or moving image (e.g., pairs `1001/1002` and `1005/1002`) and compare statistically to the average distance between latent representations of two pairs that have no images in common.