ECE 4250: Final Report

Name: Kranthi Kumar Madhavaram
ID: km853
Kaggle ID: kranthi017

Milestone 1

For Milestone 1, I have used the nibabel library to load in the images; both the regular images and manual segmented image of the training and validation data. This was done by the simple function nib.load('filename'). Once loaded, I have extracted the X-dimension, Y-dimension and slicing of the 6 testing images and two validation images from the header file which is shown below.

```
  X-dim      Y-dim    slicing
[[0.9375    0.9375     1.5      ]
 [0.9375    0.9375     1.5      ]
 [0.9375    0.9375     1.5      ]
 [0.9375    0.9375     1.5      ]
 [0.9375    0.9375     1.5      ]
 [0.9375    0.9375     1.5      ]
 [1.        1.         1.5      ]
 [0.8370536 0.8370536  1.5      ]]
```

The middle coronal slices from the image were then extracted by simply checking the array at the 64th location at the 3rd dimension of the each image. Few of these extracted images can be seen below.

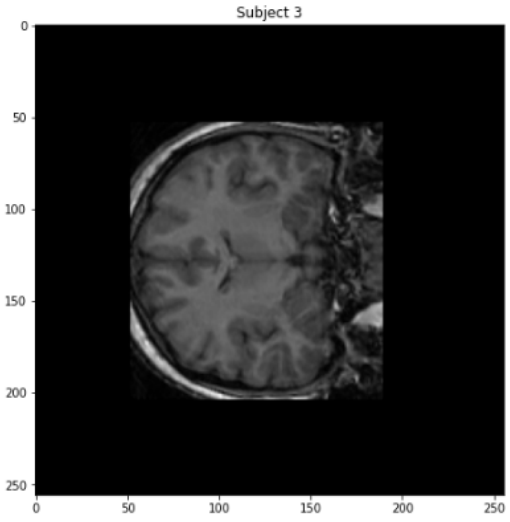


Fig: subject 3 MRI

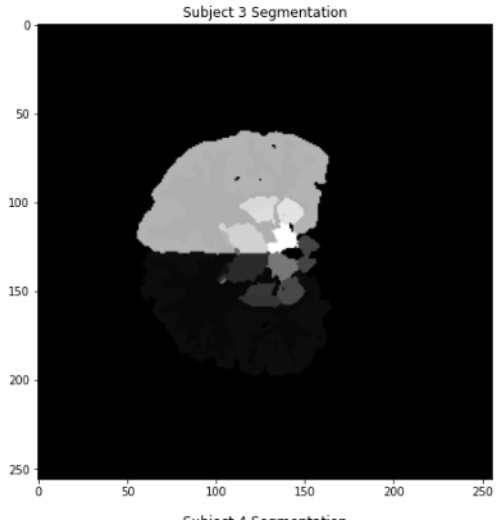
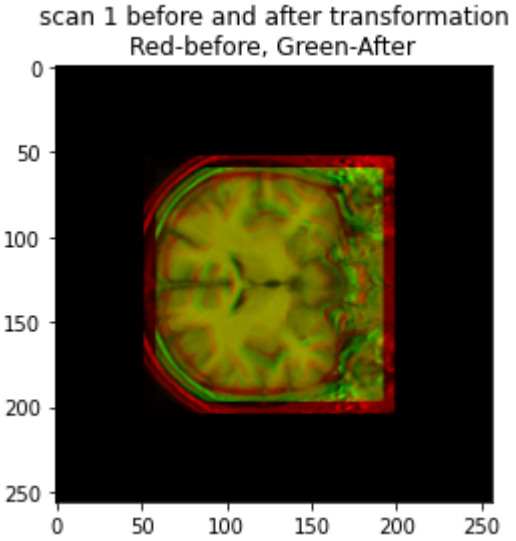


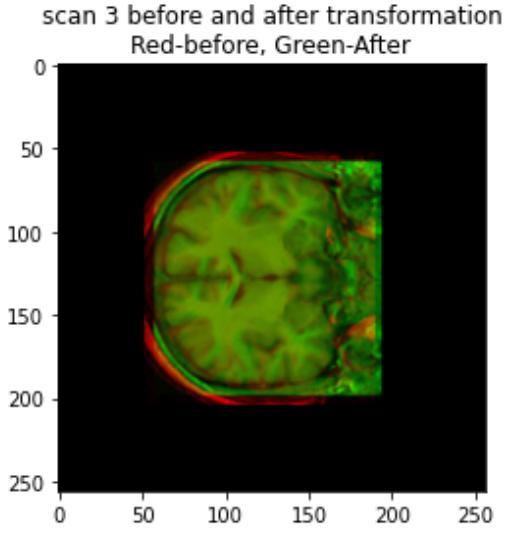
Fig: subject 3, segmentation image

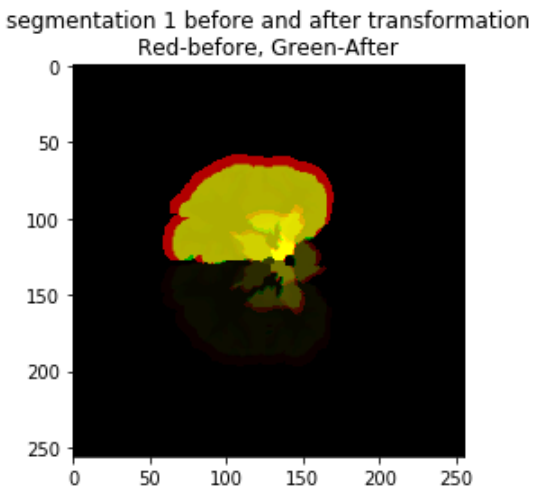
Milestone 2

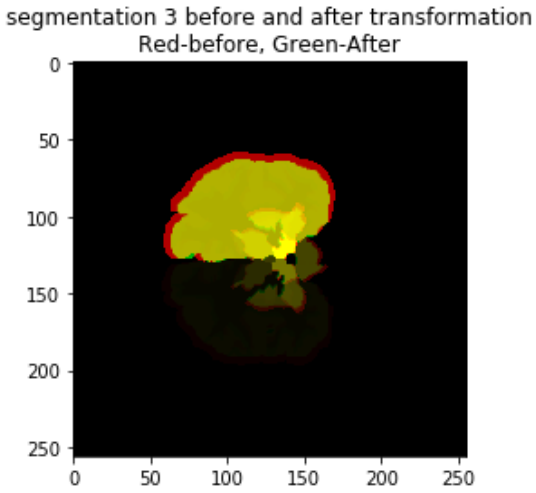
As for Milestone 2, I have started off by writing the functions for registration, loss and optimize. Registration function takes in six parameters: scale, angle, move_x, move_y, interpolate, and image. This function primarily uses openCV library to scale the given image by the value 'scale', then rotates it by the given angle and translates it along x and y direction given by the scalar values move_x and move_y. The 'interpolate' decides the interpolation method to be used during scaling. The loss function takes in two images (fix and moving), and the parameter array as inputs. It then calls the registration function on the moving image with the given parameters and sums the squared difference at each pixel between the two images and returns it. This gives the similarity measure between a fix and moving image. The optimize function takes in 3 inputs; parameters array, interpolate, images. The images are split into fixed and moving inside the function. Using scipy's minimize function in optimize package, I use 'loss' function to be optimized here by changing the parameter array (scale, angle, move_x, move_y) and set the method initially to Nelder-Mead.

Calling the optimize function on each of the training image as the moving image with each validation image as the fix image gave me (6*2) X 4 parameter array. Using these parameters I then called the registration function on the segmentation images of the training set. Below are a few visualizations.

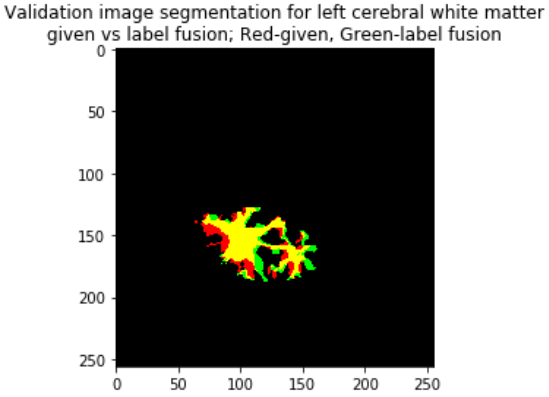


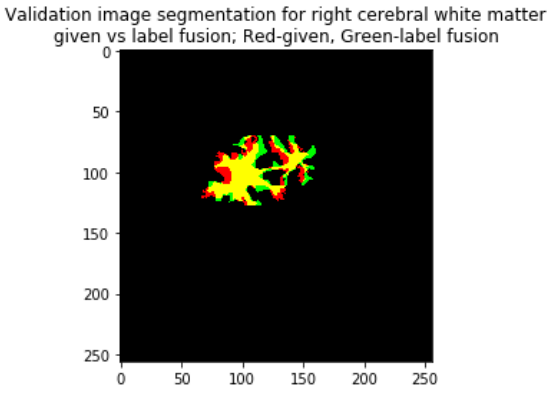






I then used the label fusion of finding mode of the 6 training segmentation images for each validation image at each pixel to find the automatic segmentation images. Comparing them with the manual segmentation images of the left and right cerebral white matter is shown below.





Calculating the Jaccard Index for each of the validation image for the four Region of interests is given below:

```
[[0.55607843 0.53966597]
 [0.50728073 0.3787234 ]
 [0.54044816 0.55881539]
 [0.49737854 0.41757286]]
```

Final Competition

Test 1

For initial testing I just used the code from the milestone 2 without any changes i.e, I simply used the 6 training images and found the best parameters using nelder-mead method wrt each of the 8 testing images. Then applied these parameters to the segmentation images of the training images to get a total of (6*9) = 54 images. Then did fusion label by finding the mode of the six images for each testing image and converted to RLE encoding and saved in the CSV file and submitted.

Test 2

Here, I increased the number of training images to 8 by adding the two validation images as well. This improved the accuracy significantly. In general, increasing the training data more often than not improves any optimization function.

Test 3

Here, I have introduced weighted label fusion to the algorithm and used it for the rest of the tests hereon. For this I stored the loss of the final optimized parameters of the training data wrt the testing images and used them as weights.

Test 4

Here, I realized using the loss values for weights doesn't improve performance because higher the loss it means the image is less similar. And so, I used the reciprocals of the losses as weights.

Test 5

Here I have changed the interpolation technique to Inter_area which yielded bad results.

Test 6

Here I have changed the interpolation technique to Inter_linear instead of nearest neightbor. Realised, nearest neighbor yields the best results.

Test 7

Here I have changed the registration function for the first time. I added logpolar transformation to the end of registration function after scaling, rotation, and translation. This, however, yielded losses over

double than the previous methods yielded. Also, plotting the transformed image didn't even look close to the original image. Hence, I didn't even bother to upload the final output.

Test 8

Here, I have changed the optimization function to Powell. This took over 3 times the computation time required by Nelder-Mead method and output loss values lower than before. However, the accuracy decreased, leading me to believe the model must've overfit the training data.

Test 9

Here, I used COBYLA optimization for the minimize function. This yielded similar results to Nelder-Mead optimization.

Test 10

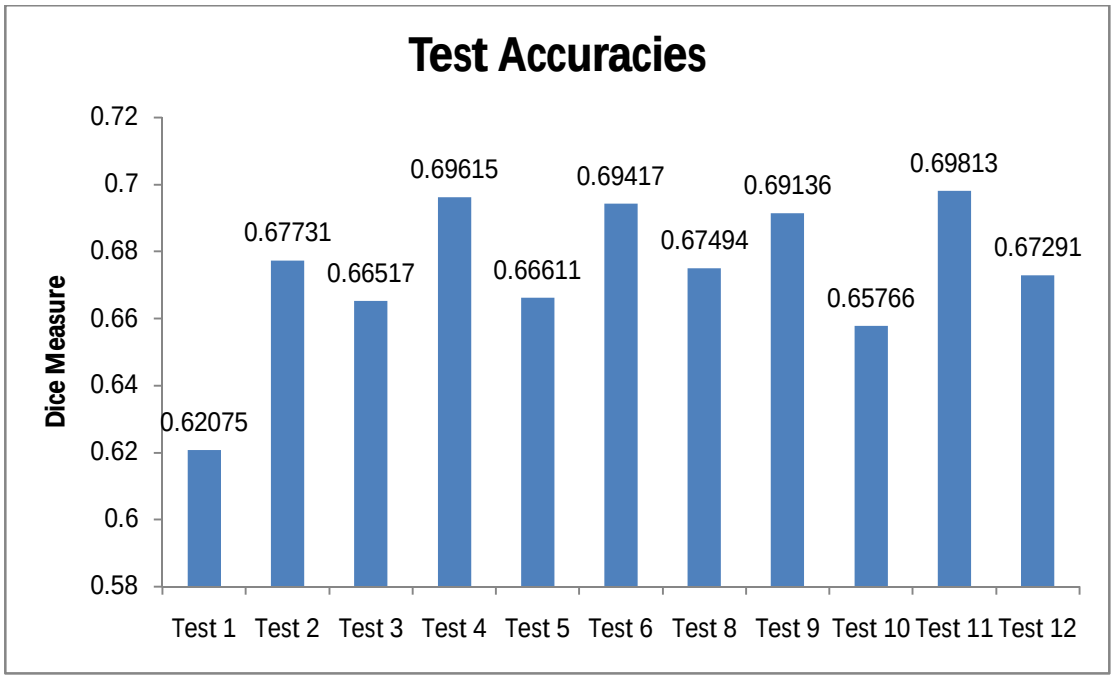
Here, I used COBYLA method with linear interpolation. However, it yielded lower accuracy.

Test 11

Here, I used Nelder-Mead method for optimization with patch label fusion. For this at each pixel I took the mode of a 3x3 patch around it and then found the weighted mode among the 8 training images.

Test 12

Similar to Test- 11, but I gave weights to the 3x3 patch and found the weighted mode of the (8*9) pixels for each pixel in the grid.



Conclusion

In the above test methods I have used various methods including but not limited to changing interpolation method, optimization functions, adding transformations to registration function. Not included in the test cases was the use of openCV's ORB (Oriented FAST and Rotated BRIEF) method for image registration. Here, it finds dominant features of each image and compares them between two images using the best features and finds the best orientation for the images to overlap. However, I couldn't resolve the errors faced when using this function and hence had to give it up. However, the best results I found were when I increased the training data to 8 by including the validation images, used Nelder-Mead for optimization, used a combination of patchwise and weighted mode label fusion.

Ethical Considerations

The project primarily targets automation of MRI segmentation and heavily relies on training images which are manually segmented first and then trained to the test subjects MRI scan and outputs an automated segmentation file for the test subject. Although different training methods and parameters are developed and tweaked to improve the performance, the underlying factor of manual segmentation is ignored. It is clearly known that the manual segmentation itself can differ for the same subject based on the person (usually an expert in the field) segmenting. Moreover, the same person can produce different segmentation for the same subject when done at different timestamps. This process of choosing the expert to define the training data can hugely impact the final output software making it not so "automated" in the end and leaves space for people to exploit and put their personal belief of different segments of a brain (say, cerebral white matter). Therefore, ethically, to keep it uniform it is ideal to have rigid protocols for even manual segmentation.