Ryan McMahon

5/15/20

Kaggle ID/Team Name: ryanmcmahon2/Ryan McMahon

netID: rm756

ECE 4250 Final Project Report

**Introduction:**

In this project, our goal was to create an algorithm that can automatically segment brain MRI scans into anatomical regions of interest. We were given 17 T1-weighted brain MRI volumes, of which six served as training data, two were for validation, and the remaining nine were for testing. We used the training and validation data to develop and test our algorithm, and then applied the algorithm to the test data. Ultimately, the goal was for our algorithm to produce segmentations for specific regions of interest for each of the nine test subjects that were as close to the manual segmentation as possible.

**Background:**

In image processing, an image is a collection of pixels, each with its own intensity value. However, only looking at the pixel intensities in a certain image generally won’t give us much useful information. Using brain MRI scans as an example, a doctor can look at an MRI image and gather useful information from it, such as the location of certain parts of the brain or tumors. This process, where an expert partitions the brain image into different sections and assigns a label to each section, is called image segmentation. After image segmentation is completed, each pixel (or voxel if we are looking at a 3D scan) will have a label that tells us what anatomical region of the brain that pixel belongs to.

There are many issues with manual segmentation that motivate creating automatic brain segmentation algorithms. First, a human, typically a neurological expert, is needed throughout the process to identify all the regions of interest. This means that the results of segmentation can change based on the interpretation of each person and the tools they use. Also, the process is very expensive and time consuming. With an automatic segmentation algorithm, we could improve the reliability and consistency of the segmentation results, as well as reducing the time and resources required to produce a segmentation.

Since we were given eight MRI volumes that already included manual segmentations, we also needed to use image registration, which is the process of aligning one image with another. Once we had the training image aligned with the testing image, we could then use the manually segmentation results to predict the segmentation for the testing image. This means that our algorithm consists of two parts: trying to line up the two images as closely as possible, and then using knowledge about the manual segmentations to try and predict the segmented test image. We quantified the results by using either Dice or Jaccard overlap scores, which both are measures to compare the amount of overlap between a region of interest in our predicted segmentation and the actual result.

**Methods:**

For the second milestone of this project, my image registration consisted of three parts. First, I wrote a transformation function that accepts four parameters (scale, rotation angle, and translation in both directions), and then transforms the input image accordingly. I used NumPy arrays and operations (Van der Walt 2011) to construct a rotation matrix and do the calculations, as well as SciPy RegularGridInterpolator (Oliphant 2007) for pixel interpolation. I also wrote a loss function that returns the sum of squared errors between two images. Finally, I created an optimization function that tried to find the transformation parameters minimize the loss function described earlier. For this, I once again used the SciPy optimize module. The overall process was then to start with an initial guess of parameters that would align one image with the other, and then the optimization module would continue to iterate until the “best” transformation was found (based on the loss function).

My image segmentation strategy was initially majority voting label based fusion. Once I found the optimal transformation from each training image to each validation image, I would apply the same transformation to each training image segmentation. Then, for each pixel in the image, I would assign the most frequent label from the six transformed training images to the predicted segmentation. I used the SciPy stats module to compute the mode of the six labels. The results of my initial algorithm can be seen in **Table 1**. In addition, I used min-max scaling, which scales the pixel intensities between 0 and 255, to improve the registration results.

The first adjustment I made to this was to add additional parameters to the registration algorithm. I initially added three additional parameters that affected the scale and shearing of the image, but this took significantly longer, and the results were usually not much better than with one universal scaling parameter. However, when I added just one additional scaling parameter (one for x direction, one for y direction), the results were much better. This was because some images were taller or shorter than others, so adding an extra parameter to account for that had a clear benefit. I also tried using histogram equalization to improve registration results, but it didn’t lead to any significant improvements.

I also tried using a weighted fusion approach, as described in “Optimal Weights for Multi-Atlas Label Fusion” (Wang 2011). One approach these authors described was rather than simply using majority voting, we could assign different weights to each image based on how similar they are to the test image. I did this by writing a function that iterated through every pixel in the test and training images and compared the intensity values at each pixel, added together these differences using some method, and then summed up the differences. Then, I used the inverse of this difference as the weights for each predicted segmentation pixel, rather than just using a normal mode. I used Scikit-learn weighted\_mode to compute the weighted mode (Pedregosa 2011). If an image wasn’t aligned with the test image very well, then it would have a lower weight. This meant that the segmentation results have less of an influence from poorly registered images.

Another approach I used was discussed in “Reliability-Based Robust Multi-Atlas Label Fusion for Brain MRI Segmentation” (Sun 2019). In this paper, they talked about locally weighted voting, where we determine the weights on a pixel-by-pixel basis. At each pixel, we calculate the similarity of neighborhoods in the training and testing images, and then assign higher weights to more similar neighborhoods. In this method, I found that using 3x3 or 5x5 neighborhoods was ideal, because anything smaller didn’t produce good results, and anything larger took very long to run. After I implemented majority voting fusion, weighted fusion, and locally weighted voting, I experimented with segmentation algorithms that combined elements from one or more of the previous methods. Based on the transformation algorithm I was using at this point, I found that using my weighted fusion algorithm and comparing 3x3 neighborhoods produced the best results.

After this, I made a few adjustments to improve my algorithm. First, I changed my optimization function so that the initial guess was a parameter. I found that if I put in a guess that was close to the final parameters, then the optimization module returned more quickly. Also, I repeated the optimization for images where the transformed image didn’t align well with slightly different initial guesses, and this also produced better registration results. I also experimented with the segmentation, but none of my approaches yielded much better results. Since the registration seemed to be where most of the potential improvements could be made, I researched available libraries that support image registration. I decided to use Dipy (Garyfallidis 2014), because it is specifically designed to be used for MRI images, and because it supported both affine and non-linear transformations. I used the SymmetricDiffeomorphicRegistration class, which uses the Symmetric Normalization algorithm. This algorithm creates a Gaussian pyramid with the specified number of levels, and a different resolution at each level. Each level has a certain maximum number of iterations, and the algorithm then uses these parameters to start the optimization. I found that using the CCMetric, which uses cross correlation between two images as the value we are trying to minimize, produced better results than the sum of squared errors for this type of registration. I also found that using three levels produced good results. Not having enough levels meant the registration wasn’t accurate, and adding more levels sometimes made the results somewhat distorted.

Once I implemented non-linear transformations using the Dipy library, I noticed that my registration results were very accurate, except for a few small errors around the edges of the brain. To fix this, I used a modified version of min-max scaling. Since many of the pixels in the image are 0 (because of the background), using histogram equalization alone didn’t produce as much contrast in the brain image as possible. I used a modified version of histogram equalization that ignored the lowest intensity values when doing the calculation, which meant the contrast between pixels was greater, so the registration results were better. I also found that increasing the number of levels in the Dipy registration function fixed these issues on the boundary of the brain images. Finally, I created a segmentation algorithm that was a hybrid between majority voting, locally weighted voting and weighted fusion. In this function, I first computed the global weights based on similarity between the entire images. This step reduced the contribution of training images with poor registration results, which then improves segmentation results. Next, for each pixel in the image, the function looked at the labels for neighborhoods for each of the six training images and computed the weighted mode to get the label for the output segmentation. I experimented with several different sized neighborhoods, and eventually I found that using a 4-neighborhood and doubling the weight of the central pixel produced the best results.

For my highest scoring submission, I used the Dipy library to determine the optimal non-linear transformation between each training and testing image (minimizing the cross-correlation), and then used a hybrid of weighted fusion and locally weighted voting for image segmentation. I found that this approach worked very well because it produced very accurate registrations, and then took advantage of these accurate registrations during the segmentation process. Throughout the project, I also used these libraries: Matplotlib (Hunter 2007) to visualize images, pandas (McKinney 2010) to convert the binary images into csv format, and Nibabel (Brett 2020) to load the MRI volumes.

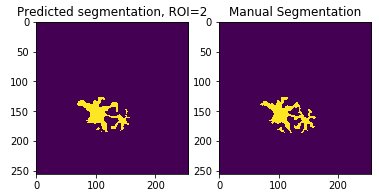
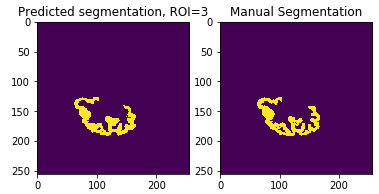
**Results:**

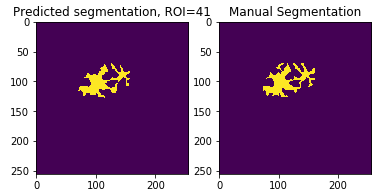
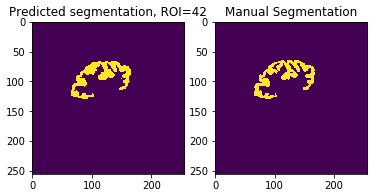
In the below table, ROI of 2 represents left cerebral white matter, 3 represents left cerebral cortex, 41 represents right cerebral white matter, and 42 represents right cerebral cortex. Milestone 2 shows the results from my submission to Milestone 2 as described above without any changes. Weighted fusion were the results when I started using weighted fusion as the label fusion strategy. Add parameters shows the results once I added an extra parameter to the transformation function and settled on locally weighted voting as the best label fusion strategy. Non-linear shows the results when I used the Dipy non-linear transformation for registration, and the final submission was when I added histogram equalization and used a hybrid segmentation algorithm. Final submission is the results for my highest scoring Kaggle submission, as described above.

****

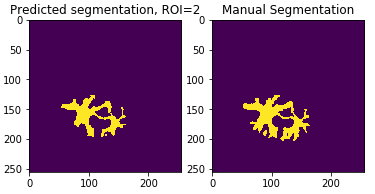
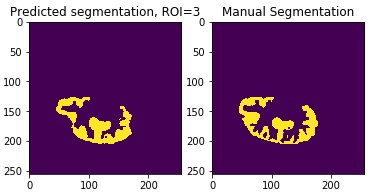
As can be seen, implementing new fusion strategies slightly improved my algorithm’s overlap scores, but the largest gains came from improving image registration by using non-linear transformations. Below, I have included visualizations of my validation segmentation results using the final submission method I described earlier and examples of image registration.

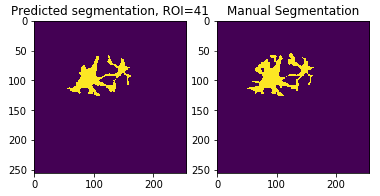
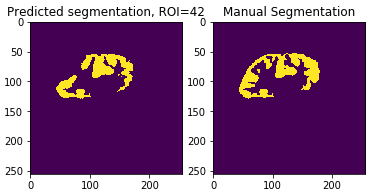
**Validation Image 1/Subject 7:**

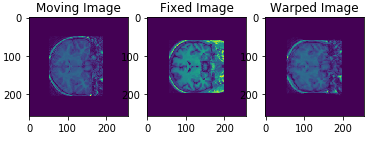
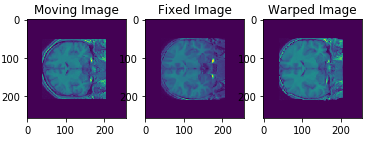
 

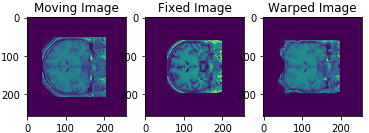
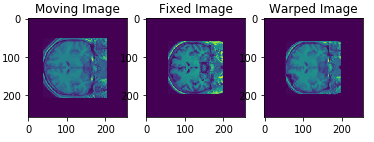
**Validation Image 2/Subject 15:**

**Selected Registration Results:**

**Discussion:**

In this project, I was able to implement an algorithm that produced image segmentations with relatively high accuracy. This was done by splitting the problem into three parts: preprocessing, image registration, and label fusion. For preprocessing, I loaded the MRI volumes, selected the middle coronal slice, scaled pixel intensities between 0 and 255, and used histogram equalization, which improved the overall accuracy of segmentation. Although I initially used SciPy’s optimize module and wrote my own transformation function initially for registration, I found that available functions in the Dipy library were much more efficient and led to better results. However, as seen in the last line in **Selected Registration Results** above, the diffeomorphic transformation didn’t always line up super well, but I was able to fix this (as seen in the second image in that line) by adjusting the number of levels and other parameters. Finally, I considered several segmentation methods, as described in the papers cited below, and eventually settled on a hybrid that considers the global similarity between each training and validation/test image in addition to the labels in each neighborhood of all six segmented training images.

One important lesson I learned was how important it is to research and have a plan before beginning to write any code for a project like this. Once I researched available Python libraries and looked at research papers that described label fusion techniques, it became much easier to implement automatic segmentation that produced better results. All the code would’ve taken a lot longer to write if I hadn’t used these libraries. Also, I was surprised by how accurate my algorithm was considering it wasn’t super complicated, which I shows how powerful the available libraries and technologies have become in recent years.

There are a few clear ways my approach could be optimized in the future. First, when using Dipy, some of the registration results weren’t perfectly aligned due to certain dark regions in the brain image. However, I was able to address most of these issues by adjusting some of the parameters, so I think it would be worthwhile to learn more about the SymmetricDiffeomorphiRegistration and CCMetric classes and some of the theory behind them. By understanding these better, the registration results would be even more accurate, which would improve the automatic segmentation. Another improvement would be changing the histogram equalization method I used slightly. Since many pixels in the image have an intensity of 0, and we don’t care about their placement, they shouldn’t be included in the equalization. Also, I could experiment with different histogram shapes to change the contrast of the brain image, which could also improve results. Finally, I could try more sophisticated segmentation methods. There are many techniques that have been proven to work very well in the context of brain MRI images that I could look at. For example, when computing global weights, also taking the correlation between images in the training set could improve results. Another improvement could be including more data in the training set.

Although I have shown that it is possible to produce fairly accurate and automatic segmentations for brain images, and that there are many ways to improve the accuracy of this algorithm, it is very important to consider the ethics of such an algorithm. One issue relating to this project, is that a readily available automatic segmentation algorithm would make it cheaper and faster to get information about someone’s brain. This may not seem like an inherently bad thing , but this would make it easier for hospitals and doctors to show their patients potentially concerning changes or growths in their brain. Since the doctors and healthcare organizations might not always have patient care as their number one priority, and instead might care more about financials, it is possible that more patients will be recommended and given treatment because of this type of technology, even if they might not really need it (or they may not really want it, but are just tricked or scared into accepting a certain treatment plan). The question is then, is it worth making this technology available, which could decrease the cost of MRI image analysis and make them more widely available, if it means some people will be exploited and given treatment they don’t need? I think the best way to address this is to make patients aware that this algorithm doesn’t replace a medical opinion, and instead should be used as a complement. In my opinion, data like this shouldn’t replace doctors, it should just remove the tedious parts from the doctor’s job so they can instead focus on treating patients. If patients are educated and told that this algorithm is just another tool, and results from it don’t always mean medical treatment is necessary, they will still be able to make an informed decision based on the facts. Another potential idea is to add required positions at hospitals that oversee the application of this technology to make sure it isn’t abused. If there is a system in place to report people abusing this system, then I think cases of abuse would decrease a lot.

As this type of technology becomes more available and reliable, it is important that we think about how decision-makers at healthcare organizations might abuse it to convince people to receive (and therefore pay for) treatments they either don’t really need. Having an algorithm like this might make it easier to spot issues, but it also would make it easier for doctors to recommend treatments or surgeries their patients don’t need. However, I think if these ethical issues are kept in mind, they can be addressed while the technology is developed to prevent ethically concerning issues.

Citations

Brett, Matthew, Markiewicz, Christopher J., Hanke, Michael, Côté, Marc-Alexandre, Cipollini, Ben, McCarthy, Paul, … freec84. (2020, April 20). nipy/**nibabel**: 3.1.0 (Version 3.1.0). Zenodo. <http://doi.org/10.5281/zenodo.3757992>

Garyfallidis E, Brett M, Amirbekian B, Rokem A, van der Walt S, Descoteaux M, Nimmo-Smith I and Dipy Contributors (2014). **DIPY**, a library for the analysis of diffusion MRI data. Frontiers in Neuroinformatics, vol.8, no.8.

Hunter, John D. **Matplotlib**: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55

McKinney, Wes. Data Structures for Statistical Computing in Python **(pandas)**, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

Oliphant, Travis E. Python for Scientific Computing, Computing in Science & Engineering **(SciPy)**, 9, 10-20 (2007), DOI:10.1109/MCSE.2007.58

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. **Scikit-learn**: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011)

Sun, Liang, et al. “Reliability-Based Robust Multi-Atlas Label Fusion for Brain MRI Segmentation.” Artificial Intelligence in Medicine, vol. 96, May 2019, pp. 12–24., doi:10.1016/j.artmed.2019.03.004.

Van der Walt, Stéfan, S. Chris Colbert and Gaël Varoquaux. The **NumPy** Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

Wang, Hongzhi, et al. “Optimal Weights for Multi-Atlas Label Fusion.” Lecture Notes in Computer Science Information Processing in Medical Imaging, 30 Nov. 2011, pp. 73–84., doi:10.1007/978-3-642-22092-0\_7.