ECE 4250 Final Submission Best

May 24, 2021

1 ECE 4250 Final Report and Competition:

Implement a more sophisticated label fusion based segmentation strategy to yield better segmentations. Feel free to optimize your approach using the validation subjects and their manual segmentations. You can explorevarious directions, including but not limited to: 1. Using an affine or non-linear transformation model that achieves better alignment than geometric transformations 2. Computing a weighted fusion approach where the training subjects (atlases) are weighed differently based on similarity between intensity values 3. A patch based approach that seeks similar atlas patches in a certain neighborhood 4. Replacing nearest neighbor interpolation with a different method

2 Loading Images

Loads all the images and separates the images into the different training, testing, validation, and segmentation lists

In [67]:
```
import sys
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import nibabel as nib
import scipy
from scipy.ndimage import rotate
from scipy import optimize
from scipy.optimize import minimize
from scipy.stats import mode
from sklearn.utils.extmath import weighted_mode
import skimage
from skimage import transform as tf
from skimage.transform import warp, AffineTransform
from sklearn.metrics import mean_absolute_error
from skimage.restoration import denoise_bilateral, estimate_sigma, denoise_nl_means
import cv2
from PIL import Image

# load the segmentation images from the file directory I made to house all the images
```

```
# Training 1 images
img1 = nib.load("ECE4250ProjectImages/IBSR_01_ana.img");

# Training 1 segmentation
seg1 = nib.load("ECE4250ProjectImages/IBSR_01_seg_ana.img");

# Training 2 images
img2 = nib.load("ECE4250ProjectImages/IBSR_02_ana.img");

# Training 2 segmentation
seg2 = nib.load("ECE4250ProjectImages/IBSR_02_seg_ana.img");

# Training 3 images
img3 = nib.load("ECE4250ProjectImages/IBSR_03_ana.img");

# Training 3 segmentation
seg3 = nib.load("ECE4250ProjectImages/IBSR_03_seg_ana.img");

# Training 4 images
img4 = nib.load("ECE4250ProjectImages/IBSR_04_ana.img");

# Training 4 segmentation
seg4 = nib.load("ECE4250ProjectImages/IBSR_04_seg_ana.img");

# Training 5 images
img5 = nib.load("ECE4250ProjectImages/IBSR_05_ana.img");

# Training 5 segmentation
seg5 = nib.load("ECE4250ProjectImages/IBSR_05_seg_ana.img");

# Training 6 images
img6 = nib.load("ECE4250ProjectImages/IBSR_06_ana.img");

# Training 6 segmentation
seg6 = nib.load("ECE4250ProjectImages/IBSR_06_seg_ana.img");

# Validation 7 images
img7 = nib.load("ECE4250ProjectImages/IBSR_07_ana.img");

# Validation 7 segmentation
seg7 = nib.load("ECE4250ProjectImages/IBSR_07_seg_ana.img");

# Testing 8 images
img8 = nib.load("ECE4250ProjectImages/IBSR_08_ana.img");

# Testing 9 images
img9 = nib.load("ECE4250ProjectImages/IBSR_09_ana.img");
```

```
# Testing 10 images
img10 = nib.load("ECE4250ProjectImages/IBSR_10_ana.img");

# Testing 11 images
img11 = nib.load("ECE4250ProjectImages/IBSR_10_ana.img");

# Testing 12 images
img12 = nib.load("ECE4250ProjectImages/IBSR_12_ana.img");

# Testing 13 images
img13 = nib.load("ECE4250ProjectImages/IBSR_13_ana.img");

# Testing 14 images
img14 = nib.load("ECE4250ProjectImages/IBSR_14_ana.img");

# Validation 15 Images
img15 = nib.load("ECE4250ProjectImages/IBSR_15_ana.img");

# Training 15 segmentation
seg15 = nib.load("ECE4250ProjectImages/IBSR_15_seg_ana.img");

# Testing 16 images
img16 = nib.load("ECE4250ProjectImages/IBSR_16_ana.img");

# Testing 17 images
img17 = nib.load("ECE4250ProjectImages/IBSR_17_ana.img");

# setting each array to read out pixel spacing and slice thickness for the correspond
imageArray = np.array([img1, img2, img3, img4, img5, img6, img7, img8, img9, img10, i
imageSegArray = np.array([seg1, seg2, seg3, seg4, seg5, seg6, seg7, seg15])

# gets the middle coronal slice of each image
def middle_coronal_slice(img):
    # gets the fdata of the image being currently used
    imageData = img.get_fdata()
    # gets the slice data from the image data
    sliceData = imageData[:, :, len(imageData[0][0])//2]

    return sliceData.reshape((256, 256))


trainingImages, trainingSegments = [], []
validationImages, validationSegments = [], []
testingImages = []

for i in range(0, 6):
    trainingImages.append(middle_coronal_slice(imageArray[i]))
```

```
for i in [6, 14]:
    validationImages.append(middle_coronal_slice(imageArray[i]))
for i in [7, 8, 9, 10, 11, 12, 13, 15, 16]:
    testingImages.append(middle_coronal_slice(imageArray[i]))
for i in range(len(imageSegArray)):
    if i < 6:
        trainingSegments.append(middle_coronal_slice(imageSegArray[i]))
    else:
        validationSegments.append(middle_coronal_slice(imageSegArray[i]))
```

3 My approach and optimizations:

I will be using an affine transformation model that achieves better alignment than geometric transformations for my optimized approach

4 Creating the tranform function

Taking my transform function from milestone 2 but edited using functions from the skimage library for better optimizations, performance, and accuracy than my implementation

```
In [68]: def transform(movingImage, scale_x, scale_y, angle, transformColumn, transformRow, gri
             # sets up the movingImage's height and width
             h, w = movingImage.shape[0], movingImage.shape[1]

             # sets up the grid size for the shifts
             gridHeight, gridWidth = gridSize[0], gridSize[1]

             # sets up the shift values as well as the temporary shift value
             shift_y, shift_x, temp = (np.array((gridHeight, gridWidth, 1)) - 1) / 2.

             # computes the first and second shifts using the skimage.transform.SimilarityTran
             shift_first = skimage.transform.SimilarityTransform(translation=[-shift_x, -shift_
             shift_back = skimage.transform.SimilarityTransform(translation=[shift_x, shift_y]

             # computes the affine transformation using the skimage AffineTransform function
             affine = AffineTransform(scale = (scale_x, scale_y), rotation = np.deg2rad(angle)

             # warps the transformed image using skimage.transform.warp
             transformed_img = warp(movingImage, (shift_first + affine + shift_back).inverse, c

             # finally returns the transformed image
             return transformed_img
```

5 Creating the Loss function

This is slightly different than in Milestone 2, now using a function found in skimage.measure to produce the loss and output it

```
In [69]: def loss(parameters, moving_img, fixedImage):

             # uses the tranfsorm function above and passing in the paramters using the *param
             # the *parameters takes the parameters passed into this function and uses them in
             transformed_img = transform(moving_img, *parameters, fixedImage.shape)

             # using the skimage.measure.comparessim function, we can compare the loss of the
             output = skimage.measure.compare_ssim(skimage.color.rgb2gray(transformed_img), ski


             return -output
```

6 Creating the optimized function

Very similar to the optimization function I created in Milestone 2, but now uses the scipy.optimize.minimum function instead of the scipy.optimize.fmin function. By doing this, I am able to get the optimalParameters by performing a non-linear approach to the opimization function, unlike Milestone 2's linear approach

```
In [70]: def optimize(fixedImage, movingImage):
             # this is the part that changed from scipy.optimize.fmin to scipy.optimize.minimu
             optimizedParams = scipy.optimize.minimize(loss, (1,1,0,0,0), args = (fixedImage,mo

             # performs the tranform again on moving and fixed images now using the optimal pa
             optimizedImage = transform(movingImage, *optimizedParams.x, fixedImage.shape)

             # returns the x component of the optimalParameters as well as the newly found opt
             return optimizedParams.x, optimizedImage
```

7 Creating the Geometric Registration function

This function is taken directly from my milestone 2

```
In [71]: def geometricRegistration(fixedImage, movingImage):
             # initialize the height and width of the fixedImage
             height, width = fixedImage.shape[0], fixedImage.shape[1]

             #initialize an array of zeros of size MxN
             normalizedImg = np.zeros((height,width))

             #compute the normalized moving image
             normalized_fixedImage = cv2.normalize(fixedImage,  normalizedImg, 0, 255, cv2.NORM

             #initialize an array of zeros of size MxN
             normalizedImg = np.zeros((height,width))
             #compute the normalized moving image
             normalized_movingImage = cv2.normalize(movingImage,  normalizedImg, 0, 255, cv2.NO
```

```
        #compute optimal parameters, and the geometric transformed image with these optim
        optParams, transfImage = optimize(normalized_fixedImage,normalized_movingImage)

        #return the geometrically transformed image with optimal paramenters
        return optParams, transfImage
```

8 Creating a filtering function for the Moving Image

This function is mean to eliminate noise in the movingImage using functions from the skimage.restoration library. This will help make the images clearer for final output.

```
In [72]: def filter_moving_imgs(movingImage):

        # the estimated_sigma function, as described in
        # https://scikit-image.org/docs/dev/api/skimage.restoration.html#skimage.restorat
        # explains that the estimate_sigma function is "the estimation algorithm that is
        # absolute deviation of the wavelet detail coefficients"
        sigma_est = estimate_sigma(movingImage, multichannel=False, average_sigmas=True)

        # the denoise_nl_means function does the following:
        """ The non-local means algorithm is well suited for denoising images with specif
        The principle of the algorithm is to average the value of a given pixel with valu
        a limited neighbourhood, provided that the patches centered on the other pixels a
        the patch centered on the pixel of interest.
        This information was taken from:
        https://scikit-image.org/docs/dev/api/skimage.restoration.html#skimage.restoratio
        """
        movingImage = denoise_nl_means(movingImage, sigma = 1.0)

        # returns the denoised movingImage
        return movingImage
```

9 Filtering the training and Validation image sets using the filter function above

```
In [73]: for i in range(len(trainingImages)):
            trainingImages[i] = filter_moving_imgs(trainingImages[i])
            print(i) # keep track of progress
        for i in range(len(validationImages)):
            validationImages[i] = filter_moving_imgs(validationImages[i])
            print(i) # keep track of progress
```

```
0
1
2
3
```

```
4
5
0
1
```

In [74]:
```python
training_opt_params, val_opt_params = [], []
count = 0
for fixedImage in testingImages:
    print(count)
    for movingImage in trainingImages:
        op, tf = geometricRegistration(fixedImage, movingImage)
        training_opt_params.append(op)
    for movingImage in validationImages:
        op, tf = geometricRegistration(fixedImage, movingImage)
        val_opt_params.append(op)
    count += 1
```

```
0

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: The behavio

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: DEPRECATED: sl

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: The behavio

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: DEPRECATED: sl

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: The behavio

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: DEPRECATED: sl

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: The behavio

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: DEPRECATED: sl

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: The behavio

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: DEPRECATED: sl

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: The behavio

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: DEPRECATED: sl

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: The behavio

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: DEPRECATED: sl
```

```
C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: The behavio

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: DEPRECATED: sl

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: The behavio

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: DEPRECATED: sl

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: The behavio

C:\Users\klx90\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: DEPRECATED: sl
```

In [75]:
```
# creating the different segmentations using the trainingImages for the Most Frequent
segT0 = [transform(trainingSegments[i], *training_opt_params[i], testingImages[0].shap
segT1 = [transform(trainingSegments[i], *training_opt_params[6+i], testingImages[1].sh
segT2 = [transform(trainingSegments[i], *training_opt_params[12+i], testingImages[2].s
segT3 = [transform(trainingSegments[i], *training_opt_params[18+i], testingImages[3].s
segT4 = [transform(trainingSegments[i], *training_opt_params[24+i], testingImages[4].s
segT5 = [transform(trainingSegments[i], *training_opt_params[30+i], testingImages[5].s
segT6 = [transform(trainingSegments[i], *training_opt_params[36+i], testingImages[6].s
segT7 = [transform(trainingSegments[i], *training_opt_params[42+i], testingImages[7].s
segT8 = [transform(trainingSegments[i], *training_opt_params[48+i], testingImages[8].s

# creating the different segmentations using the validationImages for the Most Freque
segV0 = [transform(validationSegments[i], *val_opt_params[i], testingImages[0].shape)
segV1 = [transform(validationSegments[i], *val_opt_params[i+2], testingImages[0].shape
segV2 = [transform(validationSegments[i], *val_opt_params[i+4], testingImages[0].shape
segV3 = [transform(validationSegments[i], *val_opt_params[i+6], testingImages[0].shape
segV4 = [transform(validationSegments[i], *val_opt_params[i+8], testingImages[0].shape
segV5 = [transform(validationSegments[i], *val_opt_params[i+10], testingImages[0].shap
segV6 = [transform(validationSegments[i], *val_opt_params[i+12], testingImages[0].shap
segV7 = [transform(validationSegments[i], *val_opt_params[i+14], testingImages[0].shap
segV8 = [transform(validationSegments[i], *val_opt_params[i+16], testingImages[0].shap

# initializes the registration segmentation arrays
registrationSegment0 = np.zeros((8, 256, 256))
registrationSegment1 = np.zeros((8, 256, 256))
registrationSegment2 = np.zeros((8, 256, 256))
registrationSegment3 = np.zeros((8, 256, 256))
registrationSegment4 = np.zeros((8, 256, 256))
registrationSegment5 = np.zeros((8, 256, 256))
registrationSegment6 = np.zeros((8, 256, 256))
registrationSegment7 = np.zeros((8, 256, 256))
registrationSegment8 = np.zeros((8, 256, 256))

# appending the segmentation training images into the registration segmentation array
```

```
for i in range (6):
    registrationSegment0[i,:,:] = segT0[i]
    registrationSegment1[i,:,:] = segT1[i]
    registrationSegment2[i,:,:] = segT2[i]
    registrationSegment3[i,:,:] = segT3[i]
    registrationSegment4[i,:,:] = segT4[i]
    registrationSegment5[i,:,:] = segT5[i]
    registrationSegment6[i,:,:] = segT6[i]
    registrationSegment7[i,:,:] = segT7[i]
    registrationSegment8[i,:,:] = segT8[i]

# appending the segmentation validation images into the registration segmentation arr
for i in range (2):
    registrationSegment0[i+6,:,:] = segV0[i]
    registrationSegment1[i+6,:,:] = segV1[i]
    registrationSegment2[i+6,:,:] = segV2[i]
    registrationSegment3[i+6,:,:] = segV3[i]
    registrationSegment4[i+6,:,:] = segV4[i]
    registrationSegment5[i+6,:,:] = segV5[i]
    registrationSegment6[i+6,:,:] = segV6[i]
    registrationSegment7[i+6,:,:] = segV7[i]
    registrationSegment8[i+6,:,:] = segV8[i]
```

10 Most Frequent Label Fusion upgraded

For this, I decided to do my own implementation for the reshape section, since the images I got from milestone 2 were not great with respect to clarity. By making these changes, I should see a better, cleaner, and clearer image. To make it work, I took a normal matrix reshaping operation and decided to try and smooth the output image.

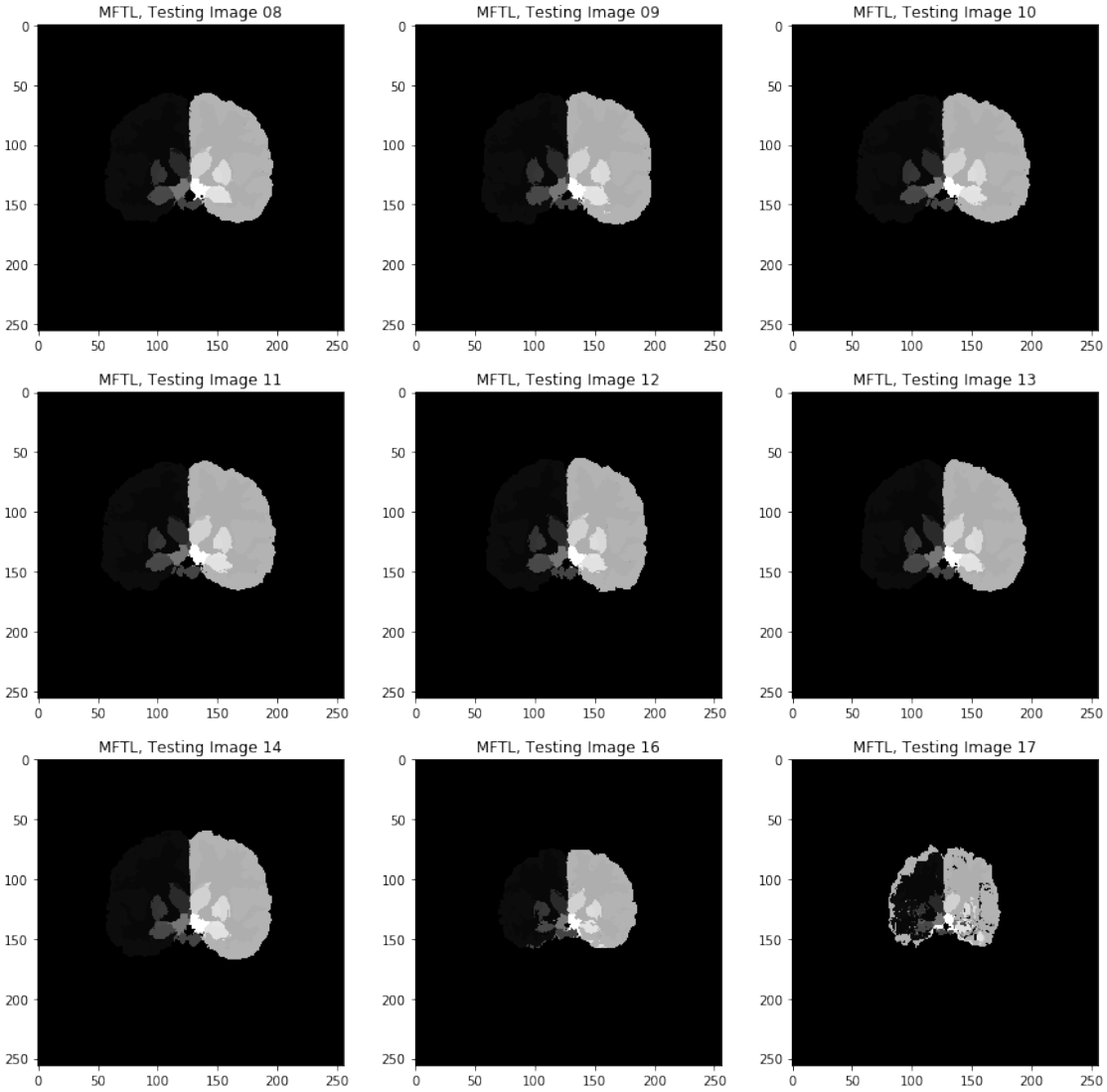
In [76]:
```
# This is the new most frequent label fusion taken from milestone 2
def MFTLUpgraded(imgs):
    modes = np.zeros((256,256))

    for i in range(len(imgs[0])):
        for j in range(len(imgs[0][0])):
            m, count = mode([imgs[x][i][j] for x in range(len(imgs))])
            if count == 1 or m == 0:
                if count == 1:
                    m = np.sum([imgs[x][i][j] for x in range(len(imgs))])//8
                else:
                    if count < 4:
                        m = np.max([imgs[x][i][j] for x in range(len(imgs))])
            modes[i][j] = m
    return modes

def MFTLOriginal(imgs):
```

```
        modes, count = mode(imgs) # Default is axis=0, don't care about count output
        modes = modes.reshape((256, 256))

        return modes
```

```
In [77]: segs = [registrationSegment0, registrationSegment1, registrationSegment2, registratio
                 registrationSegment4, registrationSegment5, registrationSegment6, registratio
         MFTLs = []
         for seg in segs:
             MFTLs.append(MFTLUpgraded(seg))
```

11 Plotting the Images

```
In [82]: figcount = 1
         label = ['08','09','10','11','12','13','14','16','17']
         w = 10
         h = 10
         fig = plt.figure(figsize = (15, 15))
         columns = 3
         rows = 3
         for i in range(1, columns*rows +1):
             fig.add_subplot(rows, columns, i)
             plt.title('MFTL, Testing Image ' + label[i-1])
             rotatedImage = MFTLs[i-1].squeeze()
             rotatedImage = rotate(rotatedImage, -90)
             plt.imshow(rotatedImage, cmap = 'gray')

         plt.show()
         fig.savefig('MFTL.png')
```


MFTL, Testing Image 08
MFTL, Testing Image 09
MFTL, Testing Image 10
MFTL, Testing Image 11
MFTL, Testing Image 12
MFTL, Testing Image 13
MFTL, Testing Image 14
MFTL, Testing Image 16
MFTL, Testing Image 17
0
50
100
150
200
250