Zachary Obrien
zo0021
May 4th 2022
BMEN 5322 Project

Note: This is a brief summary of code/images, the full code is found in the jupyter notebook "dicom_parsing.ipynb". There is significantly more code than what's included here to make things functional and I will refer to code that may not be in the summary images.

Part 1: Pulling the data in and getting metadata

For part 1 I've pulled in the set of dicom images and selected the first to get the metadata about. This had an x,y voxel size of 1mm/px and a slice size of 3mm. This is used later in the "Bonus" section to create a 3d representation, but for now I've just pulled the TR/TE and saw that there's an "Inversion Time" indicating that a spin echo was done for this.

```
157 #*******************************
    #************ PART 1 ************
    #*******************************
    voxel_size = [test_img.PixelSpacing, test_img.SliceThickness]
    tr = test_img.RepetitionTime
    te = test_img.EchoTime
    it = test_img.InversionTime
    wc = test_img.WindowCenter
    ww = test_img.WindowWidth
    print("PART 1: DICOM DATA INFORMATION")
    print("Voxel size:", voxel_size)
    print("Repitition Time:", tr)
    print("Echo Time:", te)
    print("Inversion Time:", it)
    #print("Window Center:", wc)
    print("Window Width/FOV:", ww)
```

```
PART 1: DICOM DATA INFORMATION
Voxel size: [[0.938, 0.938], '3.0']
Repitition Time: 10004.000000
Echo Time: 155.000000
Inversion Time: 2200.000000
Window Width/FOV: 455.000000
```

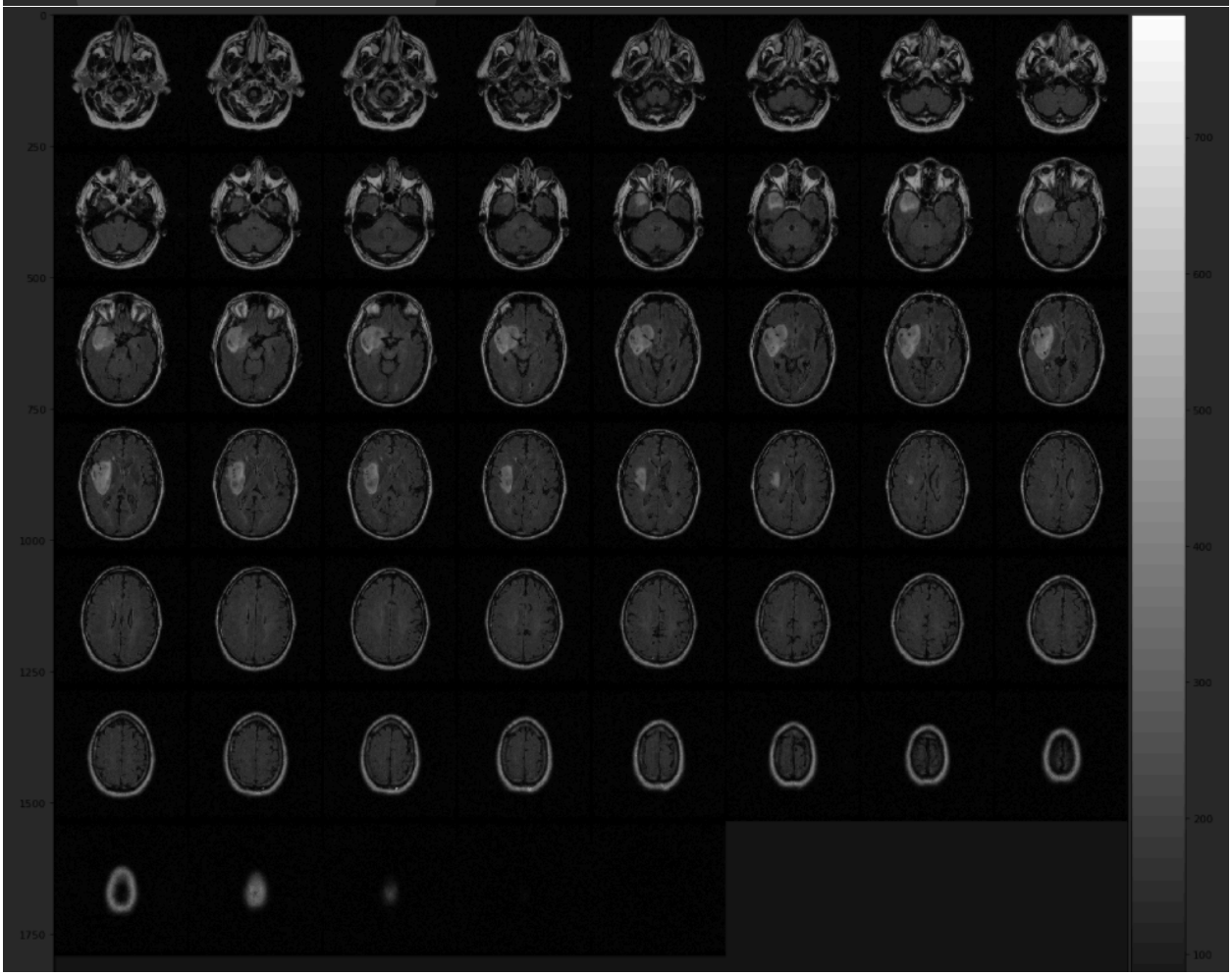
Part 2: Image Montage

For part 2 I took the array of dicom images, looped through and created an array of just the image.PixelData fields and put those into the montage. This array will also be what I use for processing later. The images were numbered correctly but the array didn't populate them in filename order, so sorting based on filename and then creating the array fixed that.

Skimage.util.montage takes an image array and returns a singular image with the array elements tiled into a square. The plt.cm.gray is what ensures displaying in grayscale and also puts the brightness bar on the right hand side.

```
159 #*******************************
    #************ PART 2 ************
    #*******************************
    print("PART 2: IMAGE MONTAGE")
    plt.figure(figsize=(16, 16), dpi=80)
    m = skimage.util.montage(img_arr)
    skimage.io.imshow(m, cmap=plt.cm.gray)
    plt.show()
```

```
PART 2: IMAGE MONTAGE
/Users/zacharyobrien/miniforge3/envs/5215-Project-1/lib/python3.
  lo, hi, cmap = _get_display_range(image)
```



Part 3: Removing the skull and scalp

I have an example of how I did this in raw code here, but did things slightly differently in finding the actual tumors. I'm doing a thresholding to get the area of the skull that had the highest values. I'm changing the pixel values to 0-255 from 0-450 so that it can be plotted in normal RGB space.
There was some fragments of the skull left on the outside so I added an erode/dilate to clean the pieces up.
Note: Using a KMEANS algorithm with the skull there and then finding the larges contiguous mass (via threshold function) gave me better trauma detection so I used that for part 4.

```
165 #*******************************
    #************ PART 3 ************
    #*******************************
    print("PART 3: EXAMPLE SKULL REMOVAL")
    tmp_img = img_arr.copy()[image_num]
    tmp_img = bounding_image(tmp_img, min=0, max=255)
    tmp_img = apply_brainmask(tmp_img)
    # tmp_img = get_inside(tmp_img)
    tmp_img = standard_255(tmp_img)
    blurred = cv2.GaussianBlur(tmp_img,(9,9),cv2.BORDER_DEFAULT)
    (thresh, im_bw) = cv2.threshold(blurred, 125, 255, cv2.THRESH_BINARY|cv2.THRESH_OTSU)
    eroded = morphology.erosion(im_bw, np.ones([5, 5]))
    dilation = morphology.dilation(eroded, np.ones([5, 5]))
    brain_mask = 1 - get_inside(dilation)
    brain_image = (brain_mask * tmp_img)
    plt.figure(figsize=(16, 16), dpi=80)
    #plt.imshow(tmp_img, cmap=plt.cm.gray)
    plt.imshow(brain_image, cmap=plt.cm.gray)
    plt.show()
```

Part 4: Trauma Mapping

I made functions for each step to make things easier, but what utilizes these functions is below.

Looping through the dicom files, I pull out the images into an array. I'm about to make a mask out of them so I make a copy to use, rather than altering the originals (since I'll need these later).

The "get_interest_areas" function is what actually creates the masks. It takes in the images and an array of what ones we want to run the tumor detecting algorithm on. It also takes a parameter "dilate" in case the coloring is being too aggressive in capturing details of the trauma.

The "get_tumor_mask" function takes the input image, runs thresholding on to get rid of noise, then does a KMeans algorithm that ends up finding the high intensity areas best (which correspond with the skull and trauma/tumor). I then get the cluster centers and take only the largest area in the image, which since I've done *some* thresholding, the skull is eroded enough to not be a contiguous piece and is thus not the largest cluster. Once this is done, I run an erosion and dilation on the cluster results because it can be very jagged and might lose 2-5px on the edge of some areas and this cleans that up. I then put it in a 0-255 range so normal image-processing libraries can be used. I do a "findContours" to get the details in the edges of the tumor/trauma back that I might have lost in the erosion/dilation step. Then I go through the perimeter pixels and create an either 0 or 1 mask that is the exact shape of the tumor/trauma in the original image.

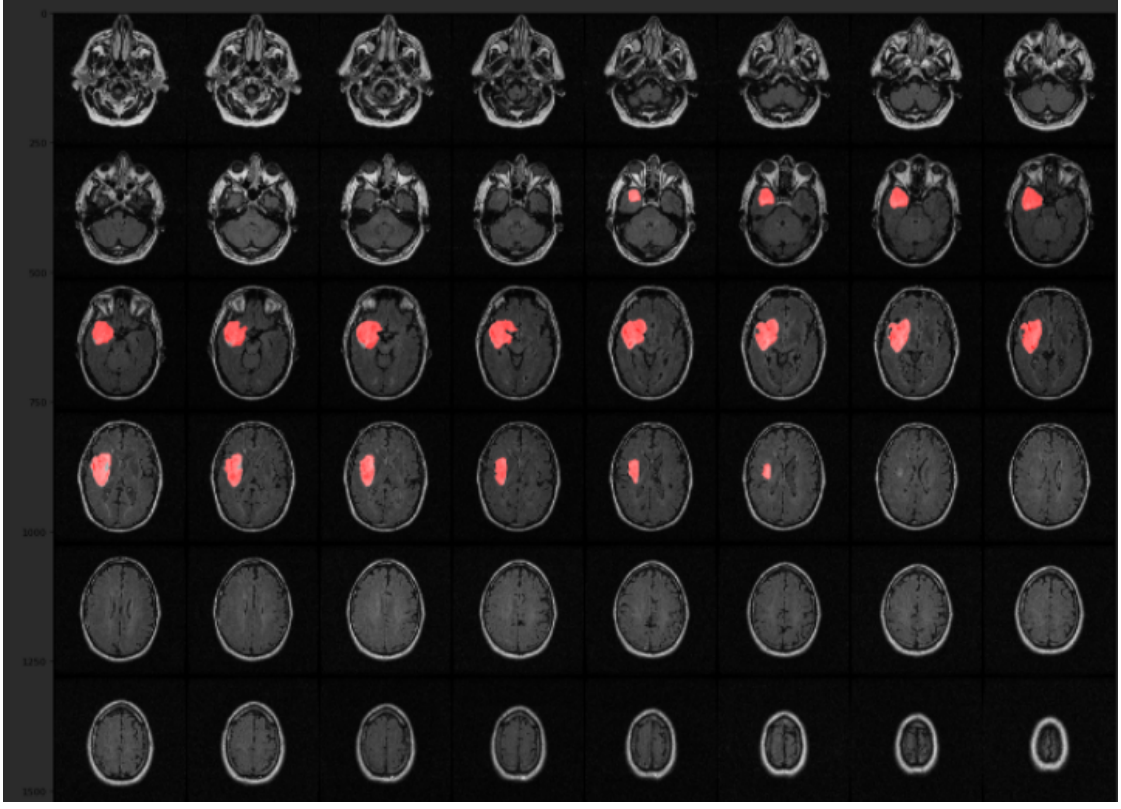
Finally, I take this mask, convert it to a color, and overlay that color using a "bitwise_or" with the original image that I've kept separately from this copy. (The bitwise or allows me to easily keep the details of the image, but turn the area whatever color I would like)

```
171 #*******************************
    #************ PART 4 ************
    #*******************************
    final_masks = get_interest_areas(narrow_images, [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                                                     23, 24, 25, 26, 27, 28, 29], dilate=False)
    img_arr = []
    meta_arr = []
    for file in sorted(onlyfiles):
        tmp_file = dicom.read_file(join(data_path, file))
        img_arr.append(tmp_file.pixel_array)
        meta_arr.append(tmp_file.file_meta)

    final_images = img_arr.copy()
    for index, image in enumerate(final_images):
        image = np.asarray(image)
        image = standard_255(image)
        image = image.reshape((256, 256, 1))
        color_img = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)

        color_mask = final_masks[index]
        color_mask = color_mask.reshape((256, 256, 1)).astype("uint8")
        color_mask = cv2.cvtColor(color_mask, cv2.COLOR_GRAY2RGB)

        turn_green = np.full((256,256,3), [255, 0, 0], dtype="uint8")
        color_mask = color_mask * turn_green

        # print(color_img[100,100])
        final_images[index] = np.bitwise_or(color_img, color_mask)

    plt.figure(figsize=(16, 16), dpi=80)
    m = skimage.util.montage(final_images, multichannel=True)
    skimage.io.imshow(m, cmap=plt.cm.gray)
    plt.show()

    /var/folders/tm/mxj8z53n0pbbt53crmv3xpl00000gn/T/ipykernel_51343/3170165775.py:31: FutureWar
      m = skimage.util.montage(final_images, multichannel=True)
```

```
def get_interest_areas(image_set, mask_set, dilate=7):
    image_masks = []
    for index, single_image in enumerate(image_set):
        # print("On image", index)
        # print("Energy is", np.sum(single_image))
        try:
            if (np.sum(single_image) > 100000) and (index in mask_set):
                tmp_mask = get_tumor_mask(single_image)
                if dilate:
                    image_masks.append(morphology.dilation(tmp_mask, np.ones([dilate, dilate])))
                else:
                    image_masks.append(tmp_mask)
            else:
                image_masks.append(np.zeros(single_image.shape))
        except:
            image_masks.append(np.zeros(single_image.shape))
    return image_masks
```

```
230 def get_tumor_mask(image):
        image = bounding_image(image)
        image = apply_brainmask(image)
        image = standard_255(image)

        blurred = cv2.GaussianBlur(image,(7,7),cv2.BORDER_DEFAULT)
        (thresh, im_bw) = cv2.threshold(blurred, 125, 255, cv2.THRESH_BINARY|cv2.THRESH_OTSU)
        (cnts, _) = cv2.findContours(im_bw.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
        #(cnts, _) = cv2.findContours(im_bw.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

        mask = np.zeros(image.shape)
        if len(cnts) > 0:
            c = max(cnts, key = cv2.contourArea)
            #cv2.drawContours(mask, cnts, -1, (255,255,255), 3)
            cv2.fillPoly(mask, pts =[c], color=(1))

        return mask
```

```
PART 4: Colored Trauma Montage
/var/folders/tm/mxj8z53n0pbbt53crmv3xpl00000gn/T/ipykernel_51343/2846897075.py:32: FutureWa
  m = skimage.util.montage(final_images, multichannel=True)
```

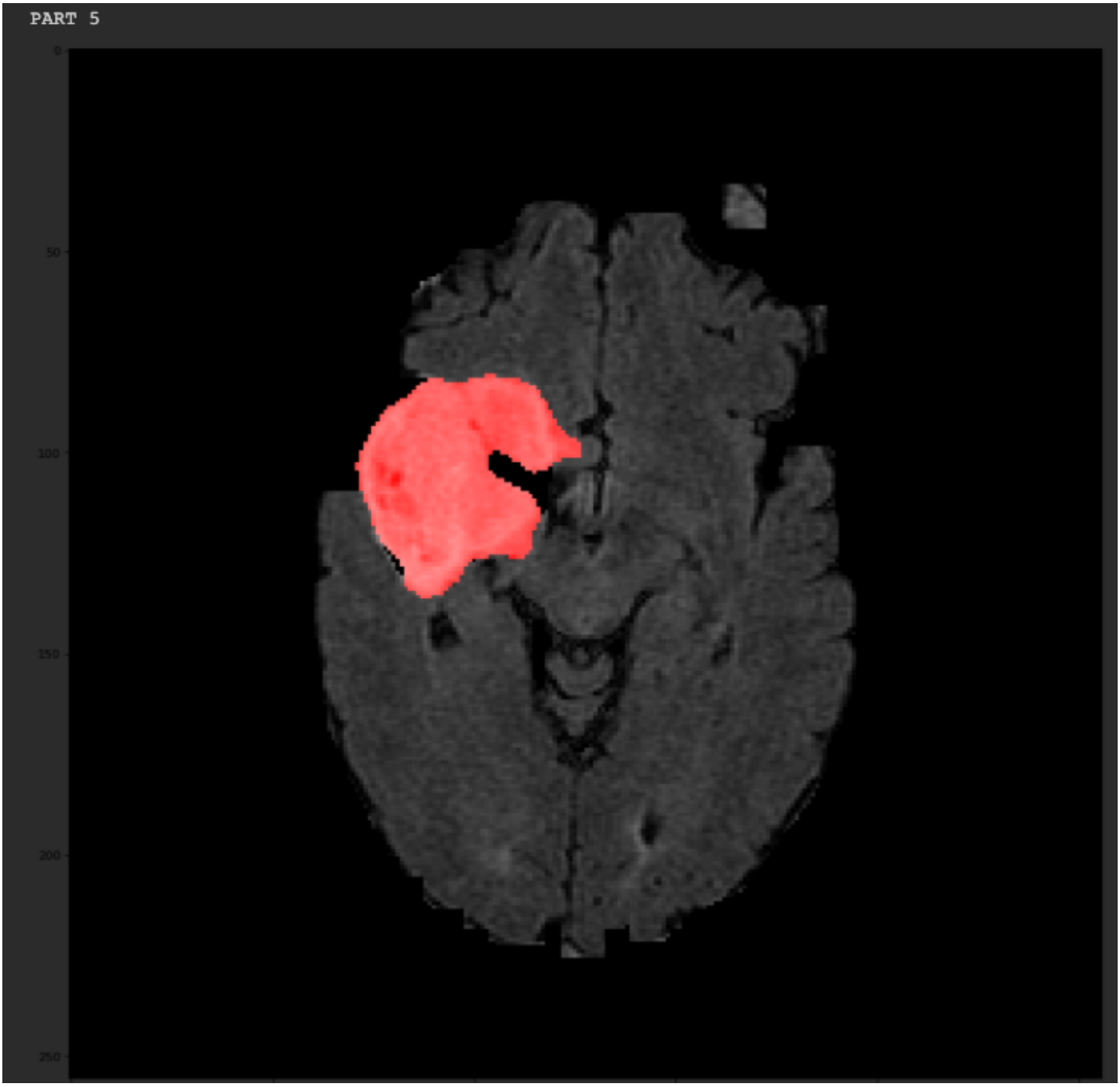


Part 5: Tumor/Trauma contour w/o skull

For this, I just combined the way I found the tumors and then performed the same skull removal as in step 2, but replaced the tumor back into the image since it *can* get partially removed if I threshold too aggressively. I'm sure the thresholding could be done more delicately or other methodologies could be used, but in my case, since I already had a mask with the located trauma, it was easier to allow this loss in the trauma (since the trauma shared threshold values with the skull).

The image copy through the "final_mask_no_skull" creation is the exact same as the tumor/trauma detection, but also finding the rest of the grey/white matter of the brain. This is then gone over and everything that is 0 or null is *NOT* brain tissue in the MASK and can be removed from the IMAGE by setting image[row,col] = 0 (corresponding to the mask[row,col]). This gives us a final image with the skull removed. Finally, I do the same conversion to RGB, reshaping, coloring things red, and bitwise_or to get the final image.

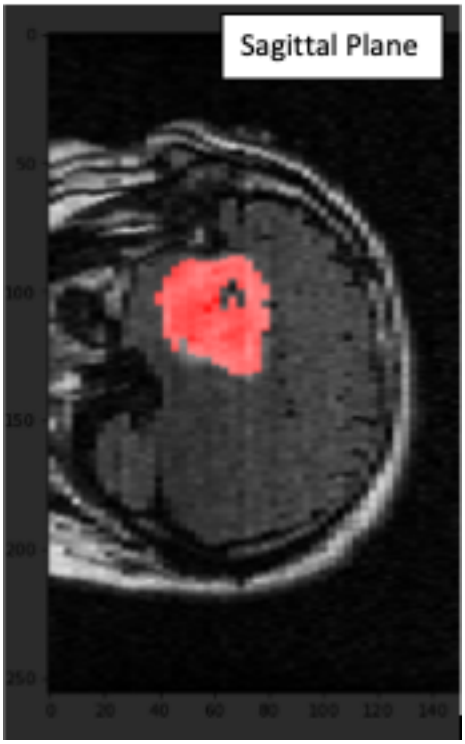
```
#*******************************
#************ PART 5 ************
#*******************************
print("PART 5")
tmp_img2 = img_arr[19].copy()
tmp_img2 = bounding_image(tmp_img2, min=0, max=250)
tmp_img2 = apply_brainmask(tmp_img2)
tmp_img2 = standard_255(tmp_img2)
blurred = cv2.GaussianBlur(tmp_img2,(9,9),cv2.BORDER_DEFAULT)
(thresh, im_bw) = cv2.threshold(blurred, 125, 255, cv2.THRESH_BINARY|cv2.THRESH_OTSU)
eroded = morphology.erosion(im_bw, np.ones([9, 9]))
dilation = morphology.dilation(eroded, np.ones([5, 5]))
brain_mask = 1 - get_inside(dilation)
brain_image = (brain_mask * tmp_img)
brain_dilation = morphology.dilation(brain_image, np.ones([9, 9]))
brain_and_injury = brain_dilation + (final_masks[19] * 255)
final_mask_no_skull = np.zeros(brain_and_injury.shape)

image = img_arr[19].copy()
image = np.asarray(image)
image = standard_255(image)
image = image.reshape((256, 256, 1))

for row in range(brain_and_injury.shape[1]):
    for col in range(brain_and_injury.shape[0]):
        if brain_and_injury[row,col] <= 0:
            image[row,col] = 0

color_img = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
mask = final_masks[19].copy()
color_mask = mask.reshape((256, 256, 1)).astype("uint8")
color_mask = cv2.cvtColor(color_mask, cv2.COLOR_GRAY2RGB)

turn_green = np.full((256,256,3), [255, 0, 0], dtype="uint8")
color_mask = color_mask * turn_green

final_image_no_skull = np.bitwise_or(color_img, color_mask)

plt.figure(figsize=(16, 16), dpi=80)
plt.imshow(final_image_no_skull, cmap=plt.cm.gray)
plt.show()
```

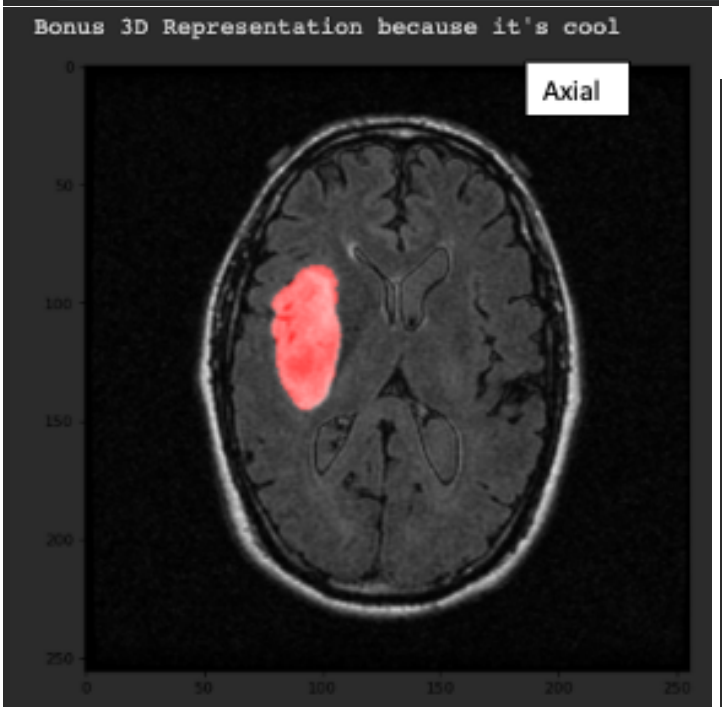

PART 5
0
50
100
150
200
250

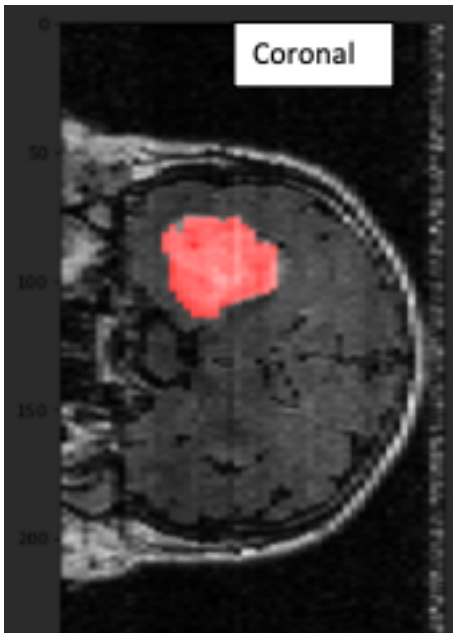
Not asked for, but cool: Viewing in 3D

I noticed that for the slice thickness it was just 3x the x,y voxel size. I knew from previous projects if I had this in a 3D array then I could look at the data from axial, sagittal, or coronal viewpoint by selecting a new "slice" in whichever direction is most interesting to me. I know there *are ways* of plotting this in 3D with rotation/zoom/coloring/etc, but for just a few lines of code it was create to be able to take an image of the tumor/trauma from all 3 angles. I did notice that because the slice thickness was 3mm vs 1mm for the x,y. There's a noticeable "blocky-ness" and loss of resolution looking at things on the sagittal or coronal planes, but it was still interesting.

```
246 #*******************************
    #************ BONUS  ************
    #*******************************

    print("Bonus 3D Representation because it's cool")
    img_shape = list((256, 256))
    img_shape.append(len(final_images) * 3)
    img_shape.append(3)
    img3d = np.full(img_shape, [0, 0, 0])
    # fill 3D array with the images from the files
    stacked_images = []
    vert_ratio = int(thick / spacing[0])
    for image in final_images:
        for rep in range(0,vert_ratio):
            stacked_images.append(image)
    for i, s in enumerate(stacked_images):
        img2d = s
        img3d[:, :, i] = img2d

    # plot 3 orthogonal slices
    plt.figure(figsize=(16, 16), dpi=80)
    a1 = plt.subplot(2, 2, 1)
    plt.imshow(img3d[:, :, 80,:])
    a1.set_aspect(ax_aspect)

    a2 = plt.subplot(2, 2, 2)
    plt.imshow(img3d[:, 80, :,:])
    a2.set_aspect(sag_aspect)

    a3 = plt.subplot(2, 2, 3)
    plt.imshow(img3d[120, :, :,:])
    a3.set_aspect(cor_aspect)

    plt.show()
```







Full, unlabeled 3D