DICOMRTTool_Tutorial

March 21, 2023

1 DICOM RT Tool Tutorial with Open-Access Data

This notebook demonstrates the various functions and utilities available in the Dicom RT tool Python package (https://github.com/brianmanderson/Dicom_RT_and_Images_to_Mask) by Anderson et. al. It serves as supplementary information for the Technical Paper titled: "Simple Python Module for Conversions between DICOM Images and Radiation Therapy Structures, Masks, and Prediction Arrays" . This notebook works through an example of publicly available brain tumor data of T1-w/FLAIR MRI sequences and corresponding RT structure files with multiple segmented regions of interest. Full information of the publicly available brain tumor data used in this notebook can be found at: https://figshare.com/articles/dataset/Data_from_An_Investigation_of_Machine_Learning_Methods_in_Delt radiomics_Feature_Analysis/9943334. This notebook was written for easy accessibility for beginners to Python programming, medical imaging, and computational analysis. It should take no more than 10-15 minutes to run in it's entirety from scratch. The notebook generates about 10 GB worth of files, so ensure you have adequate space to run it.

The notebook covers the following topics (click to go to section): 1. Getting the data 2. Reading in DICOM and RT struct files and converting to numpy array format 3. Saving arrays to nifti format and reloading them 4. Saving and loading numpy array files 5. Calculating radiomic features 6. Predictions To RT-Structure Example

The notebook assumes you have the following nested directory structure after running cells that download necessary data:

[ ]:
```
"""
Top-level directory/
  DICOMRTTool_manuscript.ipynb
  Example_Data/ <- Generated when you run the cells below
|     Image_Data/
|         Structure/ <- These correspond to the Pre-RT scans
              T1/
|                 Patient number/
|                     RT Struc file (.dcm)
              T2FLAIR/
|                 Patient number/
|                     RT Struc file (.dcm)
|         T1/
|             Post1/
|                 Patient number/
|                      DICOM image files (.dcm)
|              Post2/
|                  Patient number/
|                      DICOM image files (.dcm)
|              Pre/
|                  Patient number/
|                      DICOM image files (.dcm) <- The images we care about
|          T2FLAIR/
|              Post1/
|                  Patient number/
|                      DICOM image files (.dcm)
|              Post2/
|                  Patient number/
|                      DICOM image files (.dcm)
|              Pre/
|                  Patient number/
|                      DICOM image files (.dcm) <- The images we care about
  Data.zip <- Generated when you run the cells below, downloaded Figshare file
  Nifti_Data/ <- Generated when you run the cells below
|     Image.nii
|     Mask.nii
|     MRN_Path_To_Iteration.xlsx
|     Overall_Data_Examples_(iteration)0.nii.gz
|     Overall_mask_Examples_y(iteration)0.nii.gz
  Numpy_Data/ <- Generated when you run the cells below
|     image.npy
|     mask.npy
  RT_Structures/ <- Generated when you run the cells below
|     RS_Test_UID.dcm
"""
```

[7]:
```
%%capture
# Load or install the program, %%capture supresses print statements
!pip install DicomRTTool --upgrade
from DicomRTTool.ReaderWriter import DicomReaderWriter, ROIAssociationClass
```

[8]:
```
# importing neccessary libraries

# file mangagment
import os
import zipfile
from six.moves import urllib

# array manipulation and plotting
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# medical image manipulation
import SimpleITK as sitk
```

1.1 Part 1: Getting the data.

The RT struc files and their corresponding DICOM images can be in the same directory or different directories. Here we show a case where structure files and images are located in different directories. This is a good dataset to work with since its somewhat messy but coherent enough to show power of DICOMRTTool. Many files (pre-RT, post-RT at 2 timepoints) but only pre-RT T1 and FLAIR images have associated RT structure files. Downloading and unzipping the necessary files will take about 10 minutes on most CPUs and takes up about 8 GB of storage. One may visualize these DICOM images using a free commercially available DICOM viewer, such as Radiant (https://www.radiantviewer.com/).

```
[ ]: %%time
data_path = os.path.join('.', 'Example_Data')
if not os.path.isdir(data_path): # create Example_data directory if it doesn't␣
 ↪exist
    os.mkdir(data_path)

url_img = "https://ndownloader.figshare.com/files/20140100" # brain scans
filename_img = os.path.join(data_path, 'Data.zip')
if not os.path.exists(filename_img): # if zip file doesnt exist download
    print ("Retrieving zipped images...")
    print('Estimated download time is 5 minutes...')
    urllib.request.urlretrieve(url_img, filename_img)
    print('Finished downloading!')
else:
    print ("Zipped images already downloaded.")

if os.path.exists(filename_img):  # If we downloaded the data
    if not os.path.exists(os.path.join(data_path, 'Image_Data')): # and it␣
 ↪hasn't been unzipped
        print ("Unzipping images...")
        print('Estimated unzip time is 2 minutes')
        z = zipfile.ZipFile(filename_img)
        z.extractall(data_path)
        print ("Done unzipping images.")

print("All required files downloaded and unzipped!") # print when done
```

```
[9]: def display_slices(image, mask, skip=1):
    """
    Displays a series of slices in z-direction that contains the segmented␣
 ↪regions of interest.
    Ensures all contours are displayed in consistent and different colors.
```

```
        Parameters:
            image (array-like): Numpy array of image.
            mask (array-like): Numpy array of mask.
            skip (int): Only print every nth slice, i.e. if 3 only print every␣
↪3rd slice, default 1.
        Returns:
            None (series of in-line plots).
    """

    slice_locations = np.unique(np.where(mask != 0)[0]) # get indexes for where␣
↪there is a contour present
    slice_start = slice_locations[0] # first slice of contour
    slice_end = slice_locations[len(slice_locations)-1] # last slice of contour

    counter = 1

    for img_arr, contour_arr in zip(image[slice_start:slice_end+1],␣
↪mask[slice_start:slice_end+1]): # plot the slices with contours overlayed␣
↪ontop
        if counter % skip == 0: # if current slice is divisible by desired skip␣
↪amount
            masked_contour_arr = np.ma.masked_where(contour_arr == 0,␣
↪contour_arr)
            plt.imshow(img_arr, cmap='gray', interpolation='none')
            plt.imshow(masked_contour_arr, cmap='cool', interpolation='none',␣
↪alpha=0.5, vmin = 1, vmax = np.amax(mask)) # vmax is set as total number of␣
↪contours so same colors can be displayed for each slice
            plt.show()
        counter += 1
```

1.2 Part 2: Reading in DICOM and RT struct files and converting to numpy array format.

The principal on which this set of tools operates on is based on the DicomReaderWriter object. It is instantiated with the contours of interest (and associations) and can then be used to create numpy arrays of images and masks of the format [slices, width, height].

The following code logic is used to demonstrate searching a path and returning indices for matched structures and images (by UID) for arbitrary directory structures (DICOM image files and RT Struct files not in the same folder). If all necessary structure files are in the same folder as the corresponding images (by UID), one can alternatively use an os.walk through directories of interest and call DicomReaderWriter each time a folder is discovered. For example, I normally use a folder structure MRN -> date of image (pre,mid,post-RT) -> type of scan (MRI, CT, etc.) -> files (DICOM images + RT Struct). However, this approach calls the DicomReaderWriter iteratively, which can be computationally taxing.

```
[10]: DICOM_path = os.path.join('.', 'Example_Data', 'Image_Data') # folder where␣
        ↪downloaded data was stored
      print(DICOM_path)
```

```
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data
```

This will walk through all of the folders, and using SimpleITK, will separate them based on SeriesInstanceUIDs.

```
[ ]: %%time
     Dicom_reader = DicomReaderWriter(description='Examples', arg_max=True)
     print('Estimated 30 seconds, depending on number of cores present in your␣
       ↪computer')
     Dicom_reader.walk_through_folders(DICOM_path) # need to define in order to use␣
       ↪all_roi method
```

```
[12]: all_rois = Dicom_reader.return_rois(print_rois=True)  # Return a list of all␣
        ↪rois present, and print them
```

```
The following ROIs were found
rttempglioma
exprttempglioma
brainstem
dose 500[cgy]
dose 1000[cgy]
dose 1200[cgy]
gtvplus2
expltparrecgliom
ltparrecglioma
expltfrontrecao
ltfrontrecao
body
expltfrparrecgbm
ltfrparrecgbm
explttempglioma
lttempglioma
exprtfrontrecgbm
rtfrontrecgbm
expinfrttemprecg
infrttempgbm
dose 2400[cgy]
expltfrontgbm
ltfrontgbm
exprttemprecglio
rttemprecglioma
rtfrontrecglioma
exprtfrontrecgli
brainstem1
eye, left
```

```
eye, right
chiasm
lens, left
lens, right
optic nerve, rig
optic nerve, lef
dose 2500[cgy]
exprttemprecgbm
rttemprecgbm
exprtfrparresxn
right_front_par_
abv
abv_roi
```

As we can see, these ROIs correspond to a variety of structures. In particular, we can see many GBM and glioma structures. Note GBM denotes glioblastoma multiforme (a high grade glioma).

```
[13]: # Print the locations of all RTs with a certain ROI name, automatically lower␣
        ↪cased
      Dicom_reader.where_is_ROI(ROIName='BrAiNsTeM1')
```

```
Contours of brainstem1 are located:
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data\Structure\
T1\001\RS.CA1756_T13D.dcm
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data\Structure\
T1\011\RS.GF6065_T13D.dcm
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data\Structure\
T2Flair\001\RS.CA1756_T2Flair.dcm
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data\Structure\
T2Flair\011\RS.GF6065_T2Flairdcm.dcm
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data\T1\001\RS.
CA1756_T13D.dcm
```

```
[13]: ['C:\\Users\\markb\\Modular_Projects\\Example_Data\\All_MR_Images\\Image_Data\\S
      tructure\\T1\\001\\RS.CA1756_T13D.dcm',
       'C:\\Users\\markb\\Modular_Projects\\Example_Data\\All_MR_Images\\Image_Data\\S
      tructure\\T1\\011\\RS.GF6065_T13D.dcm',
       'C:\\Users\\markb\\Modular_Projects\\Example_Data\\All_MR_Images\\Image_Data\\S
      tructure\\T2Flair\\001\\RS.CA1756_T2Flair.dcm',
       'C:\\Users\\markb\\Modular_Projects\\Example_Data\\All_MR_Images\\Image_Data\\S
      tructure\\T2Flair\\011\\RS.GF6065_T2Flairdcm.dcm',
       'C:\\Users\\markb\\Modular_Projects\\Example_Data\\All_MR_Images\\Image_Data\\T
      1\\001\\RS.CA1756_T13D.dcm']
```

```
[14]: Dicom_reader.which_indexes_have_all_rois()  # Check to see which indexes have␣
        ↪all of the rois we want
      # Since we haven't defined anything yet, it prompts you to input a list of␣
        ↪contour names
```

```
You need to first define what ROIs you want, please use
.set_contour_names_and_associations()
```

```
[ ]: Dicom_reader.which_indexes_lack_all_rois() # Check to see which indexes LACK␣
     ↪all of the rois we want
     # Since we haven't defined any wanted ROI yet, it will prompt you to input a␣
     ↪list of contour names
```

From these ROIs, we will look for those that describe the following regions of interest: tumor (glioblastoma multiforme only) and high-dose area of radiation therapy.

```
[16]: Contour_Names = ['tumor', 'high_dose']
      associations = [ROIAssociationClass('high_dose',['dose 1000[cgy]', 'dose␣
      ↪1200[cgy]']),
                     ROIAssociationClass('tumor', ['exprtfrontrecgbm',␣
      ↪'rtfrontrecgbm', 'expltfrontgbm', 'ltfrontgbm',
                                                   'infrttempgbm', 'rttemprecgbm',␣
      ↪'exprttemprecgbm', 'expltfrparrecgbm',
                                                   'ltfrparrecgbm'])]
```

```
[ ]: !winget install pandoc
```

```
[ ]: Dicom_reader.set_contour_names_and_associations(contour_names=Contour_Names,␣
     ↪associations=associations)
```

Note: The module is printing "Found []" because many of the scans (post-1 and post-2 RT) do not have associated structure files. The module recognizes these images exist (unique UIDs) but associated structure files cannot be located for them.

```
[18]: indexes = Dicom_reader.which_indexes_have_all_rois()  # Check to see which␣
      ↪indexes have all of the rois we want, now we can see indexes
```

```
The following indexes have all ROIs present
Index 7, located at C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\I
mage_Data\T2Flair\Pre\009
Index 11, located at C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\
Image_Data\T2Flair\Pre\003
Index 18, located at C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\
Image_Data\T2Flair\Pre\010
Index 28, located at C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\
Image_Data\T2Flair\Pre\005
Index 31, located at
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data\T1\Pre\003
Index 35, located at
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data\T1\Pre\005
Index 54, located at C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\
Image_Data\T2Flair\Pre\011
Index 58, located at
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data\T1\Pre\010
```

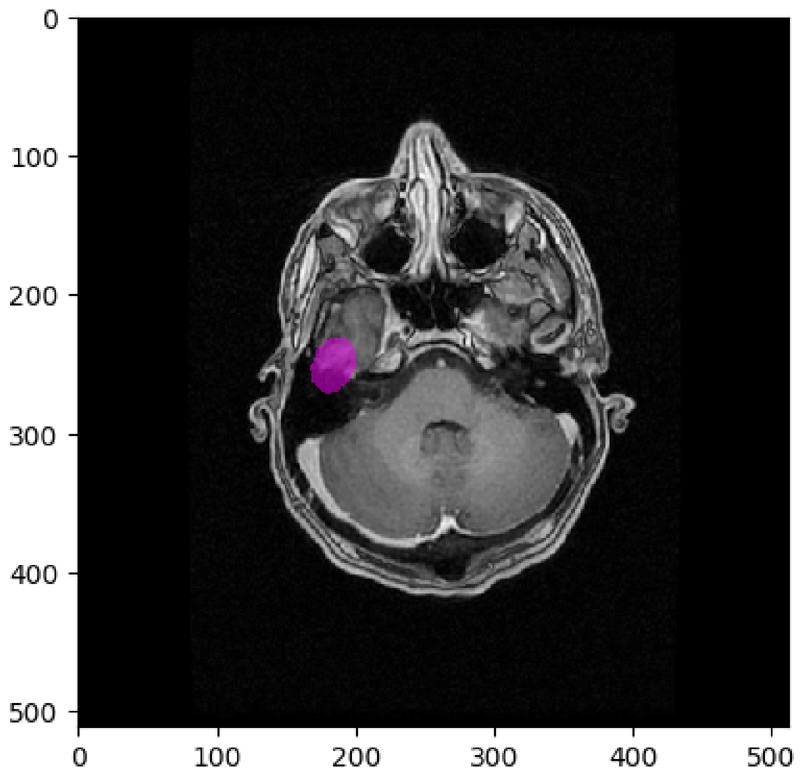
```
Index 60, located at
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data\T1\Pre\011
Finished listing present indexes
```

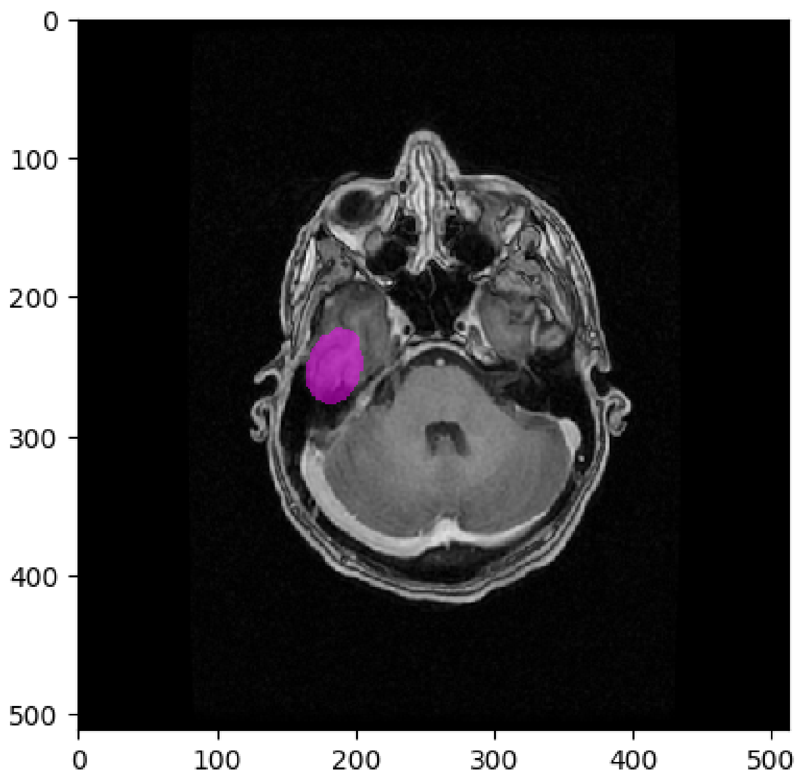
[19]:
```
pt_indx = indexes[-1]
Dicom_reader.set_index(pt_indx)  # This index has all the structures,␣
  ↪corresponds to pre-RT T1-w image for patient 011
Dicom_reader.get_images_and_mask()  # Load up the images and mask for the␣
  ↪requested index
```

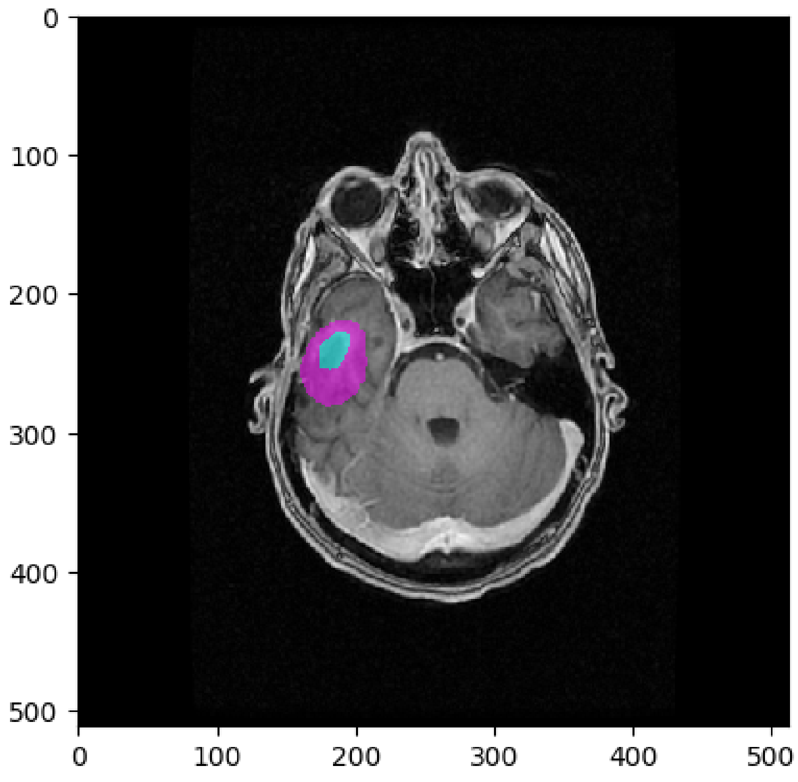
```
Loading images for ax T1 3D 1MM  +c at
C:\Users\markb\Modular_Projects\Example_Data\All_MR_Images\Image_Data\T1\Pre\011
```

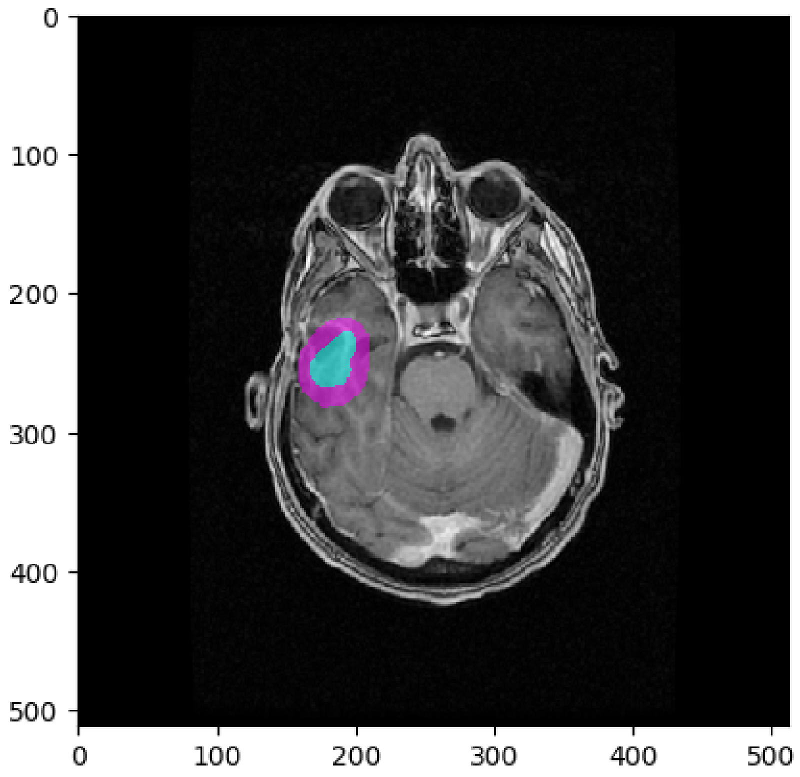
[20]:
```
image = Dicom_reader.ArrayDicom # image array
mask = Dicom_reader.mask # mask array
dicom_sitk_handle = Dicom_reader.dicom_handle # SimpleITK image handle
mask_sitk_handle = Dicom_reader.annotation_handle # SimpleITK mask handle
```

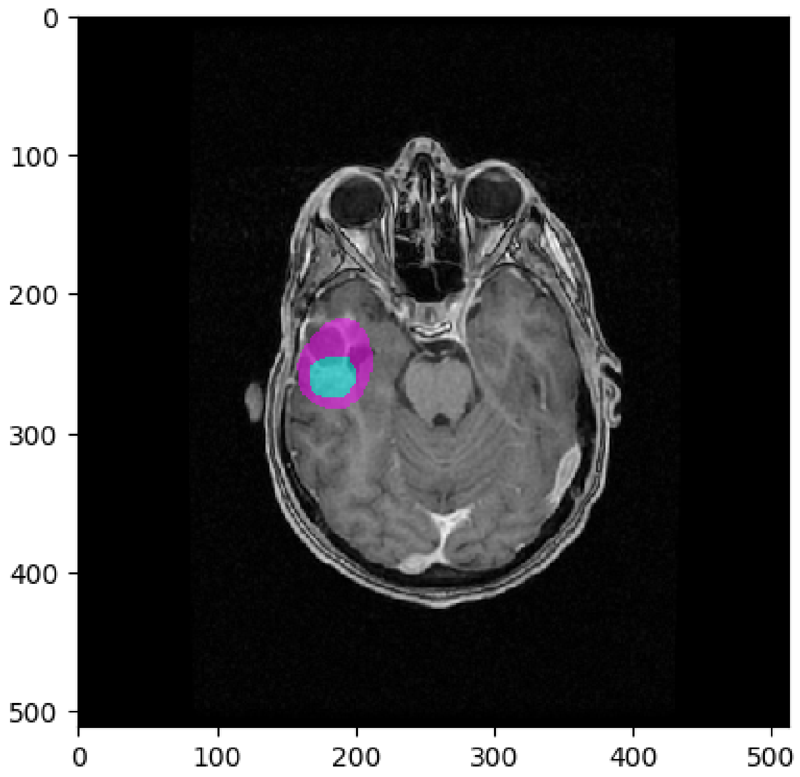
[21]:
```
n_slices_skip = 4
display_slices(image, mask, skip = n_slices_skip) # visualize that our␣
  ↪segmentations were succesfully convereted
```

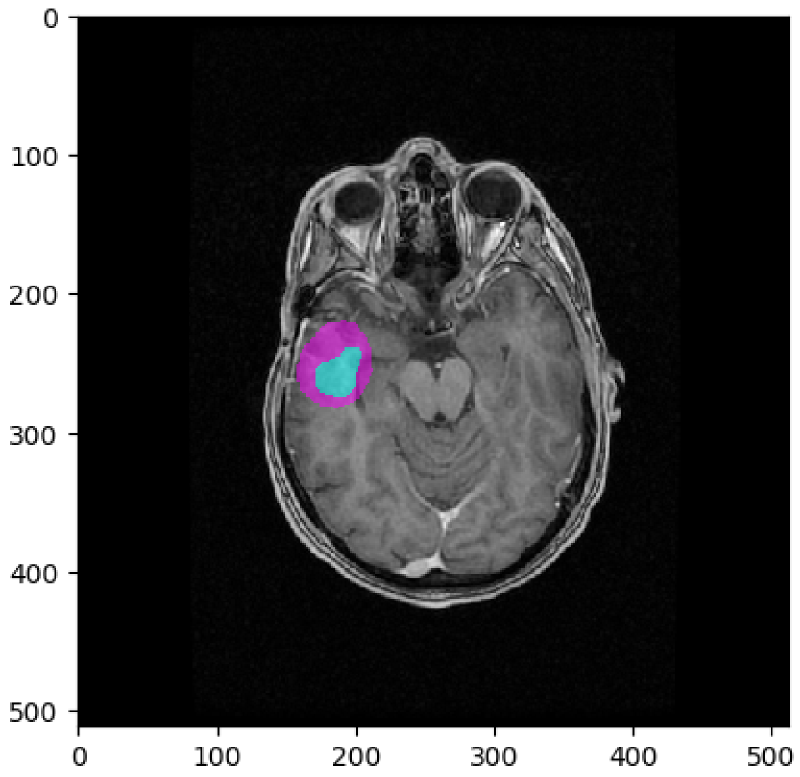


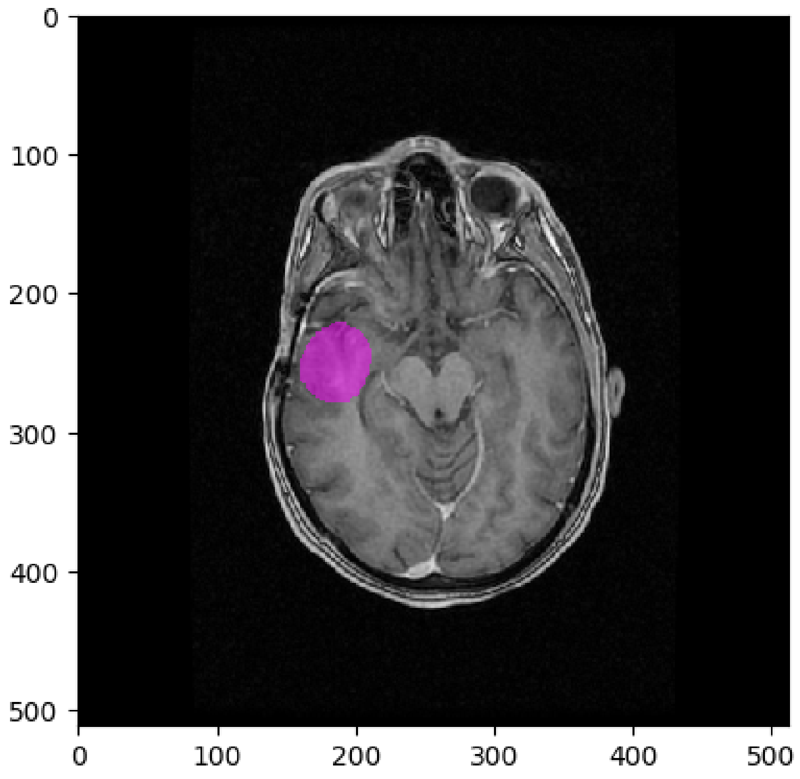

0
100
200
300
400
500
0
100
200
300
400
500

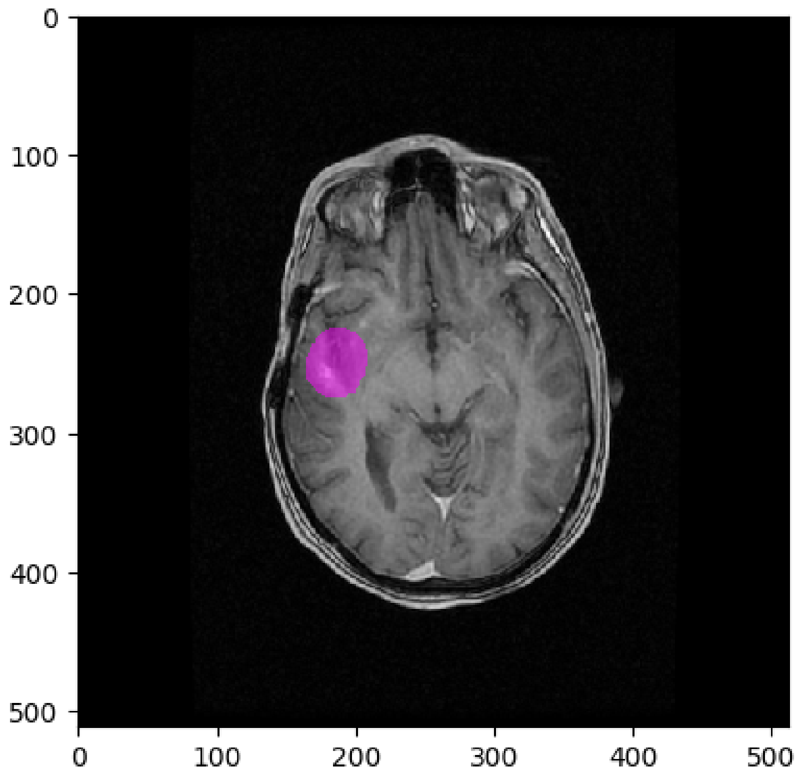

0
100
200
300
400
500
0
100
200
300
400
500

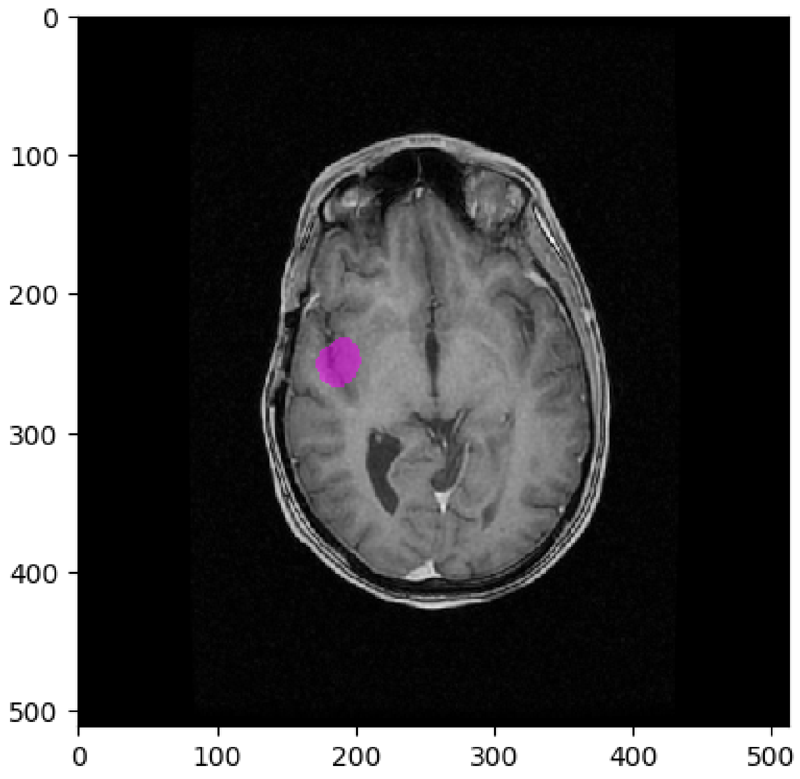

0
100
200
300
400
500
0
100
200
300
400
500


0
100
200
300
400
500
0
100
200
300
400
500


0
100
200
300
400
500
0
100
200
300
400
500


0
100
200
300
400
500
0
100
200
300
400
500





Note: Cyan color denotes tumor while magenta denotes surrounding area of high-dose radiation. Only displaying 7 slices.

1.3 Part 3: Saving arrays to nifti format.

If you want to use a manual approach, you can view the nifti files easily after running get_images_and_mask(). Saving files as nifti is advisable since spacing information is preserved.

```
[24]: nifti_path = os.path.join('.', 'Example_Data', 'Nifti_Data') # nifti subfolder
      if not os.path.exists(nifti_path):
          os.makedirs(nifti_path)
```

```
[25]: dicom_sitk_handle = Dicom_reader.dicom_handle # SimpleITK image handle
      mask_sitk_handle = Dicom_reader.annotation_handle # SimpleITK mask handle
      sitk.WriteImage(dicom_sitk_handle, os.path.join(nifti_path, 'Image.nii'))
      sitk.WriteImage(mask_sitk_handle, os.path.join(nifti_path, 'Mask.nii'))
```

One can also use the built in .write_parallel attribute to generate nifti files for all relevant pairs the DicomReaderWriter object has found/generated. In this case there are 9 image/mask pairs for unique UIDs that contain all contours we are interested in. Note a corresponding log excel file in the specified output path. The nifti files are written in the following format: "Overall_Data_{description}_ {iteration}.nii.gz" (image) or "Overall_mask_{description}_ y{iteration}.nii.gz" (mask).

```
[ ]: %%time
     %%capture
     Dicom_reader.write_parallel(out_path = nifti_path, excel_file = os.path.
       ↪join(nifti_path,'.','MRN_Path_To_Iteration.xlsx'))
```

We can now reload the nifti files and disaply them to check that nothing went wrong. You can inspect the other converted files by changing the numerical suffix as per the excel log file ('MRN_Path_To_Iteration.xlsx').

```
[ ]: nifti_image = sitk.ReadImage(os.path.join(nifti_path,"Overall_Data_Examples_8.
       ↪nii.gz")) # reload image
     image = sitk.GetArrayFromImage(nifti_image)
     nifti_mask = sitk.ReadImage(os.path.join(nifti_path,"Overall_mask_Examples_y8.
       ↪nii.gz")) # reload mask
     mask = sitk.GetArrayFromImage(nifti_mask)
```

```
[ ]: display_slices(image, mask, skip = n_slices_skip) # visualize that our␣
       ↪segmentations were succesfully convereted from nifti
```

1.4 Part 4: Saving and loading numpy files for later use.

Finally we can save the numpy arrays themselves to files for later use (so you don't have to reinstantiate the computationally expensive DicomReaderWriter object) and subsequently re-load the numpy arrays.

```
[ ]: numpy_path = os.path.join(data_path, 'Numpy_Data') # go into numpy subfolder
     if not os.path.exists(numpy_path):
         os.makedirs(numpy_path)
```

```
[ ]: np.save(os.path.join(numpy_path, 'image'), image) # save the arrays
     np.save(os.path.join(numpy_path, 'mask'), mask)
```

```
[ ]: image = np.load(os.path.join(numpy_path,'image.npy')) # load the arrays
     mask = np.load(os.path.join(numpy_path,'mask.npy'))
```

1.5 Part 5: Radiomics Use-case Example.

Here we use the popular open-source radiomics library PyRadiomics (https://pyradiomics.readthedocs.io/en/latest/) to calculate radiomic features for our ROIs. In this case, we only calculate a limited number features from the tumor as an illustrative example.

```
[ ]: try:
         from radiomics import featureextractor
     except:
         !pip install pyradiomics
         from radiomics import featureextractor
```

```
[ ]: pd.set_option('display.max_columns', None) # show all columns
```

```
[ ]: %%time
     # note: need sitk images (sitk.ReadImage(nifti file)) to plug into PyRadiomics,␣
      ↪preserves spacing

     ROI_index = 1 # index for tumor
     nifti_mask_tumor = sitk.BinaryThreshold(nifti_mask, lowerThreshold=ROI_index,␣
      ↪upperThreshold=ROI_index) # select only ROI of interest

     params = {} # can edit in more params as neccessary
     extractor = featureextractor.RadiomicsFeatureExtractor(**params) # instantiate␣
      ↪extractor with parameters
     extractor.disableAllFeatures() # in case where only want some features, can␣
      ↪delete disable/enable lines if you want deafult
     extractor.enableFeatureClassByName('firstorder')
     extractor.enableFeatureClassByName('glcm')
     features = {} # empty dictionary
     features = extractor.execute(nifti_image, nifti_mask_tumor) # unpack results␣
      ↪into features dictionary
     df = pd.DataFrame({k: [v] for k, v in features.items()}) # put dictionary into␣
      ↪a dataframe
```

```
[ ]: df # display dataframe to inspect features
```

Numerical results for radiomic features shown here are consistent with importing nifti files as image and label map in 3D Slicer (https://www.slicer.org/) and using Radiomics extension (https://www.slicer.org/wiki/Documentation/Nightly/Extensions/Radiomics).

1.6 Part 6: Predictions To RT-Structure Example

Here we will provide a simple example for converting a predicted NumPy array of a square into a Dicom RT-Structure file

[ ]:
```
RT_path = os.path.join('Example_Data', 'RT_Structures')
if not os.path.exists(RT_path):
    os.makedirs(RT_path)
```

First, we will create a fake prediction, it will be the same size as the image NumPy array

[ ]:
```
image = Dicom_reader.ArrayDicom
```

Now, deep learning model typically create segmentations in the format of (z_images, rows, cols, # of classes)

[ ]:
```
def create_circular_mask(h, w, center=None, radius=None):

    if center is None: # use the middle of the image
        center = (int(w/2), int(h/2))
    if radius is None: # use the smallest distance between the center and image␣
 ↪walls
        radius = min(center[0], center[1], w-center[0], h-center[1])

    Y, X = np.ogrid[:h, :w]
    dist_from_center = np.sqrt((X - center[0])**2 + (Y-center[1])**2)

    mask = dist_from_center <= radius
    return mask
```

[ ]:
```
predictions = np.zeros(image.shape + (4,))  # Four classes: background, square,␣
 ↪circle, target
predictions.shape
predictions[75:80, 250:350, 100:200, 1] = 1  # Here we are drawing a square
predictions[75:80, 250:350, 300:400, 2] += create_circular_mask(100, 100,␣
 ↪center=None, radius=50).astype('int')
predictions[75:80, 100:200, 200:300, 3] += create_circular_mask(100, 100,␣
 ↪center=None, radius=50).astype('int')
predictions[75:80, 100:200, 200:300, 3] -= create_circular_mask(100, 100,␣
 ↪center=None, radius=33).astype('int')
predictions[75:80, 100:200, 200:300, 3] += create_circular_mask(100, 100,␣
 ↪center=None, radius=15).astype('int')
```

[ ]:
```
display_slices(image, np.argmax(predictions, axis=-1), skip = 1) # visualize␣
 ↪our square on the image
```

Convert the NumPy arrays into RT-Structure

```
[ ]: Dicom_reader.prediction_array_to_RT(prediction_array=predictions,␣
      ↪output_dir=RT_path,
                                         ROI_Names=['square', 'circle', 'target'])
```

2 Final notes

2.0.1 I hope that this code has been useful, if you have any suggestions or problems, please open an issue ticket or merge request on the Github: https://github.com/brianmanderson/Dicom_RT_and_Images_to_Mask

Thank you!

```
[ ]:
```