# Automated Segmentation of Medical Images And Mobile Visualization in 3D

Harshal Soni[1] and Belqes Gharama [2]
Department of Computing Science, Multimedia
University of Alberta
Alberta, Canada
hsoni@ualberta.ca[1], belqes@ualberta.ca[2]

***Abstract*—Visualization of medical images such as magnetic resonance imaging (MRI) and Computerized tomography (CT) in three dimension (3D) is a lucrative option to understand the nature of the scan and decipher the disease. There are many tools [12] available in the market that empowers radiologists to not only visualize these scans into 3D but they allow them to slice, rotate, and change color functions of the reproduced 3D model. However, there is a serious dearth of mobile platforms that can perform the same task on the smartphones. Our project aims at developing a powerful mobile tool that can function as a standalone application and provide the above mentioned functionalities. Furthermore, our mobile application features an automated segmentation tool which extracts all the possible parts of the model based on an user-defined threshold.**

***Index Terms*—projections, segmentation, extraction, region growing, marching cube, registration**

## I. Introduction

### A. Overview

The advent of medical imaging has revolutionized the industry. In modern clinical practices, analysis of medical imaging occurs at all stages beginning from the diagnostics to the planning of the therapeutic procedures, surgical evaluation, and rehabilitation after the procedure.

Since it is not possible to capture all the underlying data of the human body through one image, multiple images are captured which features complementary information of the target organ. In order to envisage all the information captured through these images, a sophisticated *registration* of these images by integrating all the slices into a spatial alignment is required. Once the images are aligned and registered, they are stitched together in 3D via a process termed as *fusion* [1].

### B. Problem Statement

The nature of the information carried by medical images belonging to one class of modality may differ from the images belonging to another modality. Hence, medical software developers have always focused to extend their application to multiple modalities and display them side-by-side or fused them together to present a wholesome view of the organ.

Furthermore, it is especially difficult to present the three-dimensional information of the organs into two-dimensional images. Traditionally, multiple slices of these images are displayed on a computer screen and the radiologist manually attempts to understand the nature of ailment through these slices. Especially, it becomes challenging with the brain and spinal cord as there is no room of error. This introduces an unnecessary delay in the diagnostics which jeopardizes the health of the patient.

Additionally, visualizing any particular part of an organ with heterogeneous density is excruciating because not only it is difficult to visualize but to measure the dimensions of the malignant part is a bigger challenge. Hence, a computer-based visualization tool that can produce a 3D model out of 2D medical scans and segments the requested part out of that scan is needed for accurate diagnostics. It enables radiologists to take decisions faster and accurately saving time and resources.

### C. Research Objective

Developing a mobile platform for automatic segmentation of various parts of the organ captured in a scan and visualize it in a user interpretative 3D model.

## II. Related Work

### A. Main Types of Modalities and Image Format

The most popular modalities that are used for clinical diagnostics are CT-scan and MRI. While the former technique uses X-ray capturing an organ from various angles and stitching them together, MRI uses magnetic waves to capture the same. This makes MRI relatively safer than CT scans.

However, MRI scans are used majorly for inner soft tissues of the brain and skeleton system, CT scans are more efficiently able to capture outer hard regions of the body such as skulls. Also, CT scans are relatively faster to capture making it an optimal choice for the patients with metallic devices such as pacemaker, mentally unstable and children.

The standard protocols [1] for MRI scans are: spin-echo T1-weighted image (T1WI), T2-weighted image (T2WI), T2-weighted MRI with fluid-attenuated inversion recovery (T2 FLAIR) and proton density-weighted image (PDWI) [3].

### B. Survey of Related Technologies

S.J. Taka et al. [4] elaborated in their paper that 3D visualization of medical images happens in either image space or mesh space. Image-space involves interpolating models from image data such as Digital Imaging and Communications

The project is proposed by Dr. Nizam Ahmed and supported by Dr. Kumaradevan Punithakumar.

in Medicine (DICOM), The Neuroimaging Informatics Technology Initiative (nifti), tiff whereas mesh space manipulation involves data in mesh Visualization Toolkit (VTK) format or raw format.

There are various software and packages available for radio medical and biomedical research, 3D visualization and segmentation. The open-source software namely, 3D Slicer, imageJ, Osirix, ParaView, and 3D-Doctor [12] provides GUI-based as well as command-line based interface. All the above-mentioned software allows 3D segmentation but only Osirix provides mobile support in iOS. [1] Hence, there is no support for mobile applications powered by the Android framework.

## C. Main types of Algorithm

The popular algorithms can be classified into two categories: segmentation based and surface rendering based.

*1) Segmentation based algorithms:* The idea behind segmentation based algorithm is to extract the part of the image based on region of Interest (ROI) defined by the user. Following are some of the key algorithms belonging to this category:-

- Region Growing Algorithm: In region-based algorithm [5], initial seed points are defined in the Region of Interest (ROI) and neighboring pixels are classified as in to the region or out of the region. The process is completed until no pixels in the neighborhood is left unclassified. After the classification process is terminated, the pixels of one class are grouped and labeled together.
- Binary Thresholding Algorithm [6]: The algorithm is derived from a simple set of image processing techniques and they work by screening the pixel values above or below the given threshold. For instance, the image is converted into gray-scale image and thresholding is applied onto the image. After the process is performed, the obtained binary mask is superimposed on the original image to make a difference.
- Watershed Algorithm: It is one of the finest algorithms for tumor segmentation [7]. The algorithm can be understood from the following analogy. Each segmented region can be represented as a catchment basin or reservoir and boundaries of that region as a dam. Initially, the basins are empty and the algorithm starts pouring water at a uniform rate. If two basins belong to the same altitude, they are merged into one and a dam is built around them. The process is repeated for all the basins and segmented parts are produced. Here, the altitude of the basin can be replaced with the intensity values of each pixel and dam as the border of the regions.
- Connected Component Algorithm: The connected component algorithm (CCL) [8] is one of the most standard algorithms for segmentation and object classification. It functions by scanning the image from top-bottom, right-left, and diagonally to assign pixels a distinctive label. Then, image is thresholded into foreground and background and labels are merged iteratively until no label changes inside the image.

*2) Surface extraction and Volume rendering based:*

- Marching Cube Algorithm: "Marching cubes are also known as polygonizing a scalar field, 3D Contouring, Marching Cubes, and Surface Reconstruction" [1]. The algorithm [9] functions as follows: A voxel can be defined as a 3D pixel in a scalar field. If the isovalue of each pixel is greater than or lesser than user-defined value, then that voxel contributes the isosurface. In this way connectivity is defined between layers and each intersected edge of the cube is extracted and rendered to create a mesh surface representation in 3D.

*3) Measurement Tools:* B. Preim et al. [17] implemented the measurement tools for a 3D mode. We have used that as a reference and implemented following tools: distance (euclidean), angle (circular), and radii of blob.

## III. Design Methodology

The application is developed as an amalgamation of conventional image processing techniques and automated machine learning techniques. To prevent thresholding of mobile hardware, pre-processing of the data set is carried at the server-side. The application have been implemented using Java 8 , Python 3.6, VTK 8.2 and ITK 4.10 toolkit [15] [16] and server is deployed on local machine.

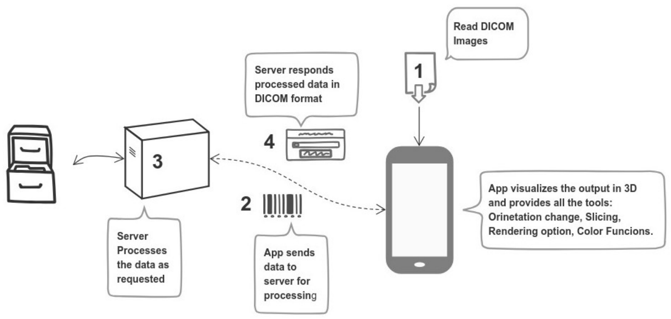

Read DICOM Images
1
Server responds processed data in DICOM format
4
3
2
App visualizes the output in 3D and provides all the tools: Orinetation change, Slicing, Rendering option, Color Funcions.
Server Processes the data as requested
App sends data to server for processing

Fig. 1. Pipeline of the application describing the roles of server and client.

### A. Dataset

The dataset comprises of two subject scans with each consisting of 3D volumetric MRI and CT data [2]. MRI data has T1, T2, T1CE, and FLAIR images of same subject.

### B. Pre-processing

- Image Acquisition: User selects the DICOM file from GUI and selection is reflected to the server.
- Skull-Stripping: The server reads the file and launches it for skull-stripping process. A CNN-based network based on [18] is used for skull-stripping.
- Filtering: Binary thresholding and CurvatureFlowImageFilter [13] [14] are used for removing noise from the skull-stripped dataset.
- Loading: Processed data is pushed back to centrally hosted dataset and it is read at client-end using a Grassroot DICOM reader (GDCM).

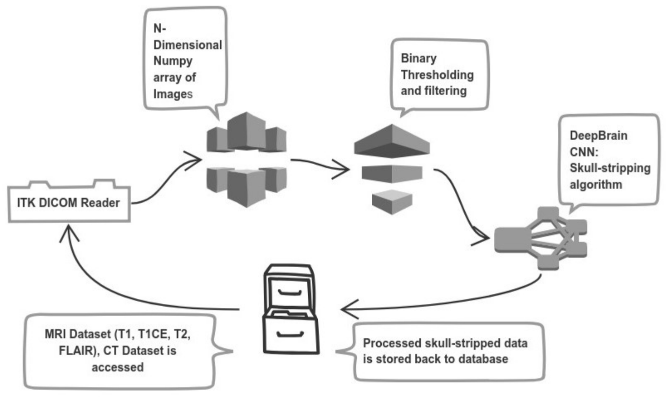

N-Dimensional Numpy array of Images
Binary Thresholding and filtering
DeepBrain CNN: Skull-stripping algorithm
ITK DICOM Reader
MRI Dataset (T1, T1CE, T2, FLAIR), CT Dataset is accessed
Processed skull-stripped data is stored back to database

Fig. 2. Pre-processing pipeline of our application that happens at server-end.

### C. Processing

1) Region Growing Algorithms: Once the images are processed, each slice of the scan is loaded and region growing algorithm [5] is applied based on a seed threshold value defined by the user from Graphical User Interface(GUI) of the application. This will segment all the possible parts of the organ for that slice.

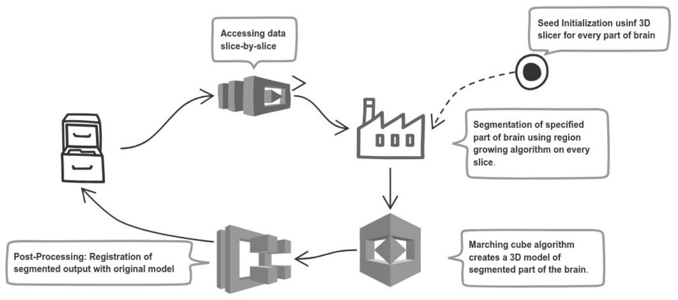

Accessing data slice-by-slice
Seed Initialization usinf 3D slicer for every part of brain
Segmentation of specified part of brain using region growing algorithm on every slice.
Post-Processing: Registration of segmented output with original model
Marching cube algorithm creates a 3D model of segmented part of the brain.

Fig. 3. Processing pipeline of our application that happens at client-end on smartphone.

2) Smoothing: The morphological operation of closing is applied to the segmented parts for a better discrimination. Sobel filter [3] is applied to the output image to remove any holes or unnecessary debris on the edge of the segmented part.
3) Feature Extraction: Once segmented contour of the user-requested part of the scan is derived, important features are extracted from it namely, contrast, area of the segmented area, solidity of the region (concavity of the segment), and intensity.
4) 3D Reconstruction: The extracted segments of the scan are supplied as ”data” to marching cube algorithm [9] where vertices and faces of the 3D model is calculated. The final model is saved in ”.stl” format. The reason for opting ’.stl” format is because it provides greater flexibility to operate on it such as slicing, rotating, measuring et cetera [10].

### D. Visualization of 3D model

The android application provides tools as a GUI for various user-controlled visualization [11] and features. Since, the processed segments and the model is saved in ”.stl” format, manoeuvring the model is easy [15]. Follows is the list of features supported by the application at present:

| Title | Description |
|---|---|
| Segmentation | Automatic segmentation of model based on user-defined threshold. |
| Region Growth | Grows the connected part from the segmented part taking into account user’s threshold. |
| Transparency slider | Adjust the transparency of image as well as segmented region based on user’s input. |
| View Changer | Changes the projection views- 3D, transverse, coronal, sagittal, and oblique. |
| Render Type | Changes render mode- Iso surface, Ray Casting, Volume rendering, Solid- shading, Maximum intensity projection, X-ray simulation, region rendering. |
| Color Table | Changes the color function from three saved functions. |
| Slider | Located at the bottom, slider changes the present visible slide of the scan. |

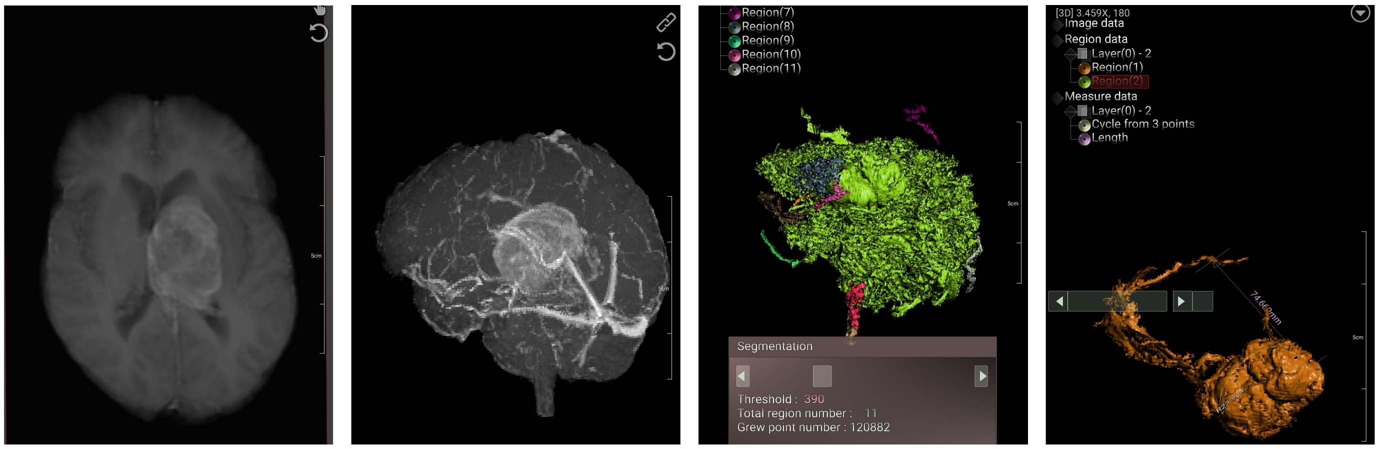
## IV. Evaluation and Results

Our mobile platform runs seamlessly on Android devices consuming no bandwidth after dataset is downloaded from the server after pre-processing. Figure 4 displays the snapshot of the application interface for different functions.

In order to evaluate the efficiency of our application, we measured standard Frames per Second (FPS), CPU consumption, and GPU consumption on Samsung Galaxy S9. We measured these metrics using OpenGL android API to measure system performance. As suggested by Figure 5, our application ran smoothly with **55% CPU usage** on an average and **18% GPU usage** on an average during segmentation task which is deemed to be the most computationally expensive of all the tasks. Also, our application delivered a stable **45 FPS** during the touch interactions for the change in orientation(rotation + translation + scaling).

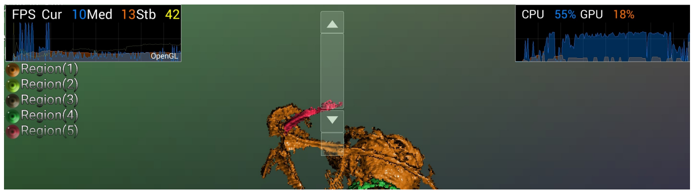
## V. Individual Contributions

Mr. Harshal has been responsible for System Architecture. His roles include client-server integration, processing steps of the application namely, region growing, marching cube, and model generation. Also, he developed the automated segmentation algorithm and implemented tools for manipulating the 3D model such as slicing and orientation changes.



Threshold : 390
Total region number : 11
Grew point number : 120882

Fig. 4. Snapshot saved for different functions. (left to right) Once Dataset is loaded, it is displayed in one of the three 2D projection view, here it can be seen in transverse view. The next output demonstrates the 3D volume rendered model of brain with maximum intensity projection rendering. The next image is the segmented output of all the parts of brain for the threshold of 390. Finally, segmented output of Glioblastoma Multiforme(GBM) tumour is displayed in 3D along with its measured dimensions.



FPS Cur 10Med 13Stb 42
CPU 55% GPU 18%

Fig. 5. Evaluation of our application. (left-pane) we are getting standard FPS of 45, (right-pane) CPU usage of 55%, and GPU usage of 18% during our test.

Color transform functions, projection views, application-level integration, database acquisition, tools for measurements, and testing has been jointly coordinated as a team.

## VI. Conclusion

We developed a robust mobile application for the segmentation of brain and its visualization. The application is only dependent on server for computationally expensive skull-stripping process and all the other features ranging from segmentation to model manipulation can be performed on smart phones. It is a powerful, robust, and compact tool that can aid radiologists to diagnose an ailment accurately. The source code can be accessed here: Sourcecode

## Acknowledgment

The authors acknowledge Dr. Nizam Ahmed, dept. of Medicine for proposing the idea and Dr. Kumaradevan Punithakumar, dept. of Radiology & Diagnostic Imaging for his guidance and welcoming support.

## References

[1] Al-Rei, Mona. “Automated 3D Visualization of Brain Cancer.” (2017).
[2] Clark, Kenneth et al. “The Cancer Imaging Archive (TCIA): maintaining and operating a public information repository.” Journal of digital imaging vol. 26,6 (2013): 1045-57. doi:10.1007/s10278-013-9622-7
[3] S. Bauer, R. Wiest, L.-P. Nolte, and M. Reyes, —A survey of MRI-based medical image analysis for brain tumor studies, Phys. Med. Biol., vol. 58, no. 13, p. R97, 2013
[4] S. J. Taka and S. Srinivasan, —NIRViz: 3D visualization software for multimodality optical imaging using visualization toolkit (VTK) and insight segmentation toolkit (ITK), J. Digit. Imaging, vol. 24, no. 6, pp. 1103–1111, 2011.
[5] S. Kamdi and R. K. Krishna, —Image Segmentation and Region Growing Algorithm, Int. J. Comput. Technol. Electron. Eng., vol. 2, no. 1, pp. 2249–6343, 2012.
[6] P. K. Sahoo, S. Soltani, and A. K. C. Wong, —A survey of thresholding techniques, Comput. Vision, Graph. Image Process., vol. 41, no. 2, pp. 233–260, 1988.
[7] S. D. Salman and A. A. Bahrani, —Segmentation of tumor tissue in gray medical images using watershed transformation method, Int. J. Adv. Comput. Technol., vol. 2, no. 4, 2010.
[8] M. Shafry et al., —Connected Component Labeling Using Components NeighborsScan Labeling Approach, J. Comput. Sci., vol. 6, no. 10, pp. 1099–1107, 2010.
[9] W. E. Lorensen and H. E. Cline, —Marching cubes: A high resolution 3D surface construction algorithm, Comput. Graph. (ACM)., vol. 21, no. 4, pp. 163–169, 1987
[10] A. Som, —Surface Extraction: Creating a mesh from pixel-data using Python and VTK, PyScience, 2014. [Online].
[11] B. Guo, —A Multiscale Model for Structure-Based Volume Rendering, IEEE Trans. Vis. Comput. Graph., vol. 1, no. 4, pp. 291–301, 1995.
[12] J. J. Caban, A. Joshi, and P. Nagy, —Rapid development of medical imaging tools with open-source libraries, J. Digit. Imaging, vol. 20, no. 1, pp. 83–93, 2007.
[13] R. H. Chan, C. W. Ho, and M. Nikolova, —Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization, IEEE Trans. Image Process., vol. 14, no. 10, pp. 1479–1485, 2005.
[14] P. Kalavathi and P. Sowdeeswari, —Edge Enhanced Anisodiffusion Filter for Denoising Gaussian Noise in Medical Images, Int. J. Control Theory Appl., vol. 9, no. 27, pp. 203–210, 2016.
[15] Hanwell, Marcus D., et al. ”The Visualization Toolkit (VTK): Rewriting the rendering code for modern graphics cards.” SoftwareX 1 (2015): 9-12.
[16] Schroeder, Will, Lydia Ng, and Josh Cates. ”The ITK software guide.” (2003).
[17] ] B. Preim, C. Tietjen, W. Spindler, and H.-O. Peitgen, —Integration of Measurement Tools in Medical 3d Visualizations, in in Proceedings of the conference on Visualization’02, IEEE Computer Society , 2002, pp. 21–28.
[18] Konstantinos Kamnitsas, Liang Chen, Christian Ledig, Daniel Rueckert, and Ben Glocker, “Multi-Scale 3D CNNs for segmentation of brain Lesions in multi-modal MRI”, in proceeding of ISLES challenge, MICCAI 2015.

## Source Code

### *0.1 Server-side*

Our application relies on server for the pre-processing task. There are two main tasks which are being carried at server-end: Noise removal and skull-stripping. We have deployed a RTMP server on our local machine. It is a threaded python based server which accepts connections from the application, reads the file requested and initiates the CNN model for pre-processing. Once, pre-processing step is finished, it automatically notifies the application to continue the volume rendering.

#### *0.1.1 Server Skeleton*

```
def run(self):
 threading.Thread(target=self.recvRtspRequest).start()

def recvRtspRequest(self):
 """Receive RTSP request from the client."""
 connSocket = self.clientInfo['rtspSocket'][0]
 while True:
  data = connSocket.recv(256)  ###
  if data:
   print '-'*60 + "\nData received:\n"
           + '-'*60
   self.processRtspRequest(data)

def processRtspRequest(self, data):
 """Process RTSP request sent from the client."""
 # Get the request type
 request = data.split('\n')
 line1 = request[0].split(' ')
 requestType = line1[0]
 # Get the media file name
 filename = line1[1]
 # Get the RTSP sequence number
 seq = request[1].split(' ')
 # Process SETUP request
 if requestType == self.SETUP:
 if self.state == self.INIT:
 # Update state
 print "SETUP Request received\n"

 self.clientInfo['videoStream'] = VideoStream(filename)
    self.state = self.READY
    except IOError:
 self.replyRtsp(self.FILE_NOT_FOUND_404, seq[1])
 # Generate a randomized RTSP session ID
 self.clientInfo['session'] = randint(100000, 999999)
 # Send RTSP reply
 self.replyRtsp(self.OK_200, seq[0])
 print "sequenceNum is " + seq[0]
 self.deepmedic.frontend.testingSession( self.data)
 print "Testing initiated"
```

```
# Get the RTP/UDP port from the last line
self.clientInfo['rtpPort'] = request[2].split(' ')[3]
print '-'*60 + "\nrtpPort is :" +
        self.clientInfo['rtpPort'] +
                                 "\n" + '-'*60
print "filename is " + filename
```

**Listing 1** Snippet of our server responding to the client and initiating CNN network for pre-processing.

### *0.1.2 Server hardware Details*

We have trained and tested our model on Intel Xeon Gold 6234 (3.30GHz,up to 4.00GHz with Turbo Boost, 8 Cores, 25MB Cache), RTX Quadro 5000 16 GB GDDR6 and RAM: 32GB + 12GB OS. The faster processor allowed us to host the server inthe local machine, which helped us learn more about network programming.

### *0.1.3 Server Pre-processing Algorithm*

We have used Deepmind-medic skull stripper CNN model for our application and designed it in such a way that it pre-processes the data automatically before loading it to the network.

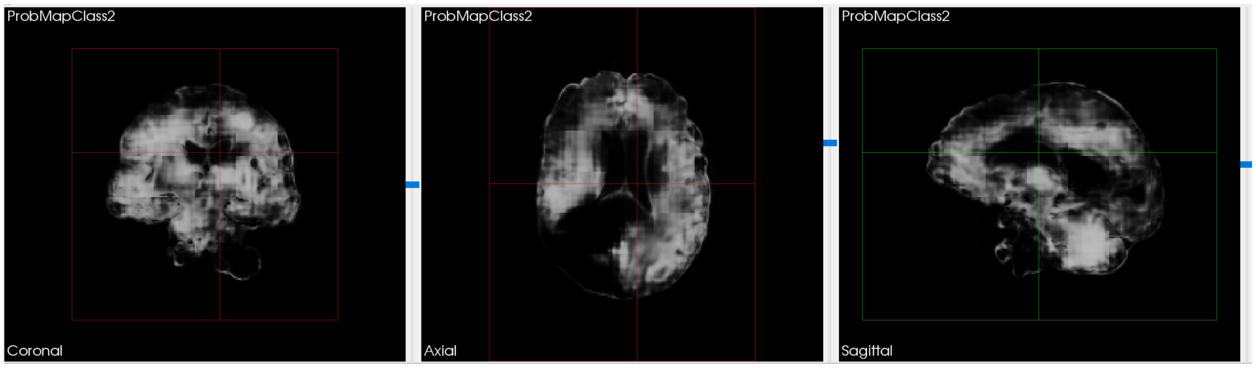


**Fig 1** Probability heat maps are generated during pre-processing

To run the pre-processing steps, install the following dependencies:

- Tensorflow $>=$ 1.6
- Python $>=$ 3.5
- Nibabel $>$ 3
- scipy
- numpy

### *0.1.4 Server Training*

We have added 8 layers to our CNN model with a batch size of 10 and learning rate = 0.001. Model is trained for 100 epoches.

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 15509835338190025920
]
2020-4-10 19:37:37.30: CONFIG: The configuration file for the [model] given is
    : C:\CaPTk_Full\1.7.7.nonRelease.20200323.4d06b4e7\data\deepMedic\
    saved_models\brainTumorSegmentation\modelConfig.txt
2020-4-10 19:37:37.31:
    =============================================================
2020-4-10 19:37:37.31: ========== PARAMETERS FOR MAKING THE ARCHITECTURE
    ===========
2020-4-10 19:37:37.32:
    =============================================================
2020-4-10 19:37:37.32: CNN model name = dm1S37
2020-4-10 19:37:37.33: ~~~~~~~~~~~~~~~~~~Model parameters~~~~~~~~~~~~~~~~
2020-4-10 19:37:37.33: Number of Classes (including background) = 4
2020-4-10 19:37:37.34: ~~Normal Pathway~~
2020-4-10 19:37:37.34: Number of Input Channels = 4
2020-4-10 19:37:37.35: Number of Layers = 8
2020-4-10 19:37:37.35: Number of Feature Maps per layer = [30, 30, 40, 40, 40,
     40, 50, 50]
2020-4-10 19:37:37.35: Kernel Dimensions per layer = [[3, 3, 3], [3, 3, 3],
    [3, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 3]]
2020-4-10 19:37:37.36: Receptive Field = [17, 17, 17]
2020-4-10 19:37:37.36: Residual connections added at the output of layers (
    indices from 0) = [3, 5, 7]
2020-4-10 19:37:37.37: Layers that will be made of Lower Rank (indices from 0)
     = []
2020-4-10 19:37:37.37: Lower Rank layers will be made of rank = []
2020-4-10 19:37:37.38: ~~Subsampled Pathway~~
2020-4-10 19:37:37.38: Use subsampled Pathway = True
2020-4-10 19:37:37.39: Number of subsampled pathways that will be built = 2
2020-4-10 19:37:37.39: Number of Layers (per sub-pathway) = [8, 8]
2020-4-10 19:37:37.40: Number of Feature Maps per layer (per sub-pathway) =
    [[30, 30, 40, 40, 40, 40, 50, 50], [30, 30, 40, 40, 40, 40, 50, 50]]
2020-4-10 19:37:37.40: Kernel Dimensions per layer = [[3, 3, 3], [3, 3, 3],
    [3, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 3]]
2020-4-10 19:37:37.41: Receptive Field = [17, 17, 17]
2020-4-10 19:37:37.41: Subsampling Factor per dimension (per sub-pathway) =
    [[3, 3, 3], [5, 5, 5]]
2020-4-10 19:37:37.42: Residual connections added at the output of layers (
    indices from 0) = [3, 5, 7]
2020-4-10 19:37:37.42: Layers that will be made of Lower Rank (indices from 0)
     = []
2020-4-10 19:37:37.43: Lower Rank layers will be made of rank = []
2020-4-10 19:37:37.43: ~~Fully Connected Pathway~~
```

```
2020-4-10 19:37:37.44: Number of additional FC layers (Excluding the Classif.
    Layer) = 2
2020-4-10 19:37:37.45: Number of Feature Maps in the additional FC layers =
    [250, 250]
2020-4-10 19:37:37.45: Residual connections added at the output of layers (
    indices from 0) = [1]
2020-4-10 19:37:37.46: Layers that will be made of Lower Rank (indices from 0)
     = []
2020-4-10 19:37:37.46: Dimensions of Kernels in the 1st FC layer (Classif.
    layer if no hidden FCs used) = [3, 3, 3]
2020-4-10 19:37:37.47: ~~Size Of Image Segments~~
2020-4-10 19:37:37.47: Size of Segments for Training = [37, 37, 37]
2020-4-10 19:37:37.48: Size of Segments for Validation = [17, 17, 17]
2020-4-10 19:37:37.48: Size of Segments for Testing = [45, 45, 45]
2020-4-10 19:37:37.49: ~~Dropout Rates~~
2020-4-10 19:37:37.50: Drop.R. for each layer in Normal Pathway = []
2020-4-10 19:37:37.50: Drop.R. for each layer in Subsampled Pathway = []
2020-4-10 19:37:37.51: Drop.R. for each layer in FC Pathway (additional FC
    layers + Classific.Layer at end) = [0.0, 0.5, 0.5]
2020-4-10 19:37:37.51: ~~Weight Initialization~~
2020-4-10 19:37:37.52: Initialization method and params for the conv kernel
    weights = ['fanIn', 2]
2020-4-10 19:37:37.52: ~~Activation Function~~
2020-4-10 19:37:37.53: Activation function to use = prelu
2020-4-10 19:37:37.53: ~~Batch Normalization~~
2020-4-10 19:37:37.54: Apply BN straight on pathways' inputs (eg straight on
    segments) = [False, False, True]
2020-4-10 19:37:37.55: Batch Normalization uses a rolling average for
    inference, over this many batches = 60
2020-4-10 19:37:37.55: ========== Done with printing session's parameters
    ==========
2020-4-10 19:37:37.56:
    ============================================================
```

**Listing 2** Snippet of the log file displays the parameters of our CNN Network.

```
     self._log.print3("")
         self._log.print3("=============   NEW TRAINING SESSION
    ==============\n")
         self._params.print_params()
         self._log.print3("
    =======================================================\n")

         return self._params

     def run_session(self, *args):
         (sess_device,
          model_params,
          reset_trainer) = args

         graphTf = tf.Graph()

         with graphTf.as_default():
             # Explicit device assignment, throws an error if GPU is specified
```

```
but not available.
        with graphTf.device(sess_device):
            self._log.print3("=========== Making the CNN graph...
===============")
            cnn3d = Cnn3d()
            with tf.variable_scope("net"):
                cnn3d.make_cnn_model(*model_params.get_args_for_arch())
                # I have now created the CNN graph. But not yet the
Optimizer's graph.

        # No explicit device assignment for the rest.
        # Because trained has piecewise_constant that is only on cpu, and
so is saver.
        with tf.variable_scope("trainer"):
            self._log.print3("=========== Building Trainer ===========\n")
            trainer = Trainer(*(self._params.get_args_for_trainer() + [
cnn3d]))
            trainer.create_optimizer(*self._params.get_args_for_optimizer
())  # Trainer and net connect here.

        tensorboard_loggers = self.create_tensorboard_loggers(['train', '
val'],
                                                               graphTf,
                                                               create_log=
self._params.get_tensorboard_bool())

        # The below should not create any new tf.variables.
        self._log.print3("=========== Compiling the Training Function
===========")
        self._log.print3("
=======================================================\n")
        cnn3d.setup_ops_n_feeds_to_train(self._log,
                                         trainer.get_total_cost(),
                                         trainer.
get_param_updates_wrt_total_cost()  # list of ops
                                         )

        self._log.print3("=========== Compiling the Validation Function
=========")
        cnn3d.setup_ops_n_feeds_to_val(self._log)

        self._log.print3("=========== Compiling the Testing Function
============")
        cnn3d.setup_ops_n_feeds_to_test(self._log,
                                        # For validation with full
segmentation
                                        self._params.
indices_fms_per_pathtype_per_layer_to_save)

        # Create the savers
        saver_all = tf.train.Saver()  # Will be used during training for
saving everything.
        # Alternative: tf.train.Saver([v for v in tf.all_variables() if v.
name.startswith("net"])
```

```
         collection_vars_net = tf.get_collection(tf.GraphKeys.
GLOBAL_VARIABLES, scope="net")
         saver_net = tf.train.Saver(var_list=collection_vars_net)  # Used
to load the net's parameters.
         collection_vars_trainer = tf.get_collection(tf.GraphKeys.
GLOBAL_VARIABLES, scope="trainer")
         saver_trainer = tf.train.Saver(var_list=collection_vars_trainer)
# Used to load the trainer's parameters.

     # self._print_vars_in_collection(collection_vars_net, "net")
     # self._print_vars_in_collection(collection_vars_trainer, "trainer")

     with tf.Session(graph=graphTf,
                     config=tf.ConfigProto(log_device_placement=False,
                                           device_count={'CPU': 999, 'GPU':
 99})) as sessionTf:
         # Load or initialize parameters
         file_to_load_params_from = self._params.
get_path_to_load_model_from()
         if file_to_load_params_from is not None:  # Load params
             self._log.print3("=========== Loading parameters from
specified saved model ===============")
             chkpt_fname = tf.train.latest_checkpoint(
file_to_load_params_from) \
                 if os.path.isdir(file_to_load_params_from) else
file_to_load_params_from
             self._log.print3("Loading checkpoint file:" + str(chkpt_fname)
)
             self._log.print3("Loading network parameters...")
             try:
                 saver_net.restore(sessionTf, chkpt_fname)
                 self._log.print3("Network parameters were loaded.")
             except Exception as e:
                 handle_exception_tf_restore(self._log, e)

             if not reset_trainer:
                 self._log.print3("Loading trainer parameters...")
                 saver_trainer.restore(sessionTf, chkpt_fname)
                 self._log.print3("Trainer parameters were loaded.")
             else:
                 self._log.print3("Reset of trainer parameters was
requested. Re-initializing them...")
                 tf.variables_initializer(var_list=collection_vars_trainer)
.run()
                 self._log.print3("Trainer parameters re-initialized.")
         else:
             self._log.print3("=========== Initializing network and trainer
 variables  ===============")
             # Initializes all.
             # tf.variables_initializer(var_list = tf.get_collection(tf.
GraphKeys.GLOBAL_VARIABLES) ).run()
             # Initialize separate as below, so that in case I miss a
variable, I will get an error and I will know.
             tf.variables_initializer(var_list=collection_vars_net).run()
```

```
                tf.variables_initializer(var_list=collection_vars_trainer).run
()
                self._log.print3("All variables were initialized.")

                filename_to_save_with = self._params.filepath_to_save_models +
 ".initial." + datetime_now_str()
                self._log.print3("Saving the initial model at:" + str(
filename_to_save_with))
                saver_all.save(sessionTf, filename_to_save_with+".model.ckpt",
 write_meta_graph=False)
                # tf.train.write_graph( graph_or_graph_def=sessionTf.graph.
as_graph_def(),
                # logdir="", name=filename_to_save_with+".graph.pb", as_text=
False)

           self._log.print3("")
           self._log.print3("
=========================================================")
           self._log.print3("============== Training the CNN model
=================")
           self._log.print3("
=========================================================")

           do_training(*([sessionTf, saver_all, cnn3d, trainer,
tensorboard_loggers] + self._params.get_args_for_train_routine()))

       self._log.print3("\n
=========================================================")
       self._log.print3("=========== Training session finished
=================")
       self._log.print3("
=========================================================")
```

**Listing 3** Snippet of the training session of our CNN model used for skull-stripping

### *0.1.5 Server Optimization*

```
############## Optimizer and schedules follows ##############
    # This is independent of the call to setup_costs (can be called before).
   Can be modularized.
    def create_optimizer(   self,
                            log,
                            sgd0orAdam1orRmsProp2,
                            lr_sched_params, # Could be given to init and
   saved there.
                            learning_rate_init,
                            momentum_init,
                            classicMomentum0OrNesterov1,
                            momentumTypeNONNormalized0orNormalized1,
                            b1ParamForAdam,
                            b2ParamForAdam,
                            epsilonForAdam,
                            rhoParamForRmsProp,
                            epsilonForRmsProp
```

```
                        ) :
    log.print3("...Initializing state of the optimizer...")

    self._lr_sched_params = lr_sched_params

    # Learning rate and momentum
    self._init_lr_tfv = tf.Variable(learning_rate_init, dtype="float32",
trainable=False, name="init_lr")
    # This is important for the learning rate schedule to work.
    self._curr_lr =  self._get_lr_from_schedule()

    # SGD and RMS only.
    self._init_mom_tfv = tf.Variable(momentum_init, dtype="float32",
trainable=False, name="init_mom")
    self._curr_mom = self._get_mom_from_schedule()

    # Optimizer
    params_to_opt = self._net.get_trainable_params(log, self.
_indicesOfLayersPerPathwayTypeToFreeze)
    if sgd0orAdam1orRmsProp2 == 0:
        self._optimizer = optimizers_dm.SgdOptimizer( params_to_opt,
                                                      self._curr_lr,
                                                      self._curr_mom,

momentumTypeNONNormalized0orNormalized1,

classicMomentum0OrNesterov1 )
    elif sgd0orAdam1orRmsProp2 == 1:
        self._optimizer = optimizers_dm.AdamOptimizer( params_to_opt,
                                                       self._curr_lr,
                                                       b1ParamForAdam,
                                                       b2ParamForAdam,
                                                       epsilonForAdam )
    elif sgd0orAdam1orRmsProp2 == 2:
        self._optimizer = optimizers_dm.RmsPropOptimizer( params_to_opt,
                                                          self._curr_lr,
                                                          self._curr_mom,

momentumTypeNONNormalized0orNormalized1,

classicMomentum0OrNesterov1,

rhoParamForRmsProp,

epsilonForRmsProp  )
```

**Listing 4** Snippet of our Optimizer during the training phase.

### *0.1.6 Server Testing*

Once the model is trained and deployed, during the testing stage, file requested from the client is read in real-time, and the model generates the output and stores in the database. This processed data is read by the client when notified by the server.

```
def run_session(self, *args):
```

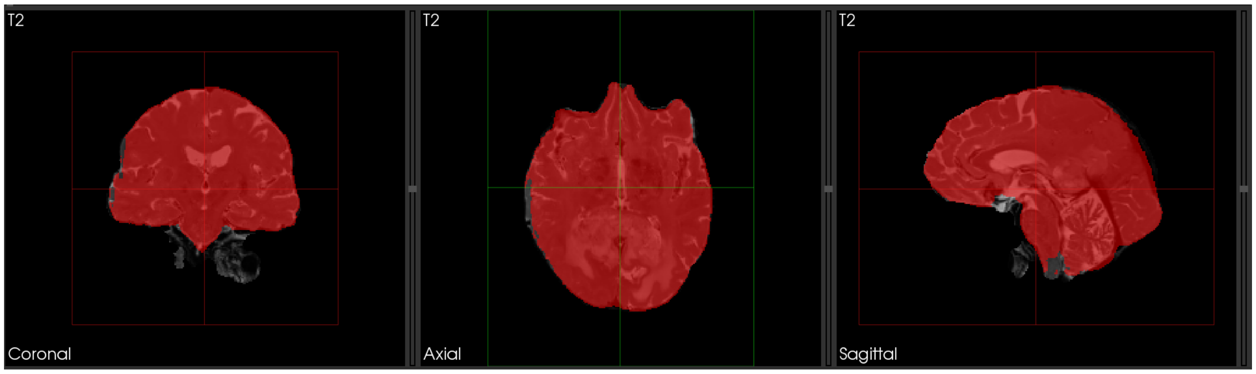


**Fig 2** Output of the skull-stripping algorithm at the end of pre-processing

```
        (sess_device,
         model_params,) = args

        graphTf = tf.Graph()

        with graphTf.as_default():
            with graphTf.device(sess_device): # Throws an error if GPU is
specified but not available.
                self._log.print3("=========== Making the CNN graph...
===============")
                cnn3d = Cnn3d()
                with tf.variable_scope("net"):
                    cnn3d.make_cnn_model( *model_params.get_args_for_arch() )
# Creates the network's graph (without optimizer).

            self._log.print3("=========== Compiling the Testing Function
============")
            self._log.print3("
=======================================================\n")

            cnn3d.setup_ops_n_feeds_to_test( self._log,
                                             self._params.
indices_fms_per_pathtype_per_layer_to_save )
            # Create the saver
            saver_all = tf.train.Saver() # saver_net would suffice

        with tf.Session( graph=graphTf, config=tf.ConfigProto(
log_device_placement=False, device_count={'CPU':999, 'GPU':99}) ) as
sessionTf:
            file_to_load_params_from = self._params.
get_path_to_load_model_from()
            if file_to_load_params_from is not None: # Load params
                self._log.print3("=========== Loading parameters from
specified saved model ===============")
                chkpt_fname = tf.train.latest_checkpoint(
file_to_load_params_from ) if os.path.isdir( file_to_load_params_from )
else file_to_load_params_from
                self._log.print3("Loading parameters from:" + str(chkpt_fname)
)
                try:
                    saver_all.restore(sessionTf, chkpt_fname)
                    self._log.print3("Parameters were loaded.")
                except Exception as e: handle_exception_tf_restore(self._log,
e)

            else:
                self._ask_user_if_test_with_random() # Asks user whether to
continue with randomly initialized model. It exits if no is given.
                self._log.print3("")
                self._log.print3("=========== Initializing network variables
===============")
                tf.variables_initializer( var_list = tf.get_collection(tf.
GraphKeys.GLOBAL_VARIABLES, scope="net") ).run()
                self._log.print3("Model variables were initialized.")


            self._log.print3("")
            self._log.print3("
======================================================")
            self._log.print3("=========== Testing with the CNN model
===============")
            self._log.print3("
======================================================")

            res_code = inference_on_whole_volumes( *( [sessionTf, cnn3d] +
self._params.get_args_for_testing() ) )

        self._log.print3("")
        self._log.print3("
======================================================")
        self._log.print3("=========== Testing session finished
=================")
        self._log.print3("
======================================================")
```

**Listing 5** Snippet of our testing session ran at the server for incoming client request

```
2020-4-10 19:37:47.13:
    ############################################################################

2020-4-10 19:37:47.13: ############################# Starting full
    Segmentation of Testing subjects ##########################
2020-4-10 19:37:47.14:
    #############################################################################

2020-4-10 19:37:47.14:
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

2020-4-10 19:37:47.15: ~~~~~~~~~~~~~~~~~~~~ Segmenting subject with index #0
    ~~~~~~~~~~~~~~~~~~~~
2020-4-10 19:37:47.15: Loading subject with 1st channel at: C:\Users\harsh\
    OneDrive\Desktop\MM804_graphics\Brain_tumour\t1_normalized.nii.gz
2020-4-10 19:37:48.28: Starting to (tile) extract Segments from the images of
    the subject for Segmentation...
```

```
2020-4-10 19:37:48.29: Finished (tiling) extracting Segments from the images
   of the subject for Segmentation.
2020-4-10 19:37:48.29: Starting to segment each image-part by calling the cnn.
   cnnTestModel(i). This part takes a few mins per volume...
2020-4-10 19:37:48.30: Total number of Segments to process:450
2020-4-10 19:37:48.30: Processed 10/450 segments.
2020-4-10 19:42:21.67: Processed 90/450 segments.
2020-4-10 19:47:27.99: Processed 180/450 segments.
2020-4-10 19:52:33.64: Processed 270/450 segments.
2020-4-10 19:57:36.88: Processed 360/450 segments.
2020-4-10 20:2:35.74: Processed 450/450 segments.
2020-4-10 20:3:8.89: TIMING: Segmentation of subject: [Extracting:] 0.39 [
   Loading:] 0.74 [ForwardPass:] 1519.25 [Total:] 1520.38 secs.
2020-4-10 20:3:9.14: Saving the new label (segmentation) image for the subject
    #0
2020-4-10 20:3:9.32: Image saved at: C:\Users\harsh\OneDrive\Desktop\
   MM804_graphics\Brain_tumour\predictions\testApiSession\predictions\Segm.
   nii.gz
2020-4-10 20:3:9.34: Saving the new label (segmentation) image for the subject
    #0
2020-4-10 20:3:10.00: Image saved at: C:\Users\harsh\OneDrive\Desktop\
   MM804_graphics\Brain_tumour\predictions\testApiSession\predictions\
   ProbMapClass0.nii.gz
2020-4-10 20:3:10.03: Saving the new label (segmentation) image for the
   subject #0
2020-4-10 20:3:11.06: Image saved at: C:\Users\harsh\OneDrive\Desktop\
   MM804_graphics\Brain_tumour\predictions\testApiSession\predictions\
   ProbMapClass1.nii.gz
2020-4-10 20:3:11.09: Saving the new label (segmentation) image for the
   subject #0
2020-4-10 20:3:12.16: Image saved at: C:\Users\harsh\OneDrive\Desktop\
   MM804_graphics\Brain_tumour\predictions\testApiSession\predictions\
   ProbMapClass2.nii.gz
2020-4-10 20:3:12.18: Saving the new label (segmentation) image for the
   subject #0
2020-4-10 20:3:13.24: Image saved at: C:\Users\harsh\OneDrive\Desktop\
   MM804_graphics\Brain_tumour\predictions\testApiSession\predictions\
   ProbMapClass3.nii.gz
2020-4-10 20:3:13.24: TIMING: Testing process lasted: 1526.10 secs.
2020-4-10 20:3:13.25: ##########################################
2020-4-10 20:3:13.26: ############# Finished full Segmentation of Testing
   subjects ################
2020-4-10 20:3:13.26:
   #############################################################
2020-4-10 20:3:13.33:
2020-4-10 20:3:13.33: =====================================================
2020-4-10 20:3:13.34: =========== Testing session finished =================
2020-4-10 20:3:13.35: =====================================================
2020-4-10 20:3:13.38: Finished.
```

**Listing 6** Snippet inside the testing phase when incoming request is processed

## 0.2 Client-side

Our application is developed for Android Platform and it is developed in Android Studio. To run the application, one must allow installation from unknown resources and enable it manually from the settings. It will run successfully on any Android version greater than 5.0

### 0.2.1 DICOM Reader

```
public View getView(int i, View view, ViewGroup viewGroup) {
            if (view == null) {
                view = View.inflate(this.mContext, C0354R.layout.
    dicom_header_list_item, null);
            }
            DicomHeader dicomHeader = (DicomHeader) getItem(i);
            if (dicomHeader != null) {
                TextView textView = (TextView) view.findViewById(C0354R.
    C0356id.tag);
                String str = "";
                if (dicomHeader.mTag != null) {
                    textView.setText(new String(dicomHeader.mTag));
                } else {
                    textView.setText(str);
                }
                TextView textView2 = (TextView) view.findViewById(C0354R.
    C0356id.value);
                if (dicomHeader.mType == 0) {
                    if (dicomHeader.mStringValue != null) {
                        textView2.setText(Constants.
    getStringFromByteWithLocale(dicomHeader.mStringValue));
                    } else {
                        textView2.setText(str);
                    }
                } else if (dicomHeader.mType == 1) {
                    if (dicomHeader.mLongValue > 19000000) {
                        Calendar instance = Calendar.getInstance();
                        int i2 = (int) (dicomHeader.mLongValue / 10000);
                        instance.set(i2, (int) (((dicomHeader.mLongValue - ((
    long) (i2 * 10000))) / 100) - 1), (int) (dicomHeader.mLongValue % 100));
                        textView2.setText(DateFormat.getDateFormat(this.
    mContext).format(instance.getTime()));
                    } else {
                        textView2.setText(str);
                    }
                }
            }
            return view;
        }
```

**Listing 7** Snipper of the DICOM Reader

### 0.2.2 Directory Selector

Once the application is launched, File selector enables the user to select the local DICOM medical file.

```
!Loads directory from phone’s Internal Storage or SD card
private void createItem(final Context context) {
            this.mItems.add(new ItemPack(C0354R.string.landing_load_internal,
   C0354R.C0355drawable.ic_internal, new Runnable() {
                public void run() {
                    GAHelper.sendTracker(context, LandingActivity.TAG, "Click"
   , "Internal storage", 0);
                    ((Activity) ItemAdapter.this.mContext).
   startActivityForResult(MainActivity.getFilePickerIntent(ItemAdapter.this.
   mContext, 1, false, null, null), 1);
                }
            }));
            final String sDCardPath = StorageUtil.getSDCardPath(this.mContext)
   ;
            if (!TextUtils.isEmpty(sDCardPath)) {
                File file = new File(sDCardPath);
                if (file.exists() && file.isDirectory() && file.canRead()) {
                    this.mItems.add(new ItemPack(C0354R.string.landing_load_sd
   , C0354R.C0355drawable.ic_sd, new Runnable() {
                        public void run() {
                            GAHelper.sendTracker(context, LandingActivity.TAG,
    "Click", "External", 0);
                            ((Activity) ItemAdapter.this.mContext).
   startActivityForResult(MainActivity.getFilePickerIntent(ItemAdapter.this.
   mContext, 1, false, null, sDCardPath), 1);
                        }
                    }));
                }
            }
!Picks file from the directory chosen
 private static class CheckBoxAdapter extends BaseAdapter implements
   OnItemClickListener {
        private FilePicker mFilePicker;
        private ArrayList<SingleItem> mItems = new ArrayList<>();
        private int mSamllIconsize = 0;

        private void loadFileFromDir(File file, ArrayList<String> arrayList) {
        if (file != null && file.isDirectory()) {
            File[] listFiles = file.listFiles();
            if (listFiles != null && listFiles.length > 0) {
                for (int i = 0; i < listFiles.length; i++) {
                    if (listFiles[i] != null && listFiles[i].canRead() &&
   listFiles[i].isFile()) {
                        arrayList.add(listFiles[i].getAbsolutePath());
                    } else if (listFiles[i] != null && listFiles[i].canRead()
   && listFiles[i].isDirectory()) {
                        loadFileFromDir(listFiles[i], arrayList);
                    }
                }
            }
        }
    }

        public String[] getSelectedFilePathes() {
            if (this.mItems.size() > 0) {
                ArrayList arrayList = new ArrayList();
                Iterator it = this.mItems.iterator();
                while (it.hasNext()) {
                    SingleItem singleItem = (SingleItem) it.next();
                    if (singleItem.mIsChecked) {
                        arrayList.add(singleItem.mFile.getAbsolutePath());
                    }
                }
                if (arrayList.size() > 0) {
                    return (String[]) arrayList.toArray(new String[arrayList.
size()]);
                }
            }
            return null;
        }

        public void onItemClick(AdapterView<?> adapterView, View view, int i,
long j) {
            SingleItem singleItem = (SingleItem) getItem(i);
            if (singleItem == null) {
                return;
            }
            if (singleItem.mFile.isDirectory()) {
                this.mFilePicker.goToDir(singleItem.mFile.getAbsolutePath());
            } else if (view != null) {
                ((CheckBox) view.findViewById(C0354R.C0356id.checkbox)).
performClick();
            }
        }
    }
```

**Listing 8** Snippet of the Directory selectory. It allows user to select DICOM data automatically

### *0.2.3 Application Init*

```
public class DroidRenderApplication extends Application {
    private static AnalysisUtils sTracker;
    !Loads the DICOMreader from Android ImageViewer library
    static {
        System.loadLibrary("PGDicomReader");
    }

    public static synchronized AnalysisUtils getDefaultTracker(Context context
) {
        AnalysisUtils analysisUtils;
        synchronized (DroidRenderApplication.class) {
            if (sTracker == null) {
                sTracker = new AnalysisUtils(context);
            }
            analysisUtils = sTracker;
        }
        return analysisUtils;
    }
```

```
 public void onCreate() {
     super.onCreate();
     Constants.setSystemString();
     StringBuilder sb = new StringBuilder();
     sb.append(Constants.getAppDataPath(this));
     sb.append(Constants.RECENT_FILE_FOLDER_PATH);
     File file = new File(sb.toString());
     if (!file.exists()) {
         file.mkdirs();
     }
     MainActivity.function(83, Integer.parseInt(getSharedPreferences(
Constants.PREF_NAME_SETTINGS, 0).getString(getString(C0354R.string.
pref_key_3d_default_type), getString(C0354R.string.
pref_key_3d_default_type_default_value))));
     MainActivity.initJNI(null, getResources().getDimension(C0354R.dimen.
onedp), this);
     MainActivity.functions(84, Constants.getAppDataPath(this));
     MainActivity.functions(85, Constants.getFolderPath(this, FileType.
cloud));
     createDeafultLayout();
 }

 !Creates a default layout on the main screen
 private void createDeafultLayout() {
     SharedPreferences sharedPreferences = getSharedPreferences(Constants.
PREF_NAME_LAYOUTS, 0);
     String str = Constants.PREF_NAME_SET_DEFAULT_LAYOUTS;
     if (!sharedPreferences.getBoolean(str, false)) {
         Editor edit = sharedPreferences.edit();
         int i = 0;
         while (true) {
             int i2 = 1;
             if (i < 4) {
                 int layoutPrefValue = Constants.getLayoutPrefValue(
sharedPreferences, PrefKeyOrder.num, i, 0);
                 if (layoutPrefValue == 0) {
                     edit.putInt(Constants.getLayoutPrefKey(PrefKeyOrder.
num, i, 0), 1);
                     edit.putInt(Constants.getLayoutPrefKey(PrefKeyOrder.
orientation, i, 0), 0);
                     edit.putInt(Constants.getLayoutPrefKey(PrefKeyOrder.
left, i, layoutPrefValue), 0);
                     edit.putInt(Constants.getLayoutPrefKey(PrefKeyOrder.
top, i, layoutPrefValue), 0);
                     edit.putInt(Constants.getLayoutPrefKey(PrefKeyOrder.
width, i, layoutPrefValue), 100);
                     edit.putInt(Constants.getLayoutPrefKey(PrefKeyOrder.
height, i, layoutPrefValue), 100);
                     if (i == 0) {
                         edit.putInt(Constants.getLayoutPrefKey(
PrefKeyOrder.type, i, layoutPrefValue), 1);
                     } else {
                         int viewTypeByOrder = Entity.getViewTypeByOrder(i
    + 1);
                            if (viewTypeByOrder >= 0) {
                                i2 = viewTypeByOrder;
                            }
                            edit.putInt(Constants.getLayoutPrefKey(
    PrefKeyOrder.type, i, layoutPrefValue), i2);
                        }
                    }
                    i++;
                } else {
                    edit.putBoolean(str, true);
                    edit.apply();
                    return;
                }
            }
        }
    }
}
```

**Listing 9** Snippet of the Main file which is triggered as the application is opened

### *0.2.4 Orientation Change on Touch*

```
private class PreviewTouchListener implements OnTouchListener {
        private Point mStartPoint;

        private PreviewTouchListener() {
            this.mStartPoint = new Point();
        }

        public boolean onTouch(View view, MotionEvent motionEvent) {
            if (motionEvent.getAction() == 0 && motionEvent.getPointerCount()
    <= 1) {
                this.mStartPoint.x = (int) motionEvent.getX();
                this.mStartPoint.y = (int) motionEvent.getY();
            } else if (motionEvent.getAction() == 2) {
                PatientPreview access$1500 = ManagePatientActivity.this.
    mPatientListAdapter.getcurrentHLPatient();
                if (access$1500 != null) {
                    int x = (int) (motionEvent.getX() - ((float) this.
    mStartPoint.x));
                    int y = (int) (motionEvent.getY() - ((float) this.
    mStartPoint.y));
                    int i = access$1500.mUValue;
                    int i2 = access$1500.mDValue;
                    access$1500.mUValue += x;
                    access$1500.mDValue += x;
                    access$1500.mUValue += y;
                    access$1500.mDValue -= y;
                    if (access$1500.mUValue < access$1500.mDValue + 6) {
                        access$1500.mUValue = access$1500.mDValue + 6;
                    }
                    this.mStartPoint.x = (int) motionEvent.getX();
                    this.mStartPoint.y = (int) motionEvent.getY();
                    if (!(i == access$1500.mUValue && i2 == access$1500.
mDValue)) {
                        ManagePatientActivity.this.updateContrast(access$1500,
 access$1500.mDValue, access$1500.mUValue, true);
                    }
                }
            }
            return true;
        }
    }
```

**Listing 10** User's touch and gesture changes orientation of the 3D model

### *0.2.5 Transparency Selection*

Underlay the snippets of some of the features embedded in the application.

```
    !Checks the quality optimal for the screen resolution
    private void checkQualityPref() {
        set3DQuality(Integer.parseInt(getSharedPreferences(Constants.
PREF_NAME_SETTINGS, 0).getString(Constants.PREF_3D_QUALITY, "1")));
    }

    !Enables transparency change and slicing
    public void onVisibilityChanged(View view, int i) {
        if (i == 8 || i == 4) {
            BitmapPack bitmapPack = this.mPack;
            if (!(bitmapPack == null || bitmapPack.mBaseBitmap == null)) {
                if (Constants.sDebug) {
                    StringBuilder sb = new StringBuilder();
                    sb.append("Recycle mBaseBitmap because this view will
become invisible , hash : ");
                    sb.append(hashCode());
                    Log.i(TAG, sb.toString());
                }
                this.mPack.mBaseBitmap.recycle();
                this.mPack.mBaseBitmap = null;
            }
            Bitmap bitmap = this.mBackground;
            if (bitmap != null) {
                bitmap.recycle();
                this.mBackground = null;
            }
        } else if (i == 0) {
            BitmapPack bitmapPack2 = this.mPack;
            if (bitmapPack2 != null) {
                redraw(bitmapPack2);
                invalidate();
            }
        }
        super.onVisibilityChanged(view, i);
    }
```

**Listing 11** Snippet of the features

### 0.2.6 *Volume Rendering*

```
private void setVRThreshold() {
        if (!this.goJNI || getValueInt(3) != 0) {
            final MyDialog myDialog = new MyDialog(this);
            View inflate = ((LayoutInflater) getSystemService("layout_inflater
   ")).inflate(C0354R.layout.layout_vr_adjust, (ViewGroup) findViewById(
   C0354R.C0356id.root_view));
            final TextView textView = (TextView) inflate.findViewById(C0354R.
   C0356id.text);
            final SeekBar seekBar = (SeekBar) inflate.findViewById(C0354R.
   C0356id.seekbar);
            final int valueInt = getValueInt(1);
            int valueInt2 = getValueInt(2);
            int valueInt3 = getValueInt(4);
            seekBar.setMax(valueInt2 - valueInt);
            C031213 r5 = new OnSeekBarChangeListener() {
                public void onStartTrackingTouch(SeekBar seekBar) {
                }

                public void onStopTrackingTouch(SeekBar seekBar) {
                }

                public void onProgressChanged(SeekBar seekBar, int i, boolean
   z) {
                    textView.setText(String.format("%d", new Object[]{Integer.
   valueOf(i + valueInt)}));
                }
            };
            myDialog.setButton(-1, "Apply change", new OnClickListener() {
                public void onClick(DialogInterface dialogInterface, int i) {
                    if (MainActivity.this.goJNI) {
                        MainActivity.function(71, seekBar.getProgress() +
   valueInt);
                    }
                    myDialog.dismiss();
                }
            });
            myDialog.setButton(-2, "Cancel", new OnClickListener() {
                public void onClick(DialogInterface dialogInterface, int i) {
                    dialogInterface.dismiss();
                }
            });
            seekBar.setOnSeekBarChangeListener(r5);
            seekBar.setProgress(valueInt3 - valueInt);
            myDialog.setTitle(C0354R.string.vr_threshold);
            myDialog.setView(inflate);
            myDialog.show();
        }
    }

    private void setVRType() {
        Builder builder = new Builder(this);
        builder.setTitle(C0354R.string.vr_type);
        builder.setItems(new CharSequence[]{"Basic", "Volume shading", "Volume
    rendering"}, new OnClickListener() {
            public void onClick(DialogInterface dialogInterface, int i) {
                int i2 = 2;
                if (i != 1) {
                    i2 = i != 2 ? 0 : 3;
                }
                if (MainActivity.this.goJNI) {
                    MainActivity.function(70, i2);
                }
                dialogInterface.dismiss();
            }
        });
        builder.create().show();
    }
```

**Listing 12** Snippet of initialization of Volume Rendering

### *0.2.7 Canvas*

```
private void drawEnlargePoints(Canvas canvas, BitmapPack bitmapPack, Bitmap
    bitmap) {
        Canvas canvas2 = canvas;
        BitmapPack bitmapPack2 = bitmapPack;
        if (bitmapPack2.mEnlargePoints != null && bitmapPack2.mEnlargePoints.
    size() > 0) {
            ArrayList<Point> arrayList = bitmapPack2.mEnlargePoints;
            Point point = new Point();
            int i = (int) (this.mOneDP * 35.0f);
            Iterator it = arrayList.iterator();
            while (it.hasNext()) {
                Point point2 = (Point) it.next();
                int i2 = i * 2;
                double d = (double) point2.f12y;
                double d2 = (double) i;
                Double.isNaN(d2);
                if (d < d2 * 2.5d) {
                    i2 = -i2;
                }
                this.mEnlargeMatrix.set(this.mMatrix);
                this.mEnlargeMatrix.preScale(FACTOR, FACTOR);
                if (bitmapPack2.mType == 0) {
                    this.mEnlargeMatrix.postTranslate(((float) (bitmapPack2.
    mXshift2D - point2.f11x)) * 1.2f, (((float) (bitmapPack2.mYshift2D -
    point2.f12y)) * 1.2f) - ((float) i2));
                } else {
                    this.mEnlargeMatrix.postTranslate(((float) point2.f11x) *
    -1.2f, (((float) point2.f12y) * -1.2f) - ((float) i2));
                }
                this.mPath.reset();
                point.f11x = point2.f11x;
                point.f12y = point2.f12y - i2;
                this.mPath.addCircle((float) point.f11x, (float) point.f12y, (
    float) i, Direction.CW);
                canvas.save();
                canvas2.clipPath(this.mPath);
                canvas2.drawARGB(128, 0, 0, 0);
                canvas2.drawBitmap(bitmap, this.mEnlargeMatrix, null);
                canvas.restore();
                this.mSchedulePaint.setStrokeWidth(4.0f);
                this.mSchedulePaint.setARGB(180, 0, 0, 0);
                canvas.drawLine((((float) point.f11x) - (this.mOneDP *
    CENTER_GAP)) + 1.0f, (float) point.f12y, (float) (point.f11x - i), (float)
     point.f12y, this.mSchedulePaint);
                canvas.drawLine((((float) point.f11x) + (this.mOneDP *
    CENTER_GAP)) - 1.0f, (float) point.f12y, (float) (point.f11x + i), (float)
     point.f12y, this.mSchedulePaint);
                canvas.drawLine((float) point.f11x, (((float) point.f12y) - (
    this.mOneDP * CENTER_GAP)) + 1.0f, (float) point.f11x, (float) (point.f12y
     - i), this.mSchedulePaint);
                canvas.drawLine((float) point.f11x, (((float) point.f12y) + (
    this.mOneDP * CENTER_GAP)) - 1.0f, (float) point.f11x, (float) (point.f12y
     + i), this.mSchedulePaint);
                this.mSchedulePaint.setStrokeWidth(2.0f);
                this.mSchedulePaint.setARGB(255, 220, 220, 220);
                canvas.drawLine(((float) point.f11x) - (this.mOneDP *
    CENTER_GAP), (float) point.f12y, (float) (point.f11x - i), (float) point.
    f12y, this.mSchedulePaint);
                Canvas canvas3 = canvas;
                canvas3.drawLine((this.mOneDP * CENTER_GAP) + ((float) point.
    f11x), (float) point.f12y, (float) (point.f11x + i), (float) point.f12y,
    this.mSchedulePaint);
                canvas.drawLine((float) point.f11x, ((float) point.f12y) - (
    this.mOneDP * CENTER_GAP), (float) point.f11x, (float) (point.f12y - i),
    this.mSchedulePaint);
                Canvas canvas4 = canvas;
                canvas4.drawLine((float) point.f11x, (this.mOneDP * CENTER_GAP
    ) + ((float) point.f12y), (float) point.f11x, (float) (point.f12y + i),
    this.mSchedulePaint);
            }
            arrayList.clear();
        }
     }
```

**Listing 13** Snipper of the draw functions that supports all the 3D manipulation on app's screen during runtime by the user

### *0.2.8 Update Contrast*

```
public void updateContrast(PatientPreview patientPreview, int i, int i2,
    boolean z) {
        if (this.mPreviewSlice != null) {
            patientPreview.mDValue = i;
            patientPreview.mUValue = i2;
            int i3 = i2 - i;
            float f = 255.0f / ((float) i3);
            for (int i4 = 0; i4 < this.mPreviewSlice.mWidth; i4++) {
                for (int i5 = 0; i5 < this.mPreviewSlice.mHeight; i5++) {
                    if ((this.mPreviewSlice.mWidth * i5) + i4 < this.
    mPreviewBits.length) {
```

```
                     int i6 = (int) (((float) (this.mPreviewSlice.mBits[(
    this.mPreviewSlice.mWidth * i5) + i4] - i)) * f);
                     if (i6 > 255) {
                         i6 = 255;
                     } else if (i6 < 0) {
                         i6 = 0;
                     }
                     this.mPreviewBits[(this.mPreviewSlice.mWidth * i5) +
    i4] = i6 | -16777216 | (i6 << 16) | (i6 << 8);
                 }
             }
         }
         this.mPreviewbitmap.setPixels(this.mPreviewBits, 0, this.
    mPreviewSlice.mWidth, 0, 0, this.mPreviewSlice.mWidth, this.mPreviewSlice.
    mHeight);
         this.mPreview.setImageBitmap(this.mPreviewbitmap);
         if (z) {
             Object[] objArr = {Integer.valueOf((i2 + i) / 2)};
             String str = "%d";
             this.mContrastToast.setText(getString(C0354R.string.
    contrast_toast, new Object[]{String.format(str, objArr), String.format(str
    , new Object[]{Integer.valueOf(i3)})}));
             this.mContrastToast.show();
         }
     }
 }
```

**Listing 14** Snippet of the code that changes the contrast of the view upon user's sliding gesture

### *0.2.9 User-drawn metrics and UI Update*

```
public void updateUI(final PatientPreview patientPreview, PreviewSlice
    previewSlice, boolean z) {
        this.mPreviewSlice = previewSlice;
        HeaderAdapter headerAdapter = this.mHeaderAdapter;
        PreviewSlice previewSlice2 = this.mPreviewSlice;
        headerAdapter.update(previewSlice2 == null ? null : previewSlice2.
    mDicomHeader);
        if (previewSlice == null) {
            this.mPreview.setImageDrawable(null);
            this.mSeekText.setVisibility(0);
            this.mSeekText.setText(C0354R.string.patient_manager_error_no_file
    );
            return;
        }
        if (patientPreview.mDValue == patientPreview.mUValue && patientPreview
    .mDValue == 0) {
            patientPreview.mDValue = previewSlice.mDValue;
            patientPreview.mUValue = previewSlice.mUValue;
        }
        if (patientPreview.mSliceNumber <= 1) {
            this.mSeekBar.setVisibility(8);
            this.mSeekText.setText("");
        } else {
            this.mSeekBar.setVisibility(0);
```

```
          if (!z) {
              this.mSeekBar.setMax(patientPreview.mSliceNumber - 1);
              this.mSeekBar.setProgress(patientPreview.mCurrentSlice);
              this.mSeekBar.setOnSeekBarChangeListener(new
 OnSeekBarChangeListener() {
                  public void onStartTrackingTouch(SeekBar seekBar) {
                  }

                  public void onStopTrackingTouch(SeekBar seekBar) {
                  }

                  public void onProgressChanged(SeekBar seekBar, int i,
 boolean z) {
                      if (z) {
                          PatientPreview patientPreview = patientPreview;
                          patientPreview.mCurrentSlice = i;
                          ManagePatientActivity managePatientActivity =
 ManagePatientActivity.this;
                          managePatientActivity.updateUI(patientPreview,
 MainActivity.getPreviewSlice((managePatientActivity.mUsedFor == 1 ||
 ManagePatientActivity.this.mUsedFor == 2) ? false : true,
 ManagePatientActivity.this.mPatientListAdapter.mSelectedItem,
 patientPreview.mCurrentSlice), true);
                      }
                  }
              });
          }
          this.mSeekText.setText(String.format("%d / %d", new Object[]{
 Integer.valueOf(patientPreview.mCurrentSlice + 1), Integer.valueOf(
 patientPreview.mSliceNumber)}));
      }
      if (!z) {
          this.mPreviewBits = new int[(previewSlice.mWidth * previewSlice.
 mHeight)];
          this.mPreviewbitmap = Bitmap.createBitmap(this.mPreviewBits,
 previewSlice.mWidth, previewSlice.mHeight, Config.ARGB_8888).copy(Config.
 ARGB_8888, true);
      }
      updateContrast(patientPreview, patientPreview.mDValue, patientPreview.
 mUValue, false);
  }
```

**Listing 15** Snippet of the code that allows user-operated manipulations such as measurements and orientation to be reflected on the screen