RenderNet and AWS cloud service for Mobile volume rendering

A V Krishnarao Padyala [1], Dr. Ajay Kaushik[2]

[1]SRM University, Sonepat, Haryana, Delhi-NCR, India
guntur.krishna@gmail.com
[2] Department of CSE, SRM University, Sonepat, Haryana, Delhi-NCR, India
ajay.kaushik@srmuniversity.ac.in

Abstract. RenderNet is a differentiable rendering convolutional network with a novel projection unit that can render 2D images from 3D shapes. Spatial occlusion and shading calculation are automatically encoded in the network. Our experiments show that RenderNet can successfully learn to implement different shaders, and can be used in inverse rendering tasks to estimate shape, pose, lighting, and texture from a single image. This paper also covers storing large volumes of healthcare data in the cloud. DICOM web service of AWS provides a variety of services for storage management.

Keywords: Volume rendering, mobile volume rendering, ray casting, rendernet, deep learning, AWS cloud service.

1 Introduction

Several volume rendering methods were developed, but most of them were desktop-based applications. very few applications among them were developed for ios mobile. No proper working application was developed for Android mobile till now. As most people nowadays use Android mobile, there is a necessity for an application for sharing patient-related information.

This paper mainly focuses on research work carried out in mobile applications for volume rendering. This highlights the effective and interactive volume visualization developed using Python packages and also storage management with AWS cloud service.

Python language is having a rich library for volume visualization. Machine learning-based volume rendering was proposed in this paper. A convolutional neural network called RenderNet implementation was presented in this paper.

As the medical images were large in size, managing them is very difficult. Store, Retrieving, and removal of medical images from the cloud as per the demand can be done with the help services available in AWS.

Existing methods for mobile volume rendering haven't used cloud service for data management. Using the cloud for storage management definitely gives good results.

Interactive volume visualization and cloud storage for mobile volume rendering applications were presented in section 3.

2 Literature Survey

This section presents techniques useful for volume rendering/visualization. Volume visualization [1] is a method of extracting meaningful information from volumetric data using interactive graphics and imaging. Volume data are 3D (possibly time-varying) entities that may have information inside them. They are obtained by sampling, simulation, or modeling techniques. For example, a sequence of 2D slices obtained from Magnetic Resonance Imaging (MRI), Computed Tomography (CT), functional MRI (fMRI), or Positron Emission Tomography (PET), is 3D reconstructed into a volume model and visualized for diagnostic purposes or the planning of treatment or surgery.

Volume rendering or direct volume rendering is the process of creating a 2D image directly from 3D volumetric data, hence it is often called direct volume rendering. Volume rendering can be achieved using an object order, an image order, or a domain-based technique.

Hybrid techniques were also proposed, they include shear-warp and Raycasting techniques.

2.1 Shear-warp algorithm

In shear-warp[2], the volume is rendered by a simultaneous traversal of RLE-encoded voxel and pixel runs, where opaque pixels and transparent voxels are efficiently skipped during these traversals. Further speed comes from the fact that sampling only occurs in the volume slices via bilinear interpolation and that the ray grid resolution matches that of the

volume slices and therefore the same bilinear weights can be used for all rays within a slice.

The caveat is that the image must first be rendered from a sheared volume onto a so-called base plane aligned with the volume slice most parallel to the true image plane. After completing the base-plane rendering, the base-plane image must be warped onto the true image plane, and the resulting image is displayed.

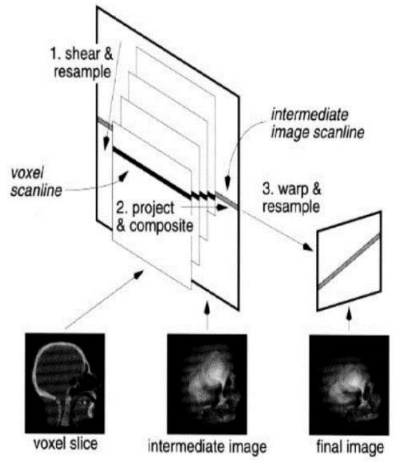

Fig. 1. Shear-warp for volume rendering

2.2 Raycasting

An alternative technique to visualize irregular grids is Raycasting[3]. Ray-casting methods tend to be more exact than projective techniques since they are able to “stab” or integrate the cells in depth order, even in the presence of cycles. This is generally not possible in cell-by-cell projection methods.

Many ray-casting approaches employ the plane sweep paradigm, which is based on processing geometric entities in an order determined by passing a line or a plane over the data. It was pioneered by Giertsen for its use in volume rendering. It is based on a sweep plane that is orthogonal to the viewing plane (e.g., orthogonal to the y-axis).

Events in the sweep are determined by vertices in the dataset and by values of y that correspond to the pixel rows. When the sweep plane passes over a vertex, an "Active Cell List" (ACL) is updated to store the set of cells intersected by the sweep plane. When the sweep plane reaches a y-value that defines the next row of pixels, the current ACL is used to process that row, casting rays, corresponding to the values of x that determine the pixels in the row, through a regular grid (hash table) that stores the elements of the ACL.

In its basic form, the volume ray casting algorithm comprises four steps:

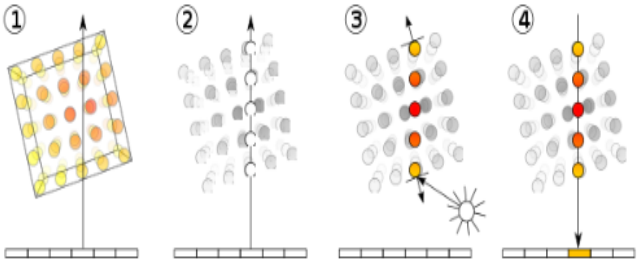

Fig. 2. Steps in Raycasting for volume rendering

Raycasting

For each pixel of the final image, a ray of sight is shot ("cast") through the volume. At this stage, it is useful to consider the volume being touched and enclosed within a bounding primitive, a simple geometric object — usually a cuboid — that is used to intersect the ray of sight and the volume.

Sampling

Along the part of the ray of sight that lies within the volume, equidistant sampling points or samples are selected. In general, the volume is not aligned with the ray of sight and sampling points will usually be located in between voxels. Because of that, it is necessary to interpolate the values of the samples from their surrounding voxels (commonly using trilinear interpolation).

Shading

For each sampling point, a transfer function retrieves an RGBA material color and a gradient of illumination values is

computed. The gradient represents the orientation of local surfaces within the volume. The samples are then shaded (i.e. colored and lit) according to their surface orientation and the location of the light source in the scene.

Compositing

After all sampling points have been shaded, they are composited along the ray of sight, resulting in the final color value for the pixel that is currently being processed. The composition is derived directly from the rendering equation and is similar to blending acetate sheets on an overhead projector. It may work back-to-front, i.e. computation starts with the sample farthest from the viewer and ends with the one nearest to the viewer. This workflow direction ensures that masked parts of the volume do not affect the resulting pixel. The front-to-back order could be more computationally efficient since the residual ray energy is getting down while the ray travels away from the camera; so, the contribution to the rendering integral is diminishing therefore more aggressive speed/quality compromise may be applied (increasing of distances between samples along ray is one of such speed/quality trade-offs).

2.3 Transfer function in volume rendering

A Transfer function[4] may be used for rendering medical images. This maps volumetric data to optical properties. The application of the transfer function to volumetric data is a part of the traditional visualization pipeline. This consists of data acquisition, processing, visual mapping, and rendering.

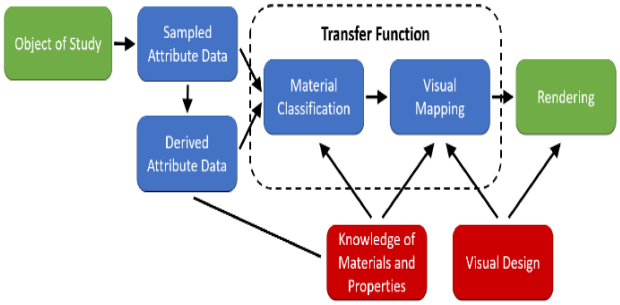

Fig. 3. Volume rendering with transfer function

The transfer function plays a major role in rendering the object. This transfer function will process the volumetric data and renders the object on the screen.

The rendering process generally classifies the materials and displays them. These classified parts were rendered in multiple colors for easy understanding. The next section describes mobile volume rendering system architecture, RenderNet used and cloud service for storage management.

3 Methodology

This section presents mainly the architecture of mobile volume rendering applications, usage of RenderNet implementation and cloud service for medical image data management.

The mobile volume rendering system is having mainly the following components.

i. A mobile device to capture and display medical images,
ii. An image processing unit,
iii. Cloud service for image management,
iv. A rendering unit.

3.1 A mobile device to capture and display medical images

A mobile device is to be equipped with special hardware to capture images. This should let the device capture images of the organ inside the human body by passing rays through it.

3.2 Healthcare Image Storage Management

Healthcare/ medical images were usually stored in DICOM (Digital Imaging and Communications in Medicine) format.

Managing medical images is difficult as they may be large in number and size. Nowadays so many cloud technologies are available for storage management. In our application, we are using cloud services for storage management.

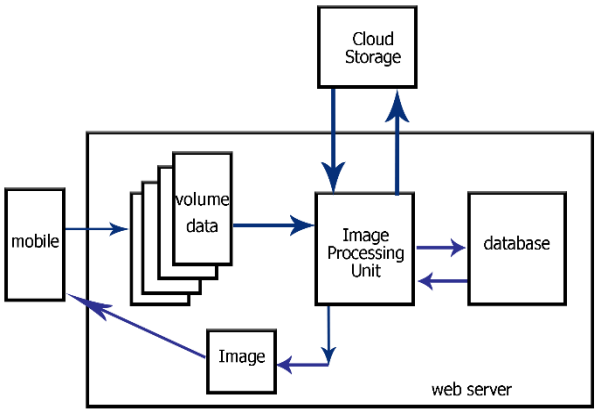

Fig. 4. Mobile volume rendering System.

3.3 Cloud Service for storing DICOM images

Cloud services are majorly available as platform as a service (PaaS), Infrastructure as a service (IaaS), and software as a service (SaaS). The most popular cloud services for health image/data management include Sync.com, pCloud, Dropbox, Icedrive, and Google Drive.

3.4 Image processing unit

This is responsible for processing and rendering the images captured. This image processing unit makes use of the convolution neural network for generating 2D image from DICOM image. The Digital Imaging and Communications in Medicine (DICOM) Standard specifies a non-proprietary data interchange protocol, digital image format, and file structure for biomedical images and image-related information.

This image processing unit uses RenderNet to render medical images. The process in Fig. 1. was modified and new techniques for effective volume rendering and a convolutional neural network called RenderNet[5]. Volume Rendering owes much of its versatility to the use of a transfer function (TF), introduced in the earliest work on Direct Volume Rendering (DVR) as a way of assigning optical properties (color and absorption) to the voxels of a volume. Designing a TF is split into classification and visual mapping steps (Fig. 5.).

The classification step derives a semantic based on original and derived voxel attributes and the visual mapping establishes a relation between visual properties (typically color and opacity). Both steps are guided by domain knowledge and an understanding of the visual parameters involved in rendering.

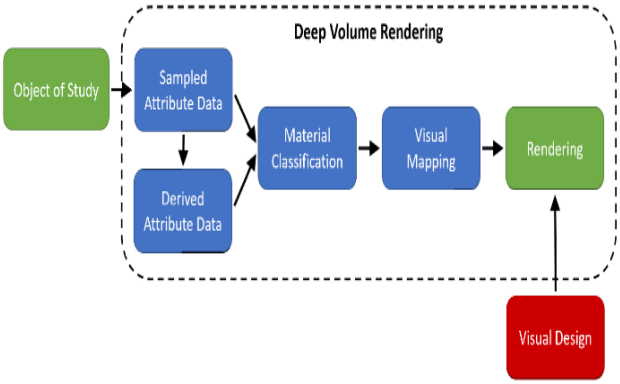

Fig. 5. Workflow in deep direct Volume rendering (Rendernet)

The differentiable nature of DVR allows the training of these models from 2D images, effectively reprojecting 2D annotations into 3D where meaningful features and visual mappings are derived implicitly during training. This process avoids the time-intensive highlighting of where interesting structures are in 3D and instead allows experts to work directly in the rendered image space to indicate what they want to see and how it should appear.

Transforming 3D images to 2D using RenderNet

Deep neural networks may be employed to effectively classify the materials. The knowledge acquired during this process is stored in the database and were used whenever necessary. Our application uses RenderNet, a convolutional neural network (CNN) architecture shown in Fig.4. that can be trained end-to-end for rendering 3D objects, including object visibility computation and pixel color calculation (shading). Voxel presentation of 3D shapes for its regularity and flexibility, and its application in visualizing volumetric data such as medical images. This focuses on voxel data. RenderNet can generate renderings of high quality, even from low-resolution and noisy voxel grids.

The viewing direction is assumed to be along the negative z-axis in the camera coordinate system. Therefore, the 3D content defined in the world coordinate system needs to be transformed into the camera coordinate system before being rendered. The two currently popular rendering methods, rasterization-based rendering and ray tracing, procedurally compute the color of each pixel in the image with two major steps: testing visibility in the scene, and computing shaded color value under an illumination model. RenderNet jointly learns both steps of the rendering process from training data, which can be generated using either rasterization or ray tracing.

RenderNet receives a voxel grid as input and applies a rigid-body transformation to convert from the world coordinate system to the camera coordinate system. The transformed input, after being trilinear sampled, is then fed to a CNN with a projection unit to produce a rendered 2D image. RenderNet consists of 3D convolutions, a projection unit that computes the visibility of objects in the scene and projects them onto 2D feature maps, followed by 2D convolutions to compute shading.

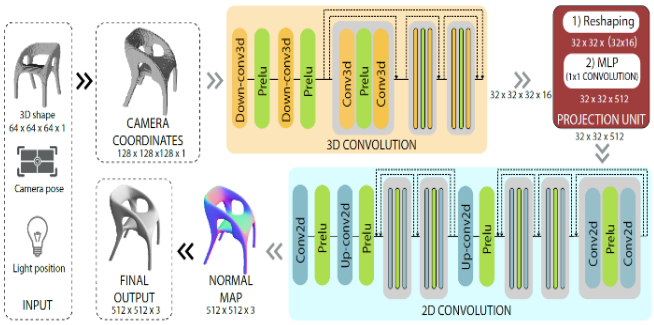

Fig. 6. Transformation of 3D image to 2D using RenderNet

The input of RenderNet is a voxel grid V of dimension $H_V*W_V*D_V*C_V$ (corresponding to height, width, depth, and channel), and the output is an image I of dimension $H_I*W_I*C_I$ (corresponding to height, width, and channel).

RenderNet is able to learn different types of shaders, including Phong shading, contour line shading, complex multi-pass shading (cartoon shading), and a ray-tracing effect (Ambient Occlusion) with the same network architecture. RenderNet was trained on datasets for each of these shaders, and the figure shows outputs generated for unseen test 3D shapes. The method also works very well when there are multiple objects in the scene. RenderNet can also handle corrupted or low-resolution volumetric data.

1D texture vector representation (these are the PCA coefficients for generating albedo texture using the BaselFace dataset) to a 3D representation of the texture that has the same

Shape reconstruction from images: RenderNet can be used for single-image reconstruction. It achieves this goal via an iterative optimization that minimizes the following reconstruction loss.

Minimize

$$z,\theta,\emptyset,\eta \quad = \; // \; I\text{-}f(z,\theta,\emptyset,\eta) \; // \qquad (1)$$

where I is the observed image and f is our pre-trained RenderNet. z is the shape to reconstruct, θ and η are the pose and lighting parameters, and Ø is the texture variable. In essence, this process maximizes the likelihood of observing the image I given the shape z.

However, directly minimizing this loss often leads to noisy, unstable results. In order to improve the reconstruction, we use a shape prior for regularizing the process, a pre-trained 3D auto-encoder similar to the TL-embedding network [5] with 80000 shapes. Instead of optimizing z, we optimize its latent representation z'.

Minimize

$$z,\theta,\emptyset,\eta \quad = \; // \; I\text{-}f(g(z'),\theta,h(\emptyset'),\eta) \; // \qquad (2)$$

where g is the decoder of the 3D auto-encoder. It regularizes the reconstructed shape g(z') by using the prior shape knowledge (weights in the decoder) for shape generation. Similarly, we use the decoder h that was trained with RenderNet

for the texture rendering task to regularize the texture variable ∅'. This corresponds to MAP estimation, where the prior term is the shape decoder and the likelihood term is given by RenderNet. Note that it is straightforward to extend this method to the multi-view reconstruction task by summing over multiple per-image losses with shared shape and appearance.

Storage

DICOM images were stored in the database temporarily while rendering. As the size of the DICOM images might be in the order of Giga Bytes/Tera Bytes, the local database may not handle it. Databases like SQLite, MySQL, Oracle, Sybase, and PostgreSQL compatible with Python were preferred in our application.

As the healthcare images that we render were large in size, using cloud services is a good solution. DICOM web service[6] of AWS cloud has shown the best performance in this kind of application. The usage of this service was presented in section 4.

4 Results and Discussions

The application developed is mobile-based and is used to render DICOM images collected from CT or MRI scans.

4.1 RenderNet Results

Datasets

We use the chair dataset from ShapeNet Core[7]. Apart from being one of the categories with the largest number of data points (6778 objects), the chair category also has a large intra-class variation. We convert the ShapeNet Dataset to 64×64×64 voxel grids using volumetric convolution[8]. We randomly sampled 120 views of each object to render training images at 512×512 resolution.

Fig. 7. Shaders generated by RenderNet

The elevation and azimuth are uniformly sampled between [10, 170] degrees and [0, 359] degrees, respectively. Camera radius are set at 3 to 6.3 units from the origin, with the object's axis-aligned bounding box normalized to 1 unit length. For the texture mapping tasks, we generate 100,000 faces from the Basel Face Dataset [9], and render them with different azimuths between [220, 320] degrees and elevations between [70, 110] degrees. We use Blender3D to generate the Ambient Occlusion (AO) dataset, and VTK for the other datasets. For the contour dataset, we implemented the pixel-based suggestive contour [10] algorithm in VTK.

Training

We adopt the patch training strategy to speed up the training process in our model. We train the network using random spatially cropped samples (along the width and height dimensions) from the training voxel grids while keeping the depth and channel dimensions intact. We only use the full-sized voxel grid input during inference. The patch size starts as small as 1/8 of the full-sized grid and progressively increases to 1/2 of the full-sized grid at the end of the training.

We train RenderNet using a pixel-space regression loss. We use mean squared error loss for colored images, and binary cross-entropy for grayscale images. We use the Adam optimizer [11], with a learning rate of 0.00001.

The image processing unit here made use of RenderNet. The results after rendering using RenderNet were shown first and then their comparison with other popular volume rendering techniques like marching cubes and Mesh render were presented in this section.

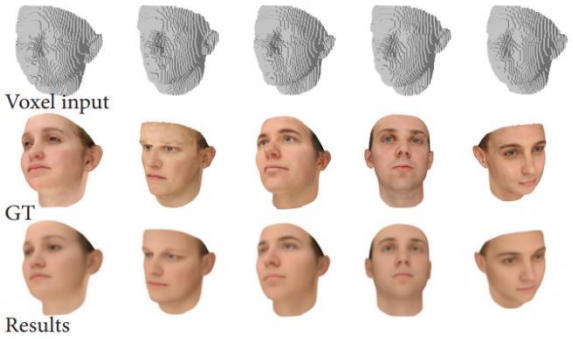

Fig. 8. Application of texture to voxel input using RenderNet

Fig. 7. Shows the Phong, Contour, cartoon, and AO shaders generated for given input images. In Figure 8. the results after the application of different textures to voxel inputs were presented. Figure 9. shows the application of different textures, the same texture to different faces, and different lighting effects for the same face applied with the same textures.

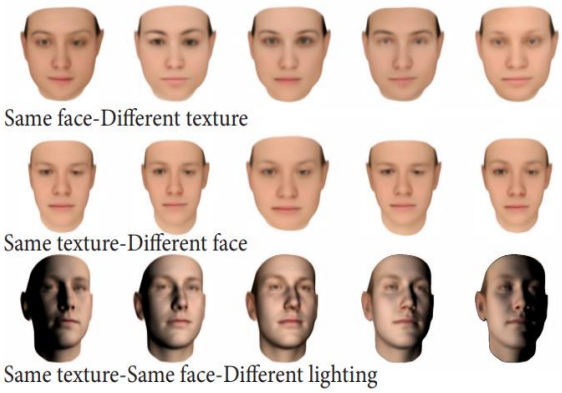

Fig. 9. Texture application and lighting effects in RenderNet

Figure 10. shows the comparison of RenderNet results with other popular volume rendering techniques. This shows results obtained from RenderNet, Mesh render, and marching cubes algorithm in rows 1, 2, and 3 respectively.

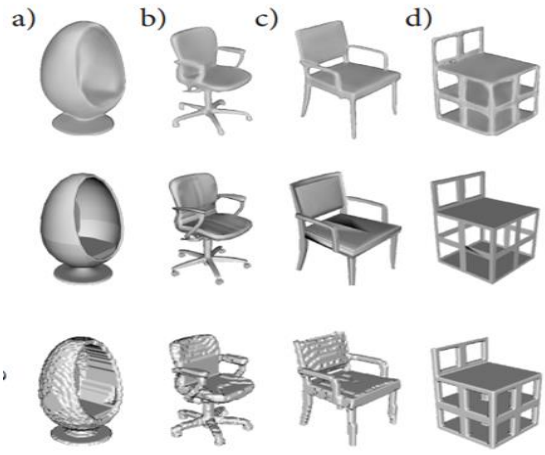

Fig. 10. comparison of RenderNet results with others

4.2. Labelling DICOM images and ML model with MONAI on Amazon

DICOM (Digital Imaging and Communications in Medicine) is an image format that contains visualizations of X-Rays and MRIs as well as any associated metadata. DICOM is the standard for medical professionals and healthcare researchers for visualizing and interpreting X-Rays and MRIs.This section addresses two issues i) Visualize and label DICOM images using a custom data labeling workflow on Amazon SageMaker[12], ii) Develop a DenseNet image classification model using the MONAI framework on Amazon SageMaker. For this experiment, we use a chest X-Ray DICOM images dataset from the MIMIC Chest X-Ray (MIMIC-CXR) Database, a publicly available database of chest X-Ray images in DICOM format and the associated radiology reports as free text files.

The following diagram shows the high-level workflow with the following key components:

i. The DICOM images are stored in a third-party picture archiving and communication system (PACS) or Vendor Neutral Archive (VNA), and retrieved through DICOMweb.
ii. An input manifest.json file is uploaded to Amazon Simple Storage Service (Amazon S3). The file contains the DICOM instance ID as the data source and potential labels used by annotators when they perform the labeling jobs.
iii. Two AWS Lambda functions are essential for creating labeling jobs on Ground Truth:
    a. A pre-labeling function reads the DICOM image ID from the manifest.json file and creates the task input object that is fed into the Liquid HTML template for automation.
    b. A post-labeling processing function consolidates the annotations from different labeling jobs.
iv. A HTML template with Crowd elements for submitting the labeling jobs and processing the output object. Subsequently, the output of labeling jobs is saved in an output label S3 bucket.
v. A SageMaker notebook can retrieve the outputs of labeling jobs and use them to train a supervised ML model.

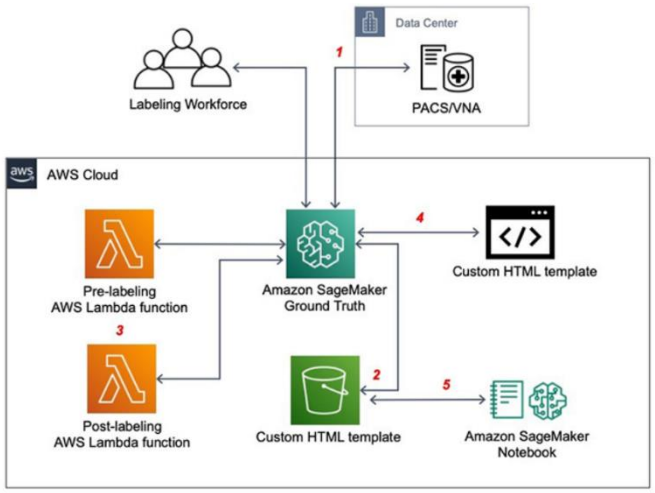

Fig. 7. DICOM labeling and classification.

Custom HTML templates can be used to classify the chest X-Ray DICOM images into one or more categories, out of the possible acute and chronic cardiopulmonary conditions. The HTML template built supports DICOM image retrieval and interactive visualization, plus several annotations.

Deploy a third-party PACS on AWS

Orthanc, a PACS is used for this experiment, in which any PACS or VNA supporting DICOMweb can be used. You can deploy the Orthanc for Docker container on AWS by the following AWS CloudFormation stack.

Fill in the required information during the deployment, including Amazon Elastic Compute Cloud (Amazon EC2) key pair to access the hosting EC2 instance and network infrastructure (VPC and subnets). An NGINX server has been added in the container to proxy the HTTPS traffic to the Orthanc server at port 8042, which also adds Access-Control-Allow-Origin headers for cross-origin resource sharing (CORS). The Orthanc container is deployed on Amazon Elastic Container Service (Amazon ECS) and connected to an Amazon Aurora PostgreSQL database. After the CloudFormation stack is successfully created, take note of the Orthanc endpoint URL on the Outputs tab.

Create the Lambda functions, S3 bucket, and SageMaker notebook instance

For the parameter PreLabelLambdaSourceEndpointURL, enter the Orthanc endpoint URL from the previous step,

which the pre-labeling task Lambda function uses to generate WADO-URI for a given DICOM instance ID. We recommend creating a notebook instance type of ml.m5.xlarge to carry out the ML modeling after image annotations.

After the stack deployment, take note of the outputs, including the SMGTLabelingExecutionRole and SageMakerAnnotationS3Bucket values. The IAM role SMGTLabelingExecutionRole is used to create the Ground Truth labeling job.

Upload DICOM images and prepare the input manifest

Uploading the DICOM images into the Orthanc server can be done either through its web UI or the WADO-RS REST API. After the DICOM images are uploaded, retrieve the DICOM instance IDs for them, and generate a manifest file with the instance IDs. Each JSON object separated by a standard line break in the manifest file represents an input task sent to the workforce for labeling. The data object in this case contains the instance ID and metadata for the potential labels, which are the possible diseases indicated by the image. After the manifest.json file is compiled, upload it to the S3 bucket created.

Build a custom data labeling job

Use the Cornerstone JavaScript library to build the labeling UI that displays a DICOM image in modern web browsers that support the HTML5 canvas element, and use Cornerstone tools to support interactions and enable region of interest (ROI) annotations in different shapes. Additionally, use the Cornerstone WADO Image Loader to retrieve Web Access to DICOM Objects (WADO) URI through the DICOMwebTM plugin in a remote Orthanc server.

The following two functions are important for retrieving and viewing the DICOM image:

i. DownloadAndView() – Adds wadouri: to the beginning of the URL so that the cornerstone.js file can find the image loader

ii. loadAndViewImage() – Uses the Cornerstone library to display the image and allows additional functionalities such as zoom, pan, and ROI annotation

One should be able to annotate any DICOM image using the custom HTML template, which lists the possible labels as multiple option selections for the image classification task.

4.3. Build an ML model using the MONAI framework

Create a new notebook instance. The MONAI library and other additional packages are managed in requirements.txt, which is in the same folder as the training script. Load the file that has the DICOM URLs and labels. Use a Python file to parse the labeled dataset.

Now that we have the DICOM URLs in image_url_list and corresponding labels in image_label_list, we can download the DICOM files from the Orthanc DICOM server directly into Amazon S3 for SageMaker training.

After you load the images to your S3 bucket, you can display a sample of the images with labels from the parsed JSON file using the MONAI framework.

We use the MONAI transforms during model training to load the DICOM files directly and preprocess them. MONAI has transforms that support both dictionary and array formats, and is specialized for the high-dimensionality of DICOM medical images. The transforms include several categories such as crop and pad, intensity, I/O, postprocessing, spatial, and utilities. In the following excerpt, the Compose class chains a series of image transforms together and returns a single tensor of the image.

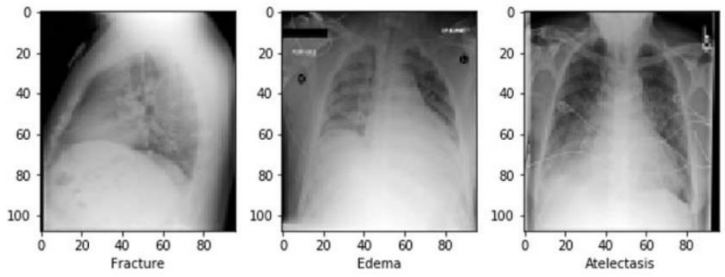

Fig. 8. DICOM images with labels.

Model training

MONAI includes deep neural networks such as UNet, DenseNet, GAN, and others, and provides sliding window inferences for large medical image volumes. In the DICOM image classification model, we train the MONAI DenseNet model on the DICOM images loaded and transformed.

5 Conclusion

In this study, we have developed a new approach for mobile volume rendering that uses RenderNet for transforming 3D images captured into 2D images. A methodology that can be used for labeling and classifying images in Azure was experimented.

Acknowledgments

All of the works carried out as part of this research work were supervised by Dr.Ajay Kaushik and supported by the Department of CSE at SRM University, Sonepat, Delhi-NCR,Haryana.

References

1. Kaufman, “Volume Visualization”, IEEE Computer Society Press Tutorial, Los Alamitos, CA.
2. P. Lacroute, and M. Levoy, “Fast volume rendering using a shear-warp factorization of the viewing transformation,” Proc. of SIGGRAPH ‘94, pp. 451-458, 1994.
3. M. Garrity, “Raytracing irregular volume data,” Computer Graphics, pp. 35-40, November 1990.
4. Patric Ljung, Jens Krüger, Eduard Groller, Markus Hadwiger, Charles D Hansen, and Anders Ynnerman. 2016. “State of the Art in Transfer Functions for Direct Volume Rendering”. Computer Graphics Forum 35, 3 (2016), 669–691. https://doi.org/10.1111/cgf.12934
5. Thu Nguyen-Phuoc, Chuan Li, Stephen Balaban, “RenderNet: A deep convolutional network for differentiable rendering from 3D shapes”. Computer Vision and Pattern Recognition, arxiv, (2019).
6. Stephen Aylward, Alex Lemm, Brianna Major, Justin Kirby, Matt McCormick, and Gang Fu, “medical image research on Amazon SageMaker Studio Lab ”,(2023).
7. Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: “An information-rich 3d model repository”. CoRR, abs/1512.03012, 2015
8. F. S. Nooruddin and G. Turk. “Simplification and repair of polygonal models using volumetric techniques”. IEEE Trans. on Vis. and Comp. Graphics, 9(2):191–205, 2003.
9. Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. “A 3d face model for pose and illumination invariant face recognition”. In Proceedings of the 2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance, pages 296–301, 2009.
10. Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. “Suggestive contours for conveying shape”. ACM TOG (Siggraph), 22(3):848–855, July 2003.
11. Diederik P. Kingma and Jimm Ba. Adam, “A method for stochastic optimization”. In International Conference on Learning Representations (ICLR), 2015.
12. Nihir Chadderwala, Bryan Marsh, and Gang Fu, “Annotate DICOM images and build an ML model using the MONAI framework on Amazon SageMaker”, AWS Machine Learning, (2021).