Tomographic Medical Image Reconstruction with Deep Learning

User Manual

Asher Burrell, Christopher Hinton, & Ty Mercer

This user manual is intended for use within Dr. Mitra's BiCLab. It will be updated when/if a public release of the code is made available.

Table of Contents

Introduction

This user manual contains instructions on how to run the Data Generation and Reconstruction AI modules of the Tomographic Medical Image Reconstruction with Deep Learning project. It is mainly intended for use within Dr. Mitra's BiCLab; However, in case the code is eventually released, the guide has been partitioned into User Manual and Developer Manual sections. User Manual includes documentation of code and processes that a user would be expected to run, and Developer Manual includes documentation of code that would only be modified by a developer.

This project is split into two modules: Data Generation and Reconstruction AI. Data Generation is focused on the creation of synthetic SPECT images via XCAT+ as well as the corresponding sinograms. It consists of programs designed to create XCAT phantoms (Requires license, which is not included with this code), turn those XCAT phantoms into XCAT+, and run GATE (requires installation, not included with this code) to obtain the corresponding sinograms. Reconstruction AI consists of a CNN designed to generate a synthetic SPECT image based on an input sinogram. It can run on real or synthetic SPECT data, although our experiments have yielded much better results with our artificial data, as this type of data was used in training.

Note that all file dimensions listed in this guide are in terms of 32-bit real values, not bytes. To find the size in bytes, multiply each dimension by 4. All user-facing programs except XCAT+ are only compatible with Linux.

User Manual

This section includes all code that a user is expected to run, as a part of this system. It is divided into Data Generation and Reconstruction AI, based on which part of the system the user is interacting with. At the end, it has information on related programs that are not a part of this system (and would not be included in a hypothetical public release), but contributed to this portion of the project and remain useful within the BiCLab.

Data Generation

These are the programs you may want to run in order to use the data generation portion of the system. The goal of the data generation system is to create large amounts of realistic synthetic SPECT data. This data was used within our project to train our Reconstruction AI, but may have other uses as well.

Folders mentioned in this section can be found under Chris/Senior_Design, in Sunflower. Corresponding folders/files are also found in the Senior_Design_1 (SD1) folder, as documented in each section. Note that the SD1 files are largely the same as (and better tested than) the files under Chris/Senior_Design, but lack some experimental quality-of-life features (currently highlighted in magenta (untested) or purple (tested)).

XCAT+

Input: XCAT phantom (or other binary image); Parameters specifying patient type; Optional parameters to change default values of organ voxels, allow for different sized input, or to turn off Gaussian blur. The input phantom can be any dimensions, but if it cannot be viewed as a coherent series of 128x128 slices the size parameter will need to be modified.

Output: Synthetic SPECT image (XCAT+ phantom). Dimensions of the output image are the same as the input.

XCAT+ is a program designed for use with XCAT (eXtended CArdiac Torso) phantoms, although it is compatible with any binary image. Its purpose is to generate synthetic SPECT images of certain organs (currently the heart and liver) from these XCAT phantoms. The code for this program, and more extensive documentation, can be found at: https://github.com/DM-BiC-Lab/XCATplus.

XCAT+ works by replacing every voxel in the input phantom that corresponds to a target organ with an appropriate tracer value. These tracer values are generated by statistical functions, which were found by analyzing a minimum of ten real patient SPECT images for each patient type: Control (non-diseased) Rest, Control Stress, Diseased Rest, or Diseased Stress. The patient type to use is specified by the user. Organs that are supported by XCAT+ at time or writing at myocardium (heart tissue), left blood pool, right blood pool, combined blood pool (uses stats from both blood pools, for phantoms where they are not differentiated), and liver. A Gaussian blur is also applied to the output image, in order to smoothen the values.

Sinogram Simulator

Input: XCAT phantom, number of sinograms to generate. XCAT phantom must conform to the default values described at https://github.com/DM-BiC-Lab/XCATplus?tab=readme-ov-file#default-values.

Output: A folder containing the following, under the Sinogram Simulator/Output: one subfolder, which contains an individual folder for each generated sinogram, as a 128x128x240 raw binary file and a small information file; The original XCAT phantom used for data generation; expected_reconstruction.raw, which is the XCAT+ phantom that was generated from the XCAT, cropped around the region of interest to a 128x128x128 binary image; The parameters file, attenuation map, or log file from XCAT creation, if they are in the same folder as the XCAT; And a log of all Sinogram Simulator runs.

SD1 Folder: XCAT_to_Sino_Pipeline

The Sinogram Simulator is a program that, given a synthetic SPECT image such as the ones generated by XCAT+, will run a physics simulation to generate raw data that corresponds to that image. This raw data is called a sinogram.

The Sinogram Simulator has XCAT+ built into it, so it should be used independently on an XCAT phantom. However, it does not allow the modification of XCAT+ parameters; Your XCAT phantom must conform to the default values listed at https://github.com/DM-BiC-Lab/XCATplus?tab=readme-ov-file#default-values. This built-in XCAT+ does not apply Gaussian blur, and will use control-rest patient statistics by default. Stress or diseased statistics will be used instead if "stress" or "diseased" is in the XCAT phantom filename.

To run the Sinogram Simulator, users must first install GATE on their computer. This is the physics simulator used to simulate the SPECT imaging process.

The Sinogram Simulator can be run by navigating to the Sinogram Simulator folder, pasting the input file, and running the command "python run_simulation.py [xcat name] [number to generate]" (do not include quotes here when typing in the command line). This input file can also be placed in Inputs. [xcat name] should be replaced with the filename of your XCAT phantom. [number to generate] should be a positive integer indicating how many sinograms to generate. Do not include a file path here. [number to generate] may be omitted, and has a default value of 1.

The Sinogram Simulator is currently only compatible with Linux. Expect this program to take a minimum of 12 hours per sinogram, depending on the processing capacity of your computer. Do not attempt to run this simulation on a laptop. Do not modify the contents of Sinogram Simulator or any of its subfolders while the simulation is running. Do not run more than one Sinogram Simulator program at a time. Running programs that require a lot of computing power while the simulator is running is not advised. Do not modify the provided contents of Sinogram Simulator unless you know what you are doing and have read the documentation under Developer Manual.

Most programs that are included in the Sinogram Simulator folder by default are dependencies of this program. They are documented in the Developer Manual.

XCAT Generator

Input: Default Organ ID Parameters file, number of XCATs to generate

Output: One XCAT phantom, and corresponding attenuation map, log, and Organ ID Params files.

SD1 File: xcat_generator.py, in XCAT (copied from this file)

This program will generate an XCAT parameters file using known parameters, then run XCAT using those parameters to create an XCAT phantom. Variation in parameters is controlled such that it resembles normal variation in human cardiac activity. Non-heart-related variables are not modified.

This program requires you to be able to run XCAT and is only compatible with Linux. XCAT is proprietary to Duke University and is not included with this code. You will need to paste the XCAT_V2_Linux folder into Sinogram Simulator; This should include your XCAT executable, license, default .nrb files, and all other XCAT dependencies. If a new version of XCAT is released, this program may become deprecated. You must also extract all other files in the Sinogram Simulator XCAT Extension folder to Sinogram Simulator.

Once these conditions are met, you can run this program with the command, "python xcat_generator.py [num_to_generate]". If [num_to_generate] is left blank, it will default to 1

Automatic Simulation

Input: Default Organ ID Params file, number of times to run the simulation on each XCAT generated. This is 1 by default.

Output: Outputs of XCAT Generator and Sinogram Simulator, repeated every time Sinogram Simulator finishes running.

SD1 File: auto_sim.py, in XCAT_to_Sino_Pipeline

In order to maintain a continuous stream of data generation, we have included a program, auto_sim.py, that will continuously create new XCATs and run the Sinogram Simulator on them. Because it is dependent on XCAT, this program is also included in Sinogram Simulator XCAT Extension.

Automatic Simulation is a continuously looping program that runs the XCAT Generator and Sinogram Simulator in sequence. It will continue to generate XCAT phantoms and run the Sinogram Simulator on them until it is manually terminated.

All requirements and restrictions for XCAT Generator and Sinogram Simulator also apply to this program.

This program can be run with the command "python auto_sim.py [number to generate]". [number to generate] can be omitted (in which case it will default to 1), or replaced with the number of sinograms to generate from each XCAT. Running this program with commands such as nohup that prevent termination via Ctrl-C is not recommended, as this will require you to stop the program via the task manager.

Sinogram Augmenter

Input: Outputs of Sinogram Simulator: Input folder containing One or more sinograms (128x128x240 raw binary) and expected reconstruction (128x128x128, raw binary); path to input folder (if it is not in Sinogram Simulator/Output). These files should be in the location they were in following the execution of the Sinogram Simulator, or else in the same format/folder layout: The input folder must contain "expected_reconstruction.raw" and exactly 1 subfolder not named "augmentation", which contains any number of subfolders, each of which contains one .bin file (the sinogram; Non-.bin files are ignored).

Output: A number of input-output pairs, equal to the number of input sinograms times 55. Inputs are sinograms (128x128x120 raw binary), outputs are synthetic SPECT images (same as input).

SD1 Folder: Augmentation

This program is used to augment the synthetic data generated by the Sinogram Simulator, so that more data is available. This data was primarily used to train the Reconstruction AI.

The Sinogram Augmenter folder should be placed in the same directory as Sinogram Simulator. It should not be placed inside Sinogram Simulator.

This program can be run with the command, "python augment.py [folder_name] [folder_path]". [folder_name] should be replaced with the name of the folder in Sinogram Simulator/Output, or the other folder with the same structure, where the input data is stored. [folder_path] should usually be omitted; If you want to access a folder that is not in Sinogram Simulator/Output, you can include the path to that folder (not including the folder itself) here. This is not recommended. Typing -all instead of [folder_name] will cause the Sinogram Augmenter to run on all folders in that directory, where the appropriate data can be found.

Output from this program will be stored under Augmented Data, in the Sinogram Augmenter folder.

This program alters the sinograms in the following ways in order to augment them:

- Resizes ("squashes") them from 128x128x240 to 128x128x120
- Rotates them at 5 angles (-10, -5, 0, 5, and 10 degrees) on the Z axis
    - Outputs 5 sinograms from the 1 input.
- Shifts them 10 times on the XY axis using a forward project → shift → back project algorithm
    - Applied to each result of rotation; Original and 10 shifted sinograms are kept per input to this step.

Sinogram Processor

Input: Outputs of Sinogram Simulator: One or more sinograms (128x128x256 raw binary) and expected reconstruction (128x128x128, raw binary); path to input folder (if it is not in Sinogram Simulator/Output). These files should be in the location they were in following the execution of the Sinogram Simulator, or else in the same format/folder layout.

Output: A number of input-output pairs, equal to the number of input sinograms. Inputs are sinograms, outputs are synthetic SPECT images.

SD1 Folder: Preprocessing

This program is similar to the Sinogram Augmenter, but it does not create additional sinograms; It only outputs modified versions of the original input. All documentation is the same as Sinogram Augmenter, with the following exceptions:

- This program is in the Sinogram Processor folder, not Sinogram Augmenter.
- This program is called preprocess.py instead of augment.py.
- Output is stored in processed_data, in the Sinogram Processor folder.
- All augmentations after "squashing" the data are omitted.

Reconstruction AI

The Reconstruction AI is a convolutional neural network (CNN) that is trained to take a sinogram as input and generate an approximation of the SPECT image that could be reconstructed from that sinogram. It is moderately successful at this task in its current iteration, but should not be relied on for medical purposes. This section only includes the code that needs to be run in order to use the AI; For documentation on AI training, loss functions, etc, see the Developer Manual.

Reconstruction AI

Input: Sinogram (128x128x120, raw binary file). May be real or synthetic data.

Output: Reconstructed SPECT image (128x128x64, raw binary). Generated by the AI, not 100% accurate.

SD1 Folder: Desktop/Senior Design/XCAT+ CEDA Pytorch

The reconstruction AI consists of a python program, xcat_ceda_pytorch.py, which can be called with the command "python ai.py [input]", where [input] is the input sinogram file. The output image will be generated in the same directory as ai.py. This program also uses a parameters file, ai_params.par, which represents the best result from our AI training; It is used to easily load the trained AI into ai.py.

For documentation on AI training, see the Developer Manual.

Lab Manual

This code is not considered to be a part of this system, and would not be included in a hypothetical release of the system. However, it is related to this project and is useful within Dr. Mitra's BiCLab, so it will be documented here.

Iterative Static Reconstruction

Input: Sinogram (folder containing series of DICOM files), Attenuation Map (single DICOM file, Attenuation-Corrected Reconstruction only).

Output: SPECT image (128x128x128 raw binary), sinograms (128x128x120 raw binary)

This is the established way to reconstruct SPECT images, using the static MLEM algorithm (see [3] for more information on static reconstruction). Iterative Reconstruction consists of the following files, in Milkyway/home/dmitra/Nuclear_Image_Simulation/Reconstruction:

- recon.py: The program to run in order to reconstruct sinograms. Modify the code at the bottom to specify one sinogram, or a path where all sinograms will be reconstructed. Then run with "python recon.py" in the terminal.
- sifads: Executable that can run static reconstruction (and dynamic reconstruction, but that is not used in this context). Called by recon.py. Current compiled version runs for 20 MLEM iterations;
- C_Code: A folder containing C code that is used to compile SIFADS. Also has the executable SIFADS_16, which can reconstruct 16-bit sinograms (the main SIFADS program is calibrated for 32 bit data, which is what we have).
- sys_mat_180_0: A raw binary file containing the system matrix for static reconstruction on our data.

Output from static reconstruction is stored in the Output folder, with raw binary sinograms (put together from the DICOM files) in the Sinograms folder. This output was reconstructed with only 10 MLEM iterations in order to be consistent with the data that was already being used for this project. If a new project is started using static reconstructed SPECT data, you may want to rerun recon.py on all the data, in order to get cleaner data, or use the attenuation corrected reconstructions.

Reconstructed images are named based on the image ID number (the number after the last period in the name of the folder holding the DICOM files) and the patient's condition (Control/Diseased, Stress/Rest). Together, these form the image ID, in the form [C/D]_[R/S]_[Number]. The number is automatically found by recon.py. If you Ctrl+F the image ID number in Data_Reference.xlsx (in the All_Data folder in

Nuclear_Image_Simulation), you can find the imaging type (Stress or rest) and information on the patient's condition (which was used to determine control vs diseased status).

We also have reconstructions with attenuation correction for each image. These reconstructions were made using the code in the Attenuation_Correction folder, and can be found in the AC_Output folder. Attenuation corrected reconstruction uses the following files, instead of the ones found in Reconstruction:

- ac_recon_no_sysmat.py: Replaces recon.py. Can be run in the same way. Has a large dictionary to identify the attenuation map for each sinogram that was successfully reconstructed without attenuation correction. "no_sysmat" means we do not generate a system matrix for each reconstruction, as we attempt to do in ac_recon.py, as that takes up a lot of space and the code to do so (sysmat_gen) doesn't seem to work.
- sifads_att: Executable file similar to SIFADS that uses the attenuation map when running static reconstruction.

Output from attenuation corrected reconstruction is very similar (not identical, but often visually indistinguishable) to normal static reconstruction at 20 iterations. This may change if we can create a custom system matrix for each sinogram, however, that project has been delayed indefinitely.

Other files/folders in the Reconstruction folder are as follows:

- Extra_Reconstructions: Additional reconstructions that we did not want to delete. Includes early experiments with attenuation correction or 20 iteration MLEM. Also has two folders, D_R_8105 and D_S_17409, that contain two different reconstructions for the same image under different patient IDs. The sinograms for these reconstructions appeared twice in the data for unknown reasons.
- recon_cropped.py: An attempt to reconstruct cropped sinograms. Results (Reconstructed.Static.Fullbody_26736_With_Cropping.Vol in Extra_Reconstructions) do not resemble SPECT data.
- test_recons.py: An early program to verify that the results of recon.py matched earlier static reconstructions. Tested reconstructions were exactly identical.

- convert_64_bit.py: Converts real data to input/output pairs for AI testing, and compresses output images to 128x128x64 raw binary to match our training data
- IO_Pairs: Input/output pairs from convert_64_bit. Attenuation corrected.

Sinogram data is sourced from the DICOM files under Nuclear_Image_Simulation/All_Data.

Segmentation Masking

Input: Reconstructed SPECT image (128x128x128 raw), manual labor

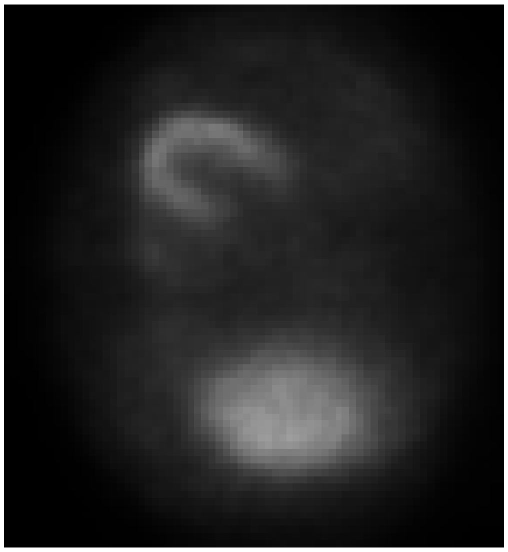
Output: Masks showing position of myocardium, left/right/combined blood pools, and liver in that image (1 128x128x128 NRRD file per mask)

Masking is a manual process that is done in ITK-Snap. The purpose of masking is to identify areas of the SPECT image that represent the Regions of Interest (ROIs: Myocardium, Liver, and Left, Right, and Combined Blood Pool) so we can statistically analyze them. Other ROIs such as background organs and myocardial infarctions may be added in the future. All folder paths are from Nuclear_Image_Simulation on Milkyway.

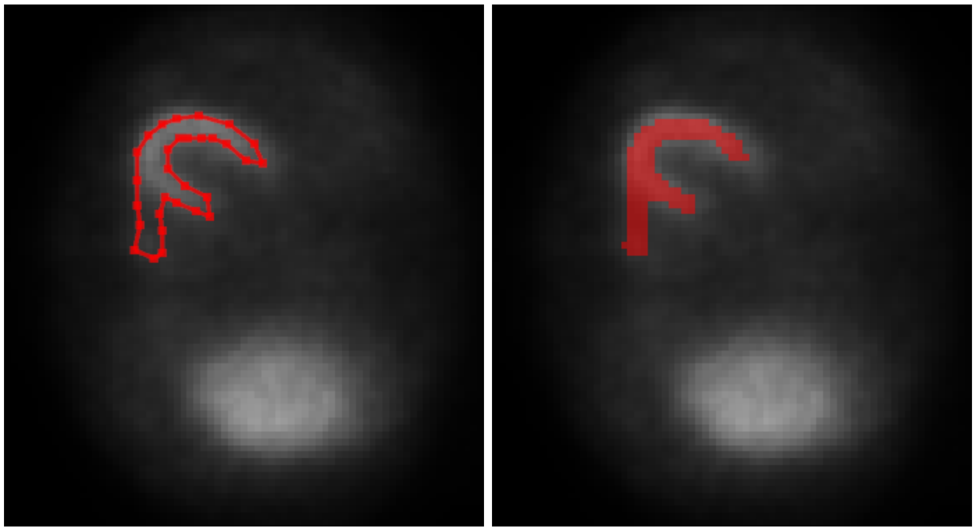
The following steps should be taken to mask an image:

1. Choose an image (from Reconstruction/Output) to mask, and ensure that it is not in All_Masks or your current working Masks folder.
    a. There are multiple Masks folders in Nuclear_Image_Simulation. Masks and Masks2 represent the first and second set of masks completed. Masks3 is currently a work in progress. The contents of completed Masks folders should be copied to All_Masks so we can get combined statistics.
2. Create a folder in your Masks folder with the same image ID as the reconstructed image.
    a. The image ID is [C/D]_[R/S]_[Number]. So for the image Recon_C_R_1497.Vol, the folder would be called C_R_1497.
3. Copy/paste the image into the folder, and rename it "image".

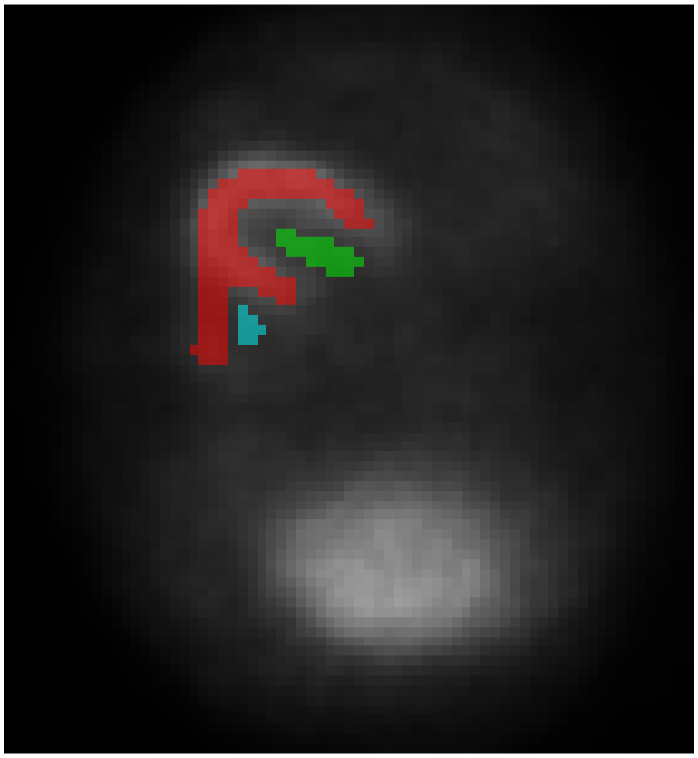
4. Run the executable at the following filepath: /home/dmitra/Desktop/itksnap-4.0.2-Linux-gcc64/itksnap-4.0.2-20230925-Linux-gcc64/bin/itksnap. The itksnap-4.0.2-Linux-gcc64 folder can be found on the Desktop for Milkyway, in the top left corner of the right monitor. This will open ITK-Snap.
5. Click "Open Image" in the bottom right corner of ITK-Snap.
6. Open the "image" file you pasted into the new folder. File format is Raw Binary, image dimensions 128x128x128, voxel type 32 bit floating point (float). Do not touch the other settings.
7. Scroll through the image until you find the heart. The image on the top left is the easiest to work with, as it is the same view as the default in ImageJ.
    a. Auto-adjust the image contrast if needed to make the heart more visible (Tools → Image Contrast → Auto-Adjust Contrast (All Layers) or Ctrl+J)



8. Using the Polygon tool (third from the left in the Main Toolbar, near the top left), draw a shape around the myocardium (that's the upside-down U or W-shaped part of the heart). It's okay if you miss some of it, but make sure you do not get non-myocardium voxels in the mask. When you are done, press "accept" (below the image) to create the mask.
    a. The left part of the myocardium is difficult to see. If you can find it, try to include it so we get more accurate statistics.



9. Repeat Step 8 for each slice containing myocardium
    a. You can use the "paste last polygon" button (under the image) to paste the polygon you just made on the next slice. You will still have to adjust it to make sure it still fits.
10. Switch to a new segmentation label (near the bottom left), then repeat on the left blood pool. Left blood pool is typically Label 5 (light blue).
11. Switch to a new segmentation label, then repeat on the right blood pool. Right blood pool is typically Label 2 (green).



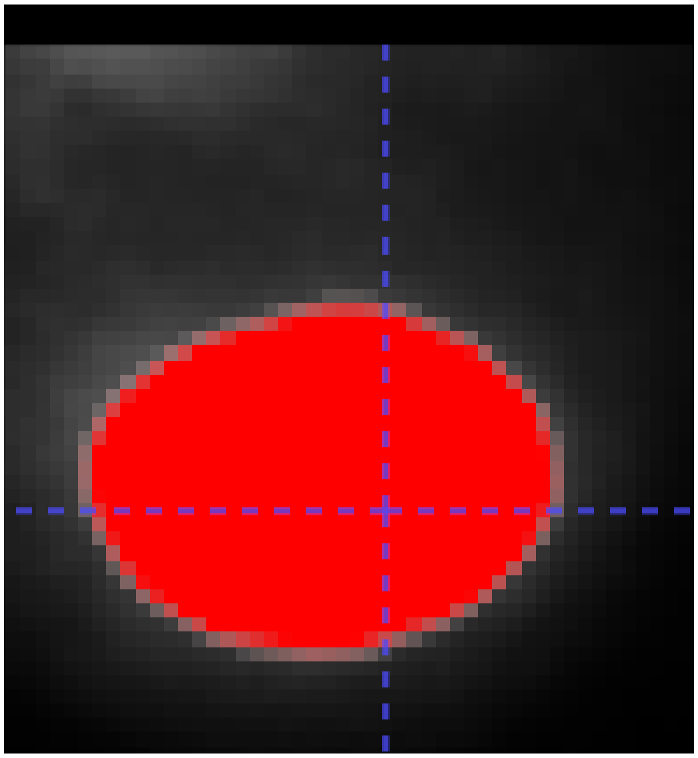
There will usually be gaps between the myocardium and the blood pool. Even if the polygons are right next to each other, only the voxels that are completely inside the polygon will be included in the mask.

12. Save the image as [image ID]_heart_and_blood_pools.nrrd, to the folder with the original image. Go to Segmentation → Save Segmentation Image in the toolbar or use Ctrl+S. Use "Browse" to make sure you are in the right directory, set the File Format to NRRD, then enter the filename. Click Finish to save.
    a. In this step ONLY, the exact naming scheme is not important as long as it is a NRRD file. All other images must be named exactly as written in this document.
13. Find the file you just created in the file explorer and make a copy. This is in case something goes horribly wrong in the next couple of steps, so you don't have to start over.
14. Go to Segmentation → Label Editor or type Ctrl+L to bring up the label editor. Delete Label 1 (myocardium).
15. Save the segmentation image as [image ID]_comb_blood.nrrd
16. Delete Label 2 (Right Blood) in the Label Editor
17. Save the segmentation image as [image ID]_left_blood.nrrd
18. Go to Segmentation → Open Segmentation or type Ctrl+O, then open the Combined Blood segmentation image you created in Step 15.
19. Delete Label 5 (Left Blood) in the Label Editor
20. Save the segmentation image as [image ID]_right_blood.nrrd
21. Open the heart and blood pools segmentation image
22. Delete Labels 2 and 5 (Blood Pools) in the Label Editor
23. Save the segmentation image as [image ID]_heart.nrrd
24. Open the original image using File → Open Main Image or Ctrl+G, or just delete Label 1 (Myocardium) in the Label Editor
25. Find the liver, then click on the Snake Tool (Main Toolbar, second from the right)
26. Move the red dashed lines so they form a box around the area the liver is in (pay attention to the boundaries in the top-right image too - the box around the liver is 3D). Make sure all slices of the liver fit in the box.
27. Click Segment 3D (near the bottom of the Snake Inspector box on the left)
28. Adjust the lower and upper thresholds of the auto-filler, on the right side of the screen. You can manually inspect the image to get an idea of the

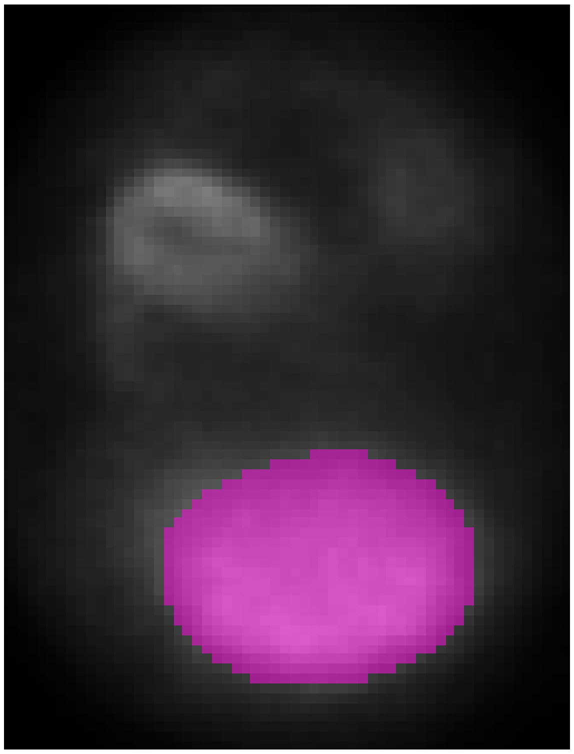
intensities in the liver, but lower threshold should be around .2 for rest or .5 for stress. Upper threshold should be set to either include bright spots (if they look like they are part of the liver) or not include them (if not). It should be arbitrarily high unless you are avoiding bright spots. Click Next when you are ready.

   a. There are other segmentation modes such as clustering that we have not explored. Feel free to experiment with them once you have experience masking.

29. Place a starting bubble for the auto-segmenter. Move the cursor (intersection of blue dashed lines) to a starting point, adjust bubble radius to be around 1 or 2, then press Add Bubble At Cursor. I have gotten best results near the lower-right corner of the third or fourth liver slice, but feel free to experiment. Press Next when ready.
30. Press the Play button and watch the auto-segmenter fill in the liver. Make sure the filled-in area does not reach the kidneys or other organs around the liver.
   a. If nothing is filled in, it probably means your lower threshold is too high. Go back and reduce it, then try again.
   b. As before, it's okay if we don't get the entire liver, and it is better not to get other organs in the segmentation mask.



31. Once you are satisfied with the segmentation, press Finish to finalize the result.
    a. If a little bit of another organ got caught in the mask, you can use the paintbrush tool (Main Toolbar, fourth from the left) and right-click and drag to erase.



32. Save the segmentation as [image ID]_liver.nrrd

When you are done with the segmentation, you can delete the copy you made earlier. Once enough segmentation masks are made, we can use the Statistics module to statistically analyze our SPECT data. Make sure to also copy your Masks folder's contents to All_Masks so they can be included in the aggregate statistics.

Statistical Analysis

Input: SPECT image (128x128x128 raw), masks of all ROIs (128x128x128 NRRD)

Output: Histograms and a JSON file, detailing individual and/or combined SPECT image statistics; Raw binary (128x128x128) versions of the input NRRD masks;

Images of each organ for each patient, as determined by the masks (128x128x128 raw binary)

The statistics pipeline allows us to analyze the data in our SPECT images over the ROIs we masked in the Segmentation Mask module. All code and output from this module is in Nuclear_Image_Simulation/Statistics on Milkyway. Input comes from the Masks folders; See code documentation on how to specify which one. Voxel values in these images were analyzed and fitted to four statistical distributions: Normal, lognormal, gamma, and laplacian. This fitting is nondeterministic and has had different results when tested on the same data. The Kolmogorov-Smirnov test was then applied to find the best fitting distribution.

The primary program you will run here is statistics_pipeline.py. It can be called from the command line with 'python statistics_pipeline.py'. However, you will want to modify a few variables at the top of the program before doing so. "Default values" are the ones they are set to now, and should be set to when you are done using the program. Variables to modify are as follows:

- organ_list: A list of which organs to get statistics for. Each entry should match the naming scheme of the masks, so the program can find the right masks. Current options are "heart", "liver", "left_blood", "right_blood", and "comb_blood". The list contains all of these by default.
- combine_histograms: A Boolean value. If this or do_all is set to True, the program will generate aggregated statistics over the mask folder specified. If both are false, it will only generate individual histograms and statistics for each image. Set to True (get combined statistics) by default.
- do_all: If this is set to True, both individual and combined histograms/statistics will be generated. If it is False, combine_hostograms will dictate which 'mode' is used.
- conditions: Which patient conditions to run statistics on. Options are "C_R_", "C_S_", "D_R_", and "D_S_". Make sure there is an underscore at the end of each entry, as this is used in string building. All options are in the list by default.
- mask_folder: Input folder (in Nuclear Image Simulation folder) that has the masks you want to use. Set to All_Masks by default.
- normalized_output_folder: Where you want the normalized (combined) histograms and statistics to go (in Statistics folder). It's best to either create a

new folder or rename everything in the old one before doing this, so that nothing gets overwritten or misformatted (writing new data to existing JSON files will append it to the old data and make a mess). Currently set to Normalized_Histograms_All. The data in this folder has been renamed to allow the next batch of data to be entered without issue.

- mask_path: Path to the folder specified by mask_folder. You shouldn't need to modify this, unless the mask folder is in a folder other than Nuclear_Image_Simulation.
- experiment_1, 2, and 3: One-off experiments I conducted to get data we normally don't need. Each has a short description of what it does, and is off (False) by default. You can turn them on by setting the relevant value to True.
    - Experiment 1 generates ideal distributions for the given statistics 100 times, and shows which type of distribution was considered the "best" each time. This takes a long time, however, it may be worth it if the distribution type is of interest.
    - Experiment 2 finds the sum of all voxel values in each patient image. No practical use has been found for this, but maybe there will be one.
    - Experiment 3 finds the highest value in each organ for each patient. It was used to look for extreme outlier values.

After creating a new set of patient masks, it is recommended to run the statistics pipeline with do_all=True on the new folder, and with do_all=False and normalize_histograms=True on All_Masks. This way you get individual, small group, and total aggregate statistics without re-running anything.

The following folders contain various outputs from the statistics pipeline:

- Output: A combined output folder from earlier runs of the pipeline
- Output[Masks folder name]: Individual patient outputs from running the pipeline on the specified folder.
    - This will get overwritten if you run individual statistics (do_all=True or normalize_histograms=False) on the folder again, but the statistics should be the same so that is okay.
    - These are divided into control/diseased, stress/rest, with a folder for each image's histograms and distribution variables JSON.
- Normalized_Histograms[identifier]: Combined statistics from the given folder. Normalized_Histograms is from Masks, -2 is from Masks2, and -_All is

from All_Masks. Folders/files in -_All have been renamed to indicate that they are generated from the first two patient sets, so that future data may be added to the same folder without overwriting anything.

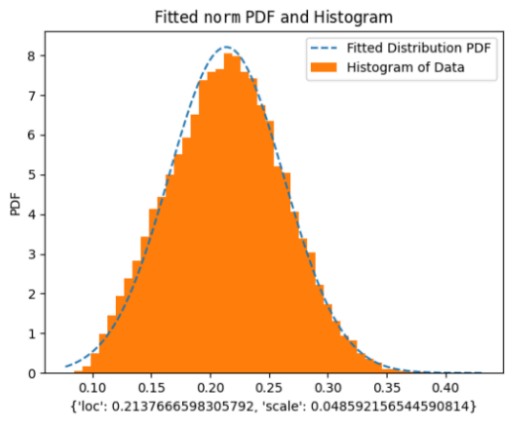
  - Histograms are divided into Control/Diseased, Stress/Rest folders. The file normalized_distribution_variables.json has the parameters for the best-fit statistical function, as well as the actual calculated mean, for each organ.

- Individual_Histograms contains all individual patient histograms, unsorted.
- Binary_Masks are raw binary versions of the NRRD masks found in the Masks folders. There is one for each image/ROI analyzed.
- Organ_Images are the sections of each image covered by the given ROI. There is one for each image/ROI analyzed.

The other random files/folders found in Statistics are as follows:

- ac_stats.py is similar to the Statistics Pipeline, but it is designed to use our existing organ masks and the corresponding attenuation corrected SPECT images. This line of research was abandoned before it could get very far, however, I believe that we can use the original image masks to get statistics from the attenuation corrected images. This code is not as maintained as statistics_pipeline, but should work the same way.
- AC_Histograms are individual patient histograms generated by ac_stats.py
- Normalized_Histograms_AC was meant to be an output folder for combined statistics from ac_stats, however, this was never attempted
- monte_carlo_xcat.py is an early version of XCAT+.
- [rest/stress]_statistical_xcat_test.bin are early XCAT+ phantoms. They are 256x256 raw binary.

Once statistics are generated for all images, the normalized_distribution_variable.json file should be transferred to the XCAT_to_Sino_Pipeline folder on Sunflower and the external computer, replacing the old JSON file, and renamed to "final_distribution_variables.json". If a "public" version is being maintained, it should also be pasted into the Sinogram Simulator folder of the next release. The statistics should also be re-hardcoded into xcat_plus.py, and the latest version pushed to the Github page.



This is an example of a normalized histogram, for control-rest myocardium data (normal distribution).

Where to Find Data

All relevant data locations, types, and sizes are included in the Miscellaneous Code section. They are also included here. All folders/filepaths start from Nuclear_Image_Simulation, on Milkyway.

- Raw SPECT data (DICOMs): Folders containing 20-30 DICOM files (for SPECT images), in subdirectories of the All_Data folder. The data_reference.xlsx spreadsheet in this folder can tell you which directory the raw data for a specific image is in - use Ctrl+F to find the image ID at the end of the number string in Column C Patient_Filename. This can also give you information about that patient's specific disease status. Some entries are repeated in this spreadsheet. Attenuation maps can be found here too, in folders containing a single DICOM file. This file's binary data is the attenuation map. This data is provided by Dr. Youngho Seo's lab at the University of California, San Francisco, and is the basis for all of the data discussed in this document. Technical information about the imaging protocol is as follows:
    - The tracer used is 99mTc-Sestamibi
    - Tracer dosage was 10mCi for rest and 25mCi for stress
    - Images were captured with a dual headed SPECT camera (Infinia Hawkeye 4, GE Healthcare)

  - More information can be found in the DICOM file for each sinogram
- Binary Sinograms (extracted from DICOMs): 128x128x128 raw binary files, in the Reconstruction/Sinograms folder. Each is labeled according to the corresponding image ID.
- Reconstructed SPECT Images: 128x128x128 raw binary files, in the Reconstruction/Output folder. Identified by patient type and image ID. 10 MLEM iterations with no attenuation correction.
- Attenuation Corrected SPECT Images: 128x128x128 raw binary files, in AC_Output.raw. 20 MLEM iterations with attenuation correction.
- Masks: 128x128x128 NRRD files, in the All_Masks, Masks, Masks2, and (eventually) Masks3 folders. Each folder in these directories is labeled with an image ID, and contains masks for the corresponding image.
- Binary Masks: 128x128x128 raw binary files, in the Statistics/Binary_Masks folder. Raw binary versions of the NRRD masks.
- Organ Images: 128x128x128 raw binary files, in the Statistics/Organ_Images folder. These are SPECT images, cropped to the region covered by the corresponding mask (everything else is zeroed/blacked out)
- Individual Histograms: PNG files, in Statistics/Individual_Histograms. There is one for each image, for each organ, for each statistical function (Normal, Lognormal, Gamma, or Laplacian).
- Normalized Histograms: PNG files, in Statistics, in any folder starting with "Normalized_Histograms". Most recent results on our full patient set are in Normalized_Histograms_All.
- Individual Distribution Variables: JSON files, in Statistics/OutputAll_Masks, sorted into subfolders by patient group (control/diseased, stress/rest).
- Normalized Distribution Variables: JSOn file, in Normalized_Histograms_All. Filename is normalized_distribution_variables_2_patient_sets.json.

Results

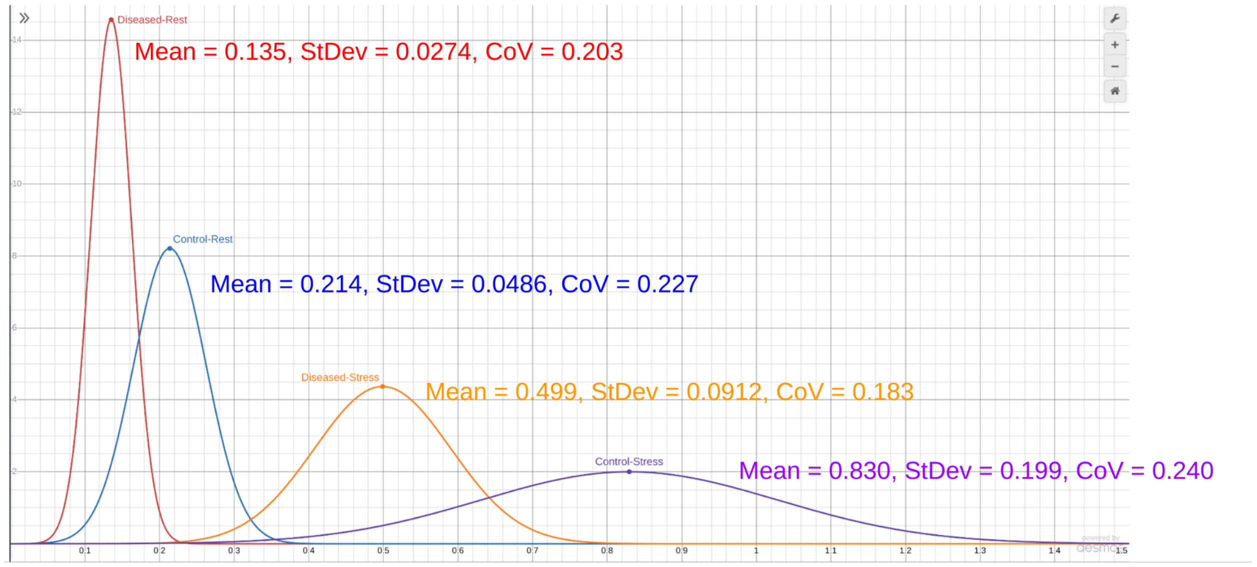
This section contains the current results of this project. Although a user manual is not necessarily the best place for results, understanding these is important to understanding the project. As such, it is included in the "Lab Manual" portion.

Statistics Project

Our observations across normalized patient data are largely consistent with what would be expected. Diseased-Rest patients consistently had the lowest voxel values (indicative of less tracer concentration, and less blood flow), followed by control-rest, diseased-stress, and control-stress. The difference between rest and stress is because stress patients were injected with 2.5 times as much tracer as rest patients.

Of the regions of interest we looked at (Myocardium, Liver, and Left, Right, and Combined Blood Pools), only the myocardium consistently followed the same distribution type in all cases, as even in the 5-image sets it was consistently normal. The blood pools tended to be either lognormal or gamma (which are very similar in terms of shape), only occasionally showing up as normal or laplacian. The liver was inconsistent, sometimes appearing normal or laplacian and sometimes appearing lognormal or gamma.

We also observed that our patient sets have relatively consistent coefficients of variation (standard deviation divided by mean). Diseased-Stress patients had the lowest CoV, followed by Diseased-Rest, Control-Rest, and Control-Stress. The reason for this is unknown and the differences may not be significant. A plot of all normalized myocardium distributions, with the CoV noted, can be found below:



Finally, we noticed a lot of variance looking at individual patient distributions. The majority of individual patient means tend to be below the normalized mean, and some outlier images will have myocardium voxel values averaging over twice the normalized mean for that patient set.

The rest of this section is a discussion of a correlation between imaging date and voxel value. It is probably more in-depth than it needs to be. But, if it ever becomes relevant, the documentation for it is here.

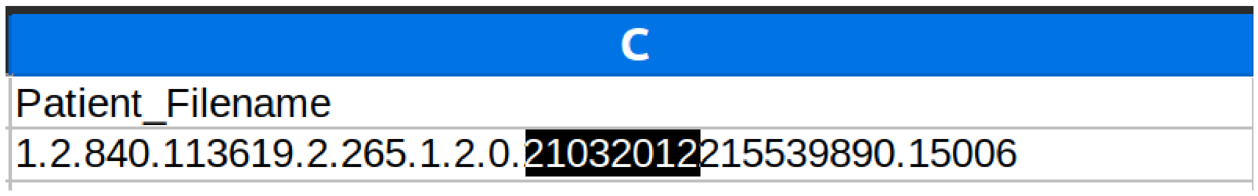
An informal analysis has revealed a correlation between year of imaging and voxel values: 7 out of the 9 patients imaged in 2012 were on the high end of their group's distribution, including two of the three whose myocardium voxel values averaged more than two (normalized) standard deviations above the (normalized) mean in at least one image. Meanwhile, only 1 of the 9 patients imaged in 2014 were identified as being on the high end of their respective patient group. 2013 had a mixture of higher and lower distributions, including the third "high" outlier. These high vs low designations were made informally, based on observed trends in individual image data rather than combined patient statistics (as those had not been generated yet), and were made without knowledge of which patients were imaged on which dates.

There is probably no reason to further investigate this, and it may very well be a waste of time, however, it is an interesting pattern and if we could figure out why it is occurring it might have some impact on how we understand our data. The same imaging protocol and Tc-99 doses were used for all patients across all three years; This was confirmed by Dr. Seo in a meeting.

If you need to find the date a SPECT image was taken, you can do so in one of two ways:

- Find the imaging date in the DICOM file
- Find the imaging date/time in the Data_reference.xlsx spreadsheet. In column C (Patient_Filename), the beginning of the second-to-last number partition includes an encoded data-time string. The first 7 or 8 numbers are the date in (D)D-MM-YYYY format; Leading 0s have been cut off, so the date will either be one or two digits, whichever one leads to a year between 2012 and 2014 in this format. An unknown number of digits after this represent a

time, which is likely the file creation time. The date typically remains the same for the same patient, while the time does not.



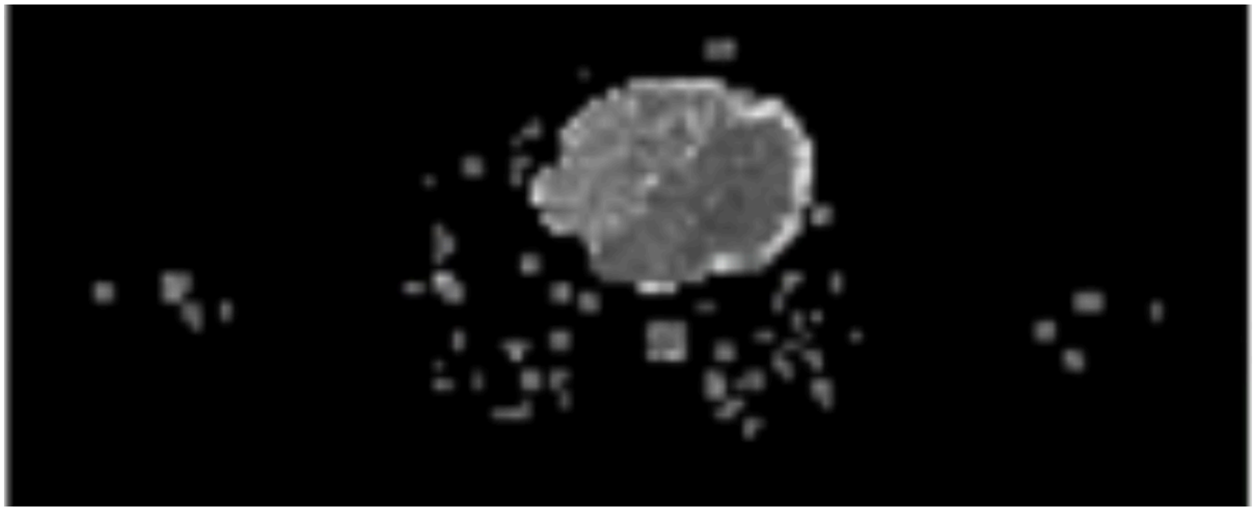
C
Patient_Filename
1.2.840.113619.2.265.1.2.0.210320122215539890.15006

Screenshot of the imaging date (March 21st, 2012) in Data_reference.xlsx. This is the date for image 15006, a Diseased-Rest image. The mean myocardium voxel value for this image is about .168, which is just over 1 standard deviation above the mean; It is high, but not (on its own) anomalously so.

Tomographic Medical Image Reconstruction with Deep Learning

This section has notes and results from Senior Design Group 1, and the programs that are considered part of the project (including XCAT+, which was mostly completed before the project began).

The XCAT+ that is used for this project is relatively simple; It only has distributions for the myocardium, liver, and left and right blood pools. Senior Design Group 2 is working on other more advanced XCAT+, including heart infarctions and background. Because we did not have background noise in our XCAT+, which was used in AI training as the expected output, our AI only really learned to reconstruct the heart and liver, on sinograms of comparable size (but not shape) to the real data. This is fine for our synthetic images, as they only include the heart and liver, but for real data it will likely interpret the entire sinogram as pertaining to the heart and liver, though this is not the case. As such, I do not believe we will see any substantial results testing the AI on real data.

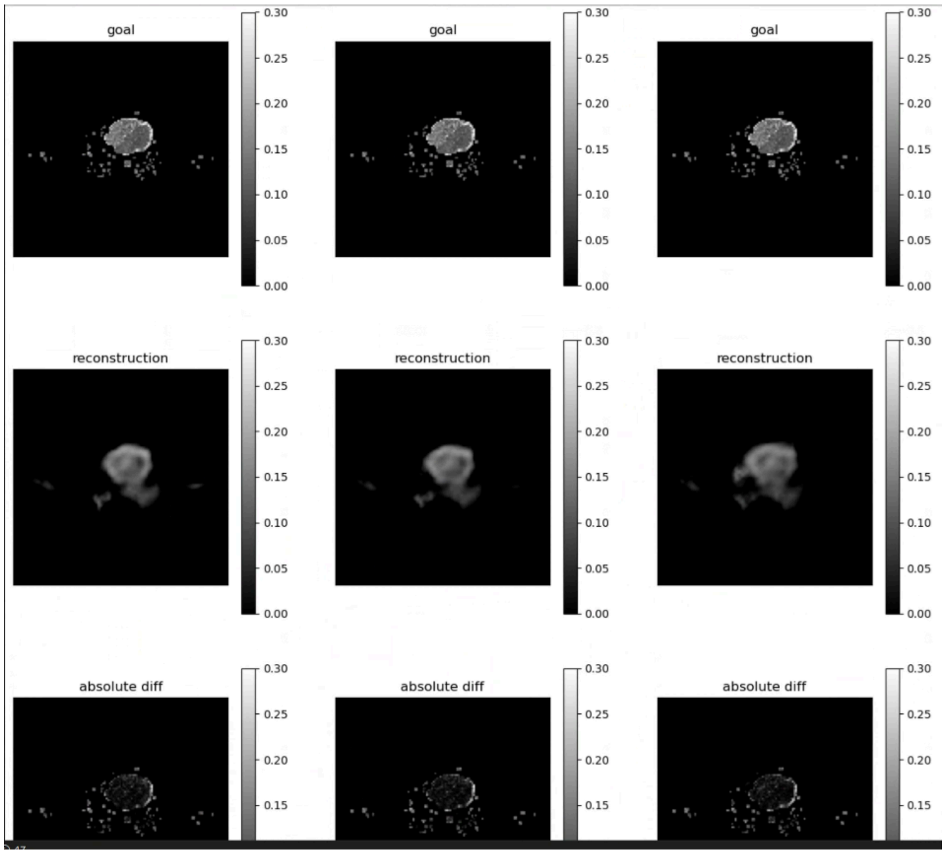

XCAT+ example from Senior Design Group 1, of the variety used for data generation. Blank space above/below the heart removed for ease of viewing.

For our data generation project, we ended up generating XXXX sinograms for training data. We achieved this number through the following processes:

- XX unique XCAT phantoms were used. 12 of these were generated from manually modified Organ ID Params files, and the rest used random parameter generation. Each phantom had XCAT+ applied to it.
- Each of these XCAT+ phantoms was used with the GATE simulation 5 times, producing 5 sinograms per phantom. At this stage we have XX*5=XXX sinograms.
- Each sinogram had z-axis rotation applied 5 times at different angles. This gave us XXX*5=XXX sinograms.
- Finally, each sinogram had 10 XY shifts applied using a forward project-shift-back project algorithm, and the original was also kept (so 11 outputs were produced per input at this stage). This gave us XXX*11=XXXX sinograms.

Unfortunately, there was very little variance in the XCAT phantoms used, even with the random parameter generation. The expected output images all look similar to the XCAT+ phantom above, and the sinograms look somewhat like the one pictured below.

Despite the visual similarities in our synthetic data, it was sufficient to train the AI to reconstruct our synthetic data. When testing on synthetic images that were kept separate from training data (the AI had not been trained on data from the same XCAT phantom), the AI was able to consistently achieve a RMSE score between 0.07 and 0.09, and a (1-SSIM) score between 0.1 and 0.11. This was achieved by training the AI using the (1-SSIM) loss function and SiLU activation function. A sample of the AI results on this synthetic data can be found below:



As can be seen from this image, the AI outputs tended to be blurry, and often placed an unknown structure near the lower right corner of the heart (in the view from these slices); However, it was able to reconstruct the general structure of the heart.

Tests on real data are pending, as the difference in output dimension between the AI and our iterative reconstruction makes automatic comparison (with SSIM or RMSE) very difficult.

Approximate Runtimes

Below is a table of select processes described in this file and approximate runtimes.

Process	Computer	Runtime	CPU Time	Notes
Static Reconstruction (SIFADS, 20 iterations)	Milkyway	45m		
Statistics pipeline normalization (6 images, all organs, same condition) (no individual results)	Milkyway	3m 24s		Number of images shouldn't have significant impact, but number of conditions will
Individual histogram generation (1 image, all organs)	Milkyway	20s		Approximate time per image
XCAT+ (128x128x600 phantom, all organs, with Gaussian blur)	Milkyway	55s		
XCAT+ (128x128x600 phantom, all organs, no Gaussian blur)	Milkyway	49s		Time added by Gaussian blur is negligible - about 6s

XCAT+ (128x128x600 phantom, all organs, with Gaussian blur)	Andromeda	1m 10s		With unknown notebook running in background
XCAT creation	Sunflower	36s		With GATE running in background
GATE simulation (1 iteration)	Sunflower	12 hours		Rough approximation; Precise timing TBD
Data augmentation (1 XCAT/5 Sinograms)	Sunflower	19m 3s		With GATE running in background
AI training (RMSE/SiLu)	Sunflower	3h 7m		With GATE running in background

Developer's Manual

This section includes details of code for the Tomographic Medical Image Reconstruction with Deep Learning project that a user should not have to interact with.

Sinogram Simulator

Code for the Sinogram Simulator can be found in the "src" and "GATE Simulation" folders. Almost all of it was created by a former lab member, Tommy Galletta, not our Senior Design group. We have mostly been running this code as a black box, however, we have included a basic overview of what each file does here.

SRC

This folder contains Python source code to run the GATE simulation. A brief description of each file, and what (if anything) you may need to update, can be found below.

XCAT_crop_into_GATE: This is the most likely file to be modified in src. It takes the XCAT phantom that is used as input, crops it to the area around the heart, applies the XCAT+ process to the XCAT phantom, and saves the XCAT phantom for use as input to the GATE simulation. Cropped XCAT and XCAT+ phantoms are saved to the GATE Simulation folder through this program.

Combine_parallel_outputs: The GATE simulation is actually a number of GATE processes run in parallel, each of which has its own output. This program combines these outputs into the final sinogram (combined_parallel_results.bin) and a header file that contains relevant metadata.

Fix_mhd: Updates the combined parallel output header file to include information about the sinogram (results) file.

Produce_macro_files: Creates macros to run the GATE simulation, using custom parameters based on the input variables and current time.

Produce_mhd_files: Creates mhd files that are used by the GATE simulation, using the parameters for the input.

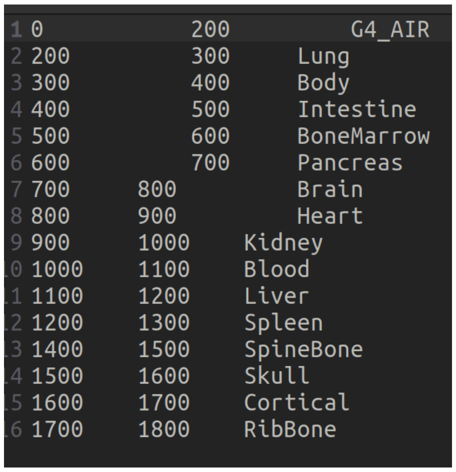
GATE Simulation

This folder contains macros and other files that are used to run the GATE simulation. Generally speaking, these are called by each other or by multiple_runs.py during normal operation. A brief description of each file, and what (if anything) is likely to be changed, can be found below.

XCAT Materials: A text file showing the values for different organs/body parts in the input (non-XCAT+) phantom. Formatted as:

Min Max Organ

for each line, where min is the min value that corresponds to the organ (inclusive), max is the max value (exclusive), and Organ is the organ name. These values should be changed to match that of your Organ ID parameters file. An example (that is consistent with the default values used throughout this project) is shown below:



```
0           200         G4_AIR
200         300     Lung
300         400     Body
400         500     Intestine
500         600     BoneMarrow
600         700     Pancreas
700     800         Brain
800     900         Heart
900     1000    Kidney
1000    1100    Blood
1100    1200    Liver
1200    1300    Spleen
1400    1500    SpineBone
1500    1600    Skull
1600    1700    Cortical
1700    1800    RibBone
```

SimVisu: A macro to show a visualization of the GATE simulation. Useful to make sure any changes have not caused anything to break.

SimAutoRun: A macro to set up and run the simulation. <u>To change the XY dimensions of the sinogram outputs (currently 128x128), change the pixelNumberX and pixelNumberY variables.</u>

setActivityPosAndScale.mac: Generated by produce_macro_files.py. Sets the scale and position of the GATE simulation to the appropriate values.

result_n.sin, result_n.hdr: Generated for n=0 to n=23. Results of the simulation running in 24 parallel processes, and combined into the final output by combine_parallel_outputs.py.

PopulateScene.mac: A macro to set up the GATE simulation. We have not experimented with modifying this file, but if you wanted to modify the GATE simulation, you would do so here.

parallel: A folder containing 24 macros to run the GATE simulation in 24 parallel processes.

MoveVisu.mac: A macro to advance the simulation at 16 37.5 second intervals. 37.5 seconds is simulation time, not real time. Originally published by OpenGATE.

GateMaterials.db: Contains information about elements and materials simulated in GATE, such as density and composition. Formatting is as follows:

```
[Elements] // list of elements, with S (Symbol), Z (Atomic #), and A (Atomic mass)
Hydrogen:   S= H   ; Z=  1. ; A=   1.01  g/mole
```

Repeated for elements He-Ge, Y, Mo, Ag, Cd, Sn, Te, I, Cs, Gd, Lu, W, Au, Tl, Pb, Bi, U

[Materials] // list of materials, with d (density), n (unknown), state, & composition. Composition is relative, by weight.

```
Carbon: d=1.8 g/cm3; n=1; state=solid
        +el: name=auto; n=1
DielectricOil: d=0.885 g/cm3; n=3; state=liquid
        +el: name=Carbon; n=12
        +el: name=Hydrogen; n=8
        +el: name=Chlorine; n=2
WaterS: d=1.00 g/cm3 ; n=2
        +el: name=Hydrogen  ; f=0.112
        +el: name=Oxygen    ; f=0.888
```

Includes materials used in the detector head (Glass, LuYAP-80, etc) and in a human body (blood, fat, muscle, etc).

cropped_XCAT_activity.mhd: Gives information about the cropped XCAT activity file (XCAT+), including image dimensions.

cropped_xcat.mhd: Gives information about the cropped XCAT file, including image dimensions.

cropped_XCAT_activity.bin: XCAT+ used as input for the simulation. Raw binary image, same dimensions/cropping as the original cropped XCAT.

cropped_XCAT.bin: XCAT phantom input, cropped on the horizontal (transverse) plane to 64 slices centered on the heart, and on the other planes to the edge of the non-zero values in the XCAT phantom. Raw binary image, size ?x?x64 (dimensions are 72x58 for out input XCAT phantoms, but phantoms generated with a different base will have different dimensions). Size information can be found in cropped_xcat.mhd. Used to determine interference from body tissues in GATE simulation.

Miscellaneous

These programs are included in the Sinogram Simulator folder, and are used to run the GATE simulation.

Parsing: Contains helper methods for file parsing.

Multiple_runs: Calls the appropriate bash scripts to run the GATE simulation. Accepts one argument, which is the number of times to run the simulation. Assumes the input XCAT is in Input/gens, as is the case when this program is called from run_simulation.py.

Reconstruction AI

The only backend file that was used for the reconstruction AI (besides those used for the Sinogram Simulator) is XCAT+_Ceda_Pytorch.ipynb. It is a Jupyter notebook, so documentation is broken down into the relevant code chunks.

[Could you please do this, Asher? I'm not too familiar with the AI notebook.]