System Overview and Workflow:

Input:

- First, use a software called XCAT to produce simulated images that will be used for training.
- XCAT consists of assorted binary files.
- A separate **text file** shows the region number for a specific body part.

Material Map Generation:

- Use the **xcat program** to generate an XCAT binary.
- A script is used to produce the **cropped XCAT** and **activity map**.
- **Material Map**: Equivalent to the cropped XCAT.

Sinogram Production:

- **Run Sinogram**: initialize-simulation, then multiple-runs.py [number_of_runs].
- Running the activity produces a **sinogram**.

OpenGate Simulation:

- Once the sinogram has been produced, the file containing it is sent to a program called OpenGate or GATE which simulates the medical imaging process.
- **-v**: Visualize the simulation after initialization.

Synthetic SPECT Production:

- First, perform data augmentation by applying **statistical methods** to generate slightly different versions of the data for better generalization.
- XCAT images are passed through **GATE** (via initialization script).
- **GATE** delivers a **synthetic SPECT** (3D array).
- The 3D array is fed into a **model for training**.

ImageJ/Fiji and ITK-SNAP

- To view the image from the binary files:
  - Open the ImageJ app
  - Click on File → Import → Raw
  - Navigate to the file you want to open via the computer's file explorer
  - Click on the file
  - Enter the file dimensions
    - For the reconstructed images, this will be 128x128x128, with data type 32-bit floating point
  - The image should appear in a new window.
- To adjust the image brightness:
  - Select the image whose brightness you want to adjust

    - In the main ImageJ window, go to Adjust → Brightness and Contrast
    - This will open a window where you can adjust the brightness and contrast of the image
    - For best results, click the "Auto" button in this window; This will automatically adjust the brightness/contrast to an "optimal" level.
- To resize the image:
    - Click on the image
    - Press the Up arrow key to increase image size, or the Down arrow key to decrease image size
- To view another slice of a 3d image:
    - Click on the image
    - Press the left arrow key to view the previous slice or the right arrow key to view the next slice
    - To view many slices in succession, press and hold the corresponding arrow key, click and drag the bar at the bottom of the image, or press the Play button next to this bar in order to automatically scroll through the slices in order, starting from the current slice.
- Alternatively, ITK-SNAP can be used to view the image across multiple views at the same time:
    - Open the ITK-SNAP application
    - Select "Open Main Image" and choose to upload from computer
    - Use the file browser to select the file you want to open
    - This will open a view across multiple planes
    - Users can also draw boundaries of the organs in ITK-SNAP

Model Output:

- Model architecture: **CIDA++** (an autoencoder-decoder network).
- The model outputs a 3D array.
- The output is compared with the **expected output**.
- The comparison result is fed into the model’s **loss function** for optimization.

External Software Information

XCAT Information

What is XCAT?
XCAT (Extended Cardiac-Torso) is a highly detailed, anatomically realistic phantom (a digital model of the human body) designed for simulating medical imaging modalities like SPECT and CT. It provides high-resolution images of human anatomy that can be used in imaging simulations, research, and medical system development.

XCAT provides detailed human models, including variations for different body parts like the head, heart, and limbs. We will be using XCAT to create models of the human heart.

These phantoms are used in imaging simulations to model realistic human tissues and organs, allowing for accurate testing and validation of medical imaging techniques like **SPECT** and **CT**.

Data Types in XCAT
XCAT phantoms are typically represented in a voxel format, where the human body is divided into a 3D grid of small cubes (voxels) that represent specific tissue types. The voxel size is 4.42 mm in every direction, meaning each voxel represents a 4.42 mm cube of tissue. This allows for detailed anatomical modeling in simulations.

Material Maps
Material maps are generated from XCAT files and are essential for distinguishing different types of tissues in the simulated anatomy. The material map provides information on tissue densities and compositions, which can be fed into simulation software like GATE to model radiation transport and interactions with different tissue types.

Activity Maps:
XCAT can also generate activity maps that describe where radioactive tracers (or other agents) are distributed in the body for imaging simulations. These activity maps are essential for simulating imaging systems like SPECT, where the distribution of a radioactive tracer affects the outcome of the imaging process.

OpenGate (GATE) Information

What is GATE?
GATE (Geant4 Application for Tomographic Emission) is a simulation toolkit tailored for medical imaging, radiation therapy, and nuclear medicine. It is built on top of Geant4, a platform used for simulating particle interactions with matter, widely utilized in physics and medical research.

GATE can simulate gamma-ray detection for SPECT imaging, involving radioactive tracers. The simulation captures the interaction of photons within a detector, creating realistic SPECT imaging for testing and validation purposes. Similarly to XCAT, GATE supports voxel-based phantoms, representing human tissues or organs as 3D grids of volume elements (voxels). Voxels are 4.42 mm in every direction, helping simulate the interaction of particles with a detailed anatomical structure.

The **material map** represents different tissue types and their densities, essential for accurate radiation and interaction modeling.

For example, files like **cropped XCAT** describe specific human anatomy sections, providing essential information for GATE to model radiation behavior accurately.

The **-v** flag or similar settings can enable real-time visualization of the simulation, allowing users to observe particle interactions, radiation transport, and imaging procedures within the simulation.

ImageJ Information

What is ImageJ?
ImageJ is an open-source image processing program widely used in scientific and medical research. It is primarily designed for analyzing and manipulating images, particularly in fields like biomedical imaging, microscopy, and radiology. It can handle a variety of image file formats and has extensive support for plugins and macros that extend its functionality.

Running ImageJ via Terminal

- ImageJ can be run from the terminal by executing the command:
  `run imageJ [file_name]`
- This allows users to quickly visualize and process binary images (such as those produced by simulations).

Image Analysis
ImageJ provides a suite of image analysis tools for tasks such as:

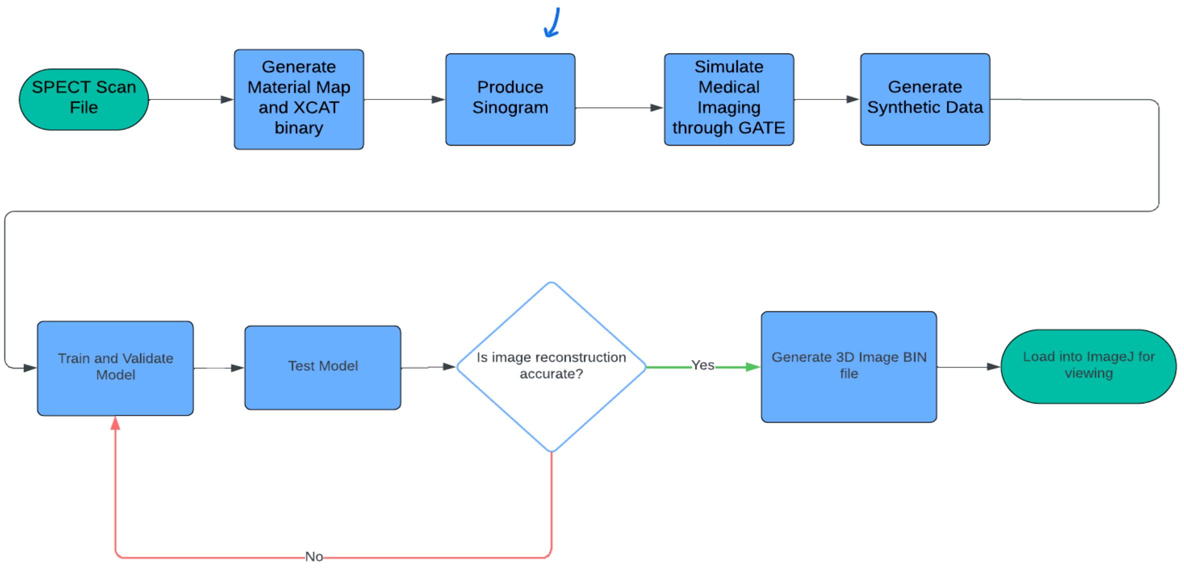
- **Segmentation**: Identifying and outlining structures within an image (e.g., organs or tissues).
- **Measurements**: Calculating area, perimeter, or intensity of structures.
- **Filtering**: Smoothing, sharpening, or otherwise enhancing images for better visual or analytical results.

Neural Network Architecture Information:

The neural network architecture is based on CEDA (Convolutional encoder-decoder with Attention).
4
It will consist of two components: one encoder and one decoder. The encoder will consist of at least four convolutional layers, two attention layers, and two linear layers.

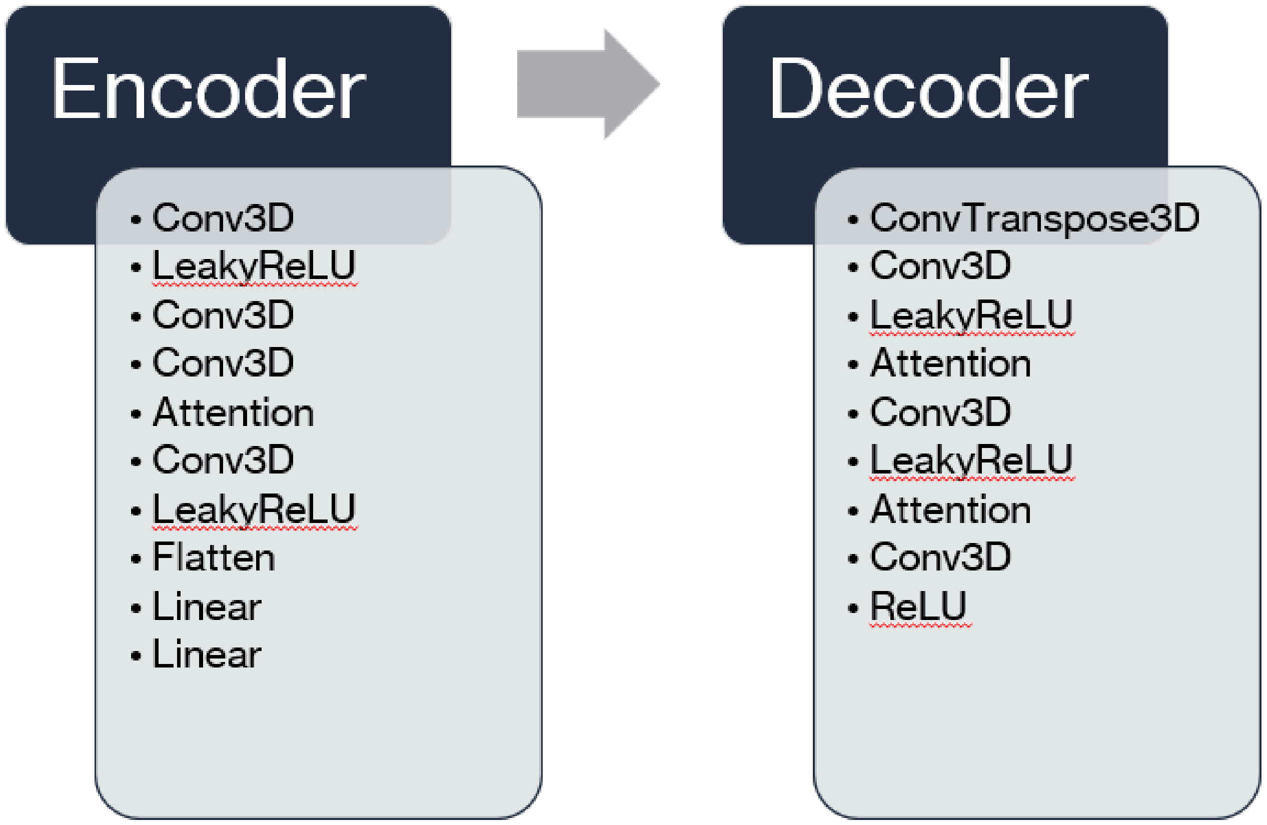
The decoder will consist of a similar layout, with multiple convolutional layers followed by attention layers and two linear layers.

System Flowchart:


SPECT Scan File
Generate Material Map and XCAT binary
Produce Sinogram
Simulate Medical Imaging through GATE
Generate Synthetic Data
Train and Validate Model
Test Model
Is image reconstruction accurate?
Yes
No
Generate 3D Image BIN file
Load into ImageJ for viewing

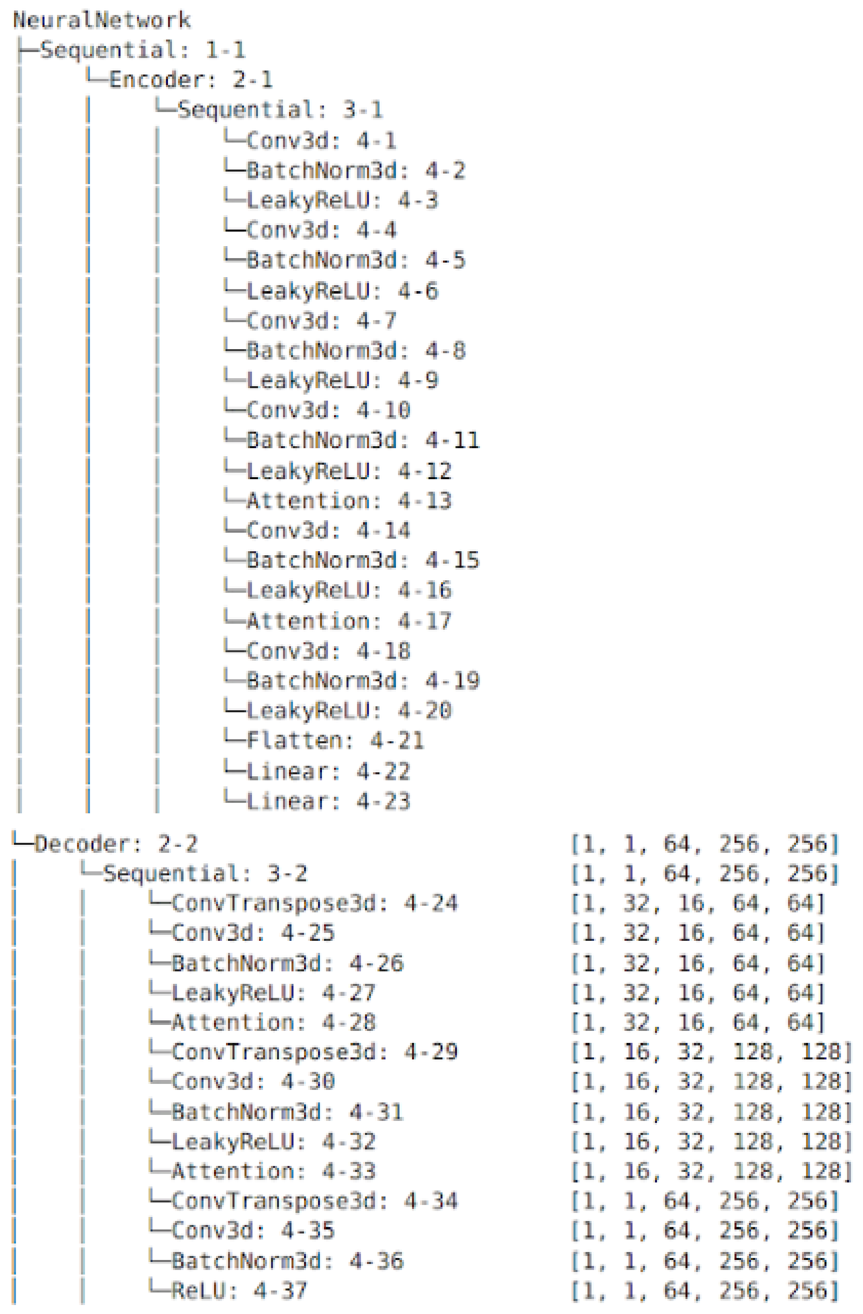
Model Architecture Proposals:

Basic Template



Old Model Architecture

Note: This is the baseline model that is in the lab, but since it is not giving the desired results, it will be completely revised. However, the encoder-decoder structure with CNNs and attention layers will remain the same.



```
NeuralNetwork
├─Sequential: 1-1
│    └─Encoder: 2-1
│    │    └─Sequential: 3-1
│    │    │    └─Conv3d: 4-1
│    │    │    └─BatchNorm3d: 4-2
│    │    │    └─LeakyReLU: 4-3
│    │    │    └─Conv3d: 4-4
│    │    │    └─BatchNorm3d: 4-5
│    │    │    └─LeakyReLU: 4-6
│    │    │    └─Conv3d: 4-7
│    │    │    └─BatchNorm3d: 4-8
│    │    │    └─LeakyReLU: 4-9
│    │    │    └─Conv3d: 4-10
│    │    │    └─BatchNorm3d: 4-11
│    │    │    └─LeakyReLU: 4-12
│    │    │    └─Attention: 4-13
│    │    │    └─Conv3d: 4-14
│    │    │    └─BatchNorm3d: 4-15
│    │    │    └─LeakyReLU: 4-16
│    │    │    └─Attention: 4-17
│    │    │    └─Conv3d: 4-18
│    │    │    └─BatchNorm3d: 4-19
│    │    │    └─LeakyReLU: 4-20
│    │    │    └─Flatten: 4-21
│    │    │    └─Linear: 4-22
│    │    │    └─Linear: 4-23
└─Decoder: 2-2                              [1, 1, 64, 256, 256]
│    └─Sequential: 3-2                      [1, 1, 64, 256, 256]
│    │    └─ConvTranspose3d: 4-24           [1, 32, 16, 64, 64]
│    │    └─Conv3d: 4-25                    [1, 32, 16, 64, 64]
│    │    └─BatchNorm3d: 4-26               [1, 32, 16, 64, 64]
│    │    └─LeakyReLU: 4-27                 [1, 32, 16, 64, 64]
│    │    └─Attention: 4-28                 [1, 32, 16, 64, 64]
│    │    └─ConvTranspose3d: 4-29           [1, 16, 32, 128, 128]
│    │    └─Conv3d: 4-30                    [1, 16, 32, 128, 128]
│    │    └─BatchNorm3d: 4-31               [1, 16, 32, 128, 128]
│    │    └─LeakyReLU: 4-32                 [1, 16, 32, 128, 128]
│    │    └─Attention: 4-33                 [1, 16, 32, 128, 128]
│    │    └─ConvTranspose3d: 4-34           [1, 1, 64, 256, 256]
│    │    └─Conv3d: 4-35                    [1, 1, 64, 256, 256]
│    │    └─BatchNorm3d: 4-36               [1, 1, 64, 256, 256]
│    │    └─ReLU: 4-37                      [1, 1, 64, 256, 256]
```