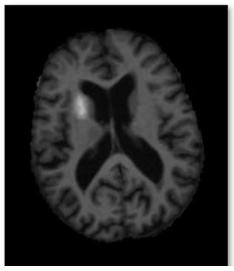
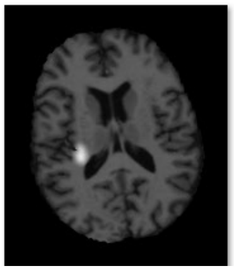
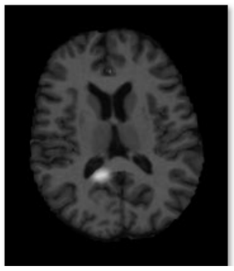
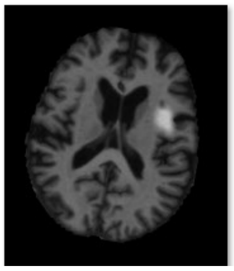
MEDICAL IMAGING
DR. NIKOLAOS V. TSEKOS

IMAGE SYNTHESIS, AUGMENTATION AND CHARACTERIZATION

Bryan Abner 2131017, Computer Science, Undergraduate; Natalie Severino, 1811557, Computer Science, Undergraduate; Melvin Turcios, 2037459, Computer Science, Undergraduate; David Wu, 1962059 , Computer Science Undergraduate

Introduction

In this report we present a medical image synthesis and augmentation simulator. We are generating undersampled images from a high-quality image. The algorithms and Graphical User Interface (GUI) were developed using Python. The algorithms performed are an image generation and two kinds of modification on the images, and the results of the simulator were shown.

Objectives

1. MRI acquisition: Simulate MRI data collection
2. K-Space truncation: Perform K-Space truncation on the original image
3. Gaussian Noise addition: Perform Gaussian Noise addition on the K-Space truncated image
4. Image augmentation: Generate augmented images of the original image
5. Generate final processed image: Generate final images with K-Space truncation and Gaussian Noise.
6. Graphical User Interface: Design and implement a GUI that is able to perform different tasks of the previous objective

Methods

We will be using in the following MRI phantoms from the groundtruth folder in our report.

Image 01

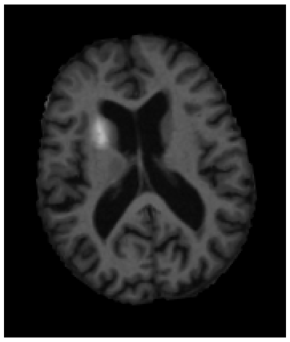


Image 02

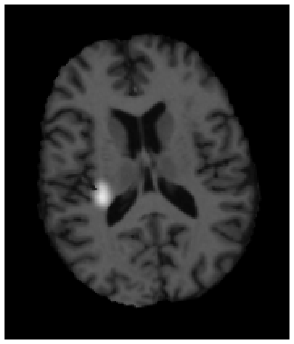


Image 03

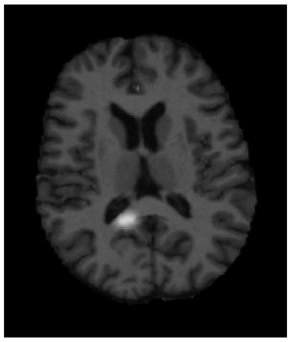


Image 04

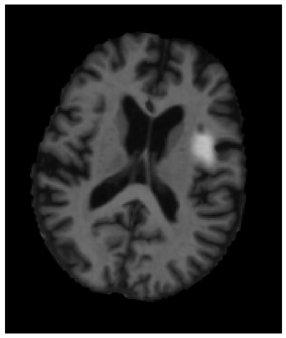
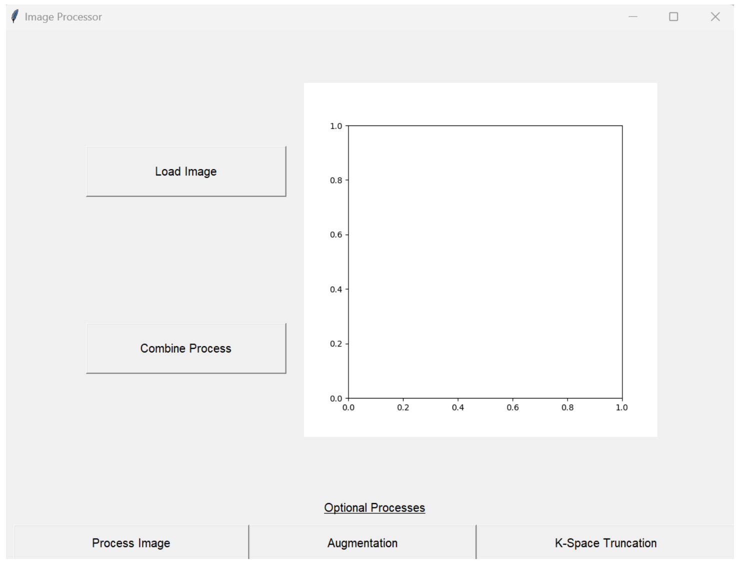
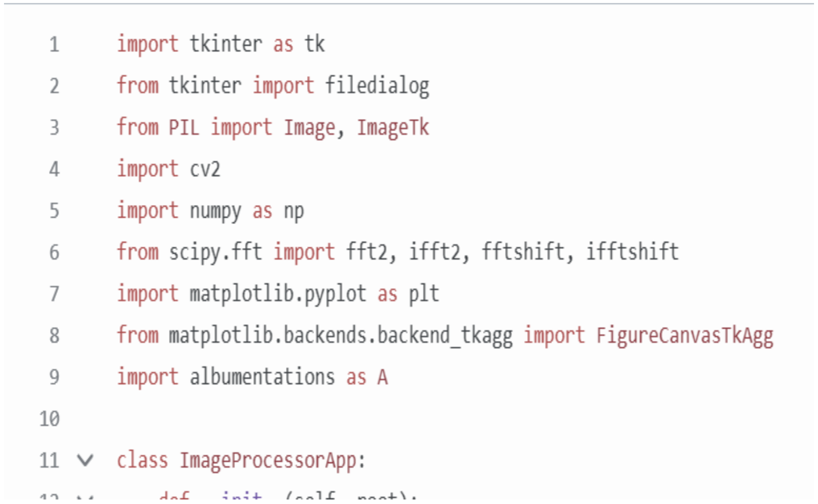


Figure of GUI:



Algorithms

Image 01

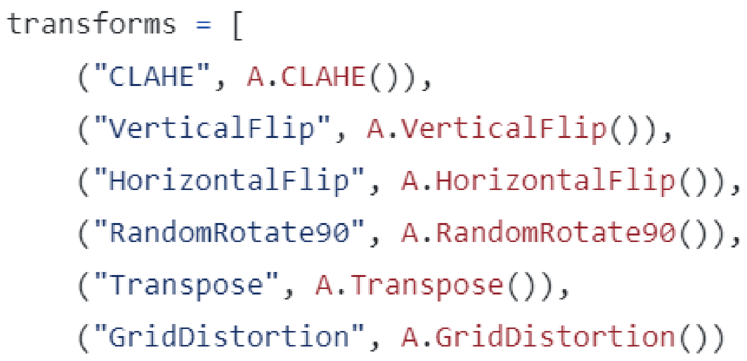


```
1     import tkinter as tk
2     from tkinter import filedialog
3     from PIL import Image, ImageTk
4     import cv2
5     import numpy as np
6     from scipy.fft import fft2, ifft2, fftshift, ifftshift
7     import matplotlib.pyplot as plt
8     from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
9     import albumentations as A
10
11    class ImageProcessorApp:
12        def __init__(self, root):
```

In order to artificially expand the size of a training dataset by creating modified versions of images in the dataset referred to on ***Aim #4***, we used the following algorithms shown in the code snipped of figure 01. During the augmentation transformation algorithms were imported by the albumentations (line 9) library in Python. Here is a breakdown of the transforms shown in figure 02:

CLAHE (Contrast Limited Adaptive Histogram Equalization):
This is an image processing technique used to improve contrast in images. It computes several

Image 02



```
transforms = [
    ("CLAHE", A.CLAHE()),
    ("VerticalFlip", A.VerticalFlip()),
    ("HorizontalFlip", A.HorizontalFlip()),
    ("RandomRotate90", A.RandomRotate90()),
    ("Transpose", A.Transpose()),
    ("GridDistortion", A.GridDistortion())
```

histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values.

This will help output each image separately and allow us to call any predetermined function we might want or need.

VerticalFlip:

This algorithm flips the image vertically. It's a simple transformation that mirrors the image along the horizontal axis.

HorizontalFlip:

Similar to VerticalFlip, this algorithm flips the image horizontally, mirroring it along the vertical axis.

RandomRotate90:

This algorithm rotates the image by a random multiple of 90 degrees.

Transpose:

In the context of image processing, transposing an image involves swapping the row and column indices of each pixel, effectively rotating the entire image 90 degrees counterclockwise.

GridDistortion:

This algorithm distorts the image according to a grid of transformations.

Each transformation is applied to the image, and the result is saved to a file. The transformations are applied in the order they are listed in the transforms list. The exact details of each algorithm can vary based on the parameters provided when they are called, and these can be updated at any time upon request.

Moreover, there were several algorithms working with our libraries to help implement our ***Aim #2*** related to image processing and Fourier transforms (line 6, image 01).

Fast Fourier Transform (FFT):

The fft2 function (refer to image 01) from the scipy.fft module is used to compute the 2-dimensional FFT of the image. This transforms the image from the spatial domain to the frequency domain, also known as k-space in the context of MRI.

Inverse Fast Fourier Transform (IFFT):

The ifft2 function (refer to image 01) is used to compute the 2-dimensional inverse FFT. This transforms the image back from the frequency domain to the spatial domain.

FFT Shift:

The fftshift function (refer to image 01) is used to shift the zero-frequency component to the center of the spectrum. This is often done for visualization purposes, as it makes the spectrum easier to interpret.

Inverse FFT Shift:

The ifftshift function (refer to image 01) undoes the shift applied by fftshift, returning the zero-frequency component to the corners of the spectrum.

Image 03

```python
def apply_k_space_truncation(image_array, truncation_radius):
    k_space = fftshift(fft2(image_array))
    center_x, center_y = np.array(k_space.shape) // 2
    y, x = np.ogrid[:k_space.shape[0], :k_space.shape[1]]
    mask = (x - center_x) ** 2 + (y - center_y) ** 2 <= truncation_radius ** 2
    k_space[~mask] = 0
    truncated_image = ifft2(ifftshift(k_space)).real
    return np.clip(truncated_image, 0, 255).astype(np.uint8)
```

<u>K-space Truncation:</u>

The apply_k_space_truncation function(image 03) applies a <u>circular mask</u> to the k-space representation of the image, setting all frequencies outside the truncation radius to zero. This effectively removes high-frequency components from the image.

Image 04

```python
def downsample_via_kspace_truncation(image, downsampling_factor, blur_sigma=1):
    # Apply Gaussian blur for noise reduction
    blurred_image = image.filter(ImageFilter.GaussianBlur(blur_sigma))

    # Convert to numpy array
    blurred_image_array = np.array(blurred_image)

    # Calculate the truncation radius
    k_space = fftshift(fft2(blurred_image_array))
    truncation_radius = find_truncation_radius(k_space)

    # Perform k-space truncation
    k_space = fftshift(fft2(blurred_image_array))
    center_x, center_y = np.array(k_space.shape) // 2
    y, x = np.ogrid[:k_space.shape[0], :k_space.shape[1]]
    mask = (x - center_x) ** 2 + (y - center_y) ** 2 <= truncation_radius ** 2
    k_space[~mask] = 0
    truncated_image = ifft2(ifftshift(k_space)).real
    # Resize (downsample) the image
    downsampled_image = Image.fromarray(np.clip(truncated_image, 0, 255).astype(np.uint8))
    downsampled_image = downsampled_image.resize(
        (blurred_image.width // downsampling_factor, blurred_image.height // downsampling_factor),
        Image.LANCZOS)
    return downsampled_image
```

<u>Image Downsampling:</u>

The downsample_via_kspace_truncation function downsamples the image by first applying a Gaussian blur, then performing k-space truncation, and finally resizing the image using the Image.resize function from the PIL module.

The overall algorithm implemented by this code is a form of image downsampling that preserves the most important (i.e., energy-dominant) features of the image while reducing its size. This is achieved by removing high-frequency components in the frequency domain before resizing the image in the spatial domain.

Lastly, there are few algorithms relating to our ***Aim #3*** in order to achieve the image processing and noise generation in this part.

Image 05

<u>Gaussian Noise Generation:</u>

```python
gaussNoise_Calculator(mean_value,std_deviation_value,imgXRowSize,
imgYRowSize):
    # Helper function: Calculating noise

gauss_noise=np.zeros((imgXRowSize,imgYRowSize),dtype=np.uint8) #
```

The gaussNoise_Calculator function (refer to image 05) generates Gaussian noise with a specified mean and standard deviation. This is done using the cv2.randn function, which fills an array with random numbers drawn from a normal distribution.
Image Addition:
The cv2.add function is used to add the generated Gaussian noise to the original image. This effectively applies the noise to the image.
Image Display:
The matplotlib.pyplot.imshow function (refer to image 01) is used to display the original image, the Gaussian noise, and the noisy image. The images are displayed in grayscale, as indicated by the ‘gray’ colormap.

The overall algorithm implemented by this code is a form of image noise generation. It generates Gaussian noise and applies it to an image, then displays the original image, the noise, and the noisy image.

Equations

In order for ***Aim #4***’s algorithms,refer to be executed properly they had to follow the mathematical equations.
CLAHE (Contrast Limited Adaptive Histogram Equalization):
This method improves the contrast of an image by applying histogram equalization in small, disjoint regions of the image. It limits the amplification by clipping the histogram at a predefined value, reducing the impact of noise. The transformation function is proportional to the cumulative distribution function (CDF) of pixel values in the neighborhood.
VerticalFlip and HorizontalFlip:
These transformations flip the image vertically and horizontally respectively. The flipping operation can be represented mathematically as a reflection across an axis.
RandomRotate90:
This transformation rotates the image by a random multiple of 90 degrees. The rotation of an image can be represented using rotation matrices in linear algebra.
Transpose:
This transformation flips the image over its main diagonal, effectively rotating the image 90 degrees counterclockwise and then flipping it horizontally. This can be represented mathematically by switching the row and column indices of the matrix representing the image.
GridDistortion:
This transformation distorts the image following a grid pattern. The exact mathematical representation can vary, but it generally involves moving pixels based on a grid of control points.

Please note that these are high-level explanations and the actual implementation in the albumentations library may involve additional steps and optimizations. Also, the exact mathematical equations can vary based on the specific parameters provided to each transformation function.
— — — — — — — —
GUI: code is a simple graphical user interface (GUI) application using the Tkinter library in Python. It doesn't implement any specific mathematical equations, but it does use several functions and methods from the Tkinter library to create and manage the GUI. Here's a brief description of each:
Integer Validation:
The validate_input function checks if the input character is a digit. This is a simple boolean operation.
Button Click Action:
The on_button_click function is called when the button is clicked. It retrieves the values from the Entry widgets, adds them together, and displays the result in a message box. If the input values are not valid integers, it shows an error message. The addition operation can be represented mathematically as result=value1+value2
Tkinter Event Loop:
The root.mainloop() method starts the main event loop. It's necessary for handling events such as button clicks and user input.

The tasks mentioned in the comments (generating a phantom, data acquisition and reconstruction) are not implemented in the provided code. These tasks would likely involve more complex algorithms and additional libraries, especially for medical imaging tasks. For example, generating a phantom could involve algorithms for simulating the response of human tissue to different imaging modalities, while data acquisition and reconstruction could involve algorithms for processing raw imaging data into a format that can be visualized and analyzed. However, these specific algorithms are not present in the provided code.
— — — — — — — —
K-Space Truncation:

Energy Distribution Calculation:
The find_truncation_radius function calculates the radius in k-space that contains a certain percentage of the total energy of the image. This is done by sorting the frequencies by their distance from the center of the spectrum, calculating the cumulative sum of their magnitudes, and finding the radius at which the cumulative sum exceeds the energy threshold.
performs several operations on an image, each of which is based on a specific mathematical concept or equation. Here's a breakdown:
Fast Fourier Transform (FFT):

The fft2 function computes the 2-dimensional discrete Fourier Transform. This is a way to represent a complex-valued image in the frequency domain. The equation for a 2D FFT X of an image x of size MxN is given by:
X(u,v)=m=0∑M−1n=0∑N−1x(m,n)⋅e−j2π(Mum+Nvn)

$$X(u,v) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x(m,n) \cdot e^{-j2\pi\left(\frac{um}{M} + \frac{vn}{N}\right)}$$

where j is the imaginary unit12.

Inverse Fast Fourier Transform (IFFT):

The ifft2 function computes the 2-dimensional inverse discrete Fourier Transform, converting the image back from the frequency domain to the spatial domain34. The equation for a 2D IFFT x of an image X of size MxN is given by:
x(m,n)=MN1u=0∑M−1v=0∑N−1X(u,v)⋅ej2π(Mum+Nvn)

$$x(m,n) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} X(u,v) \cdot e^{j2\pi\left(\frac{um}{M} + \frac{vn}{N}\right)}$$

FFT Shift:

The fftshift function shifts the zero-frequency component to the center of the spectrum567.

Inverse FFT Shift:

The ifftshift function undoes the shift done by fftshift, shifting the zero-frequency component back to the corners of the spectrum89.

Energy Distribution Calculation:

The energy of the image in k-space (frequency domain) is calculated as the sum of the absolute values of the k-space image. The energy distribution is then calculated by sorting these values and taking their cumulative sum10.

K-space Truncation:

This is a process where all frequencies in the k-space image that are beyond a certain radius (the truncation radius) from the center are set to zero.

Gaussian Blur:

The ImageFilter.GaussianBlur function applies a Gaussian blur to the image. This is done by convolving the image with a Gaussian kernel.

Image Downsampling:

The resize function is used to downsample the image. This involves reducing the number of pixels in the image, effectively reducing its size.

Each of these operations is based on mathematical equations and concepts from signal processing and image analysis. The overall goal of the code is to apply a Gaussian blur to the image, truncate its k-space representation, and then downsample it. This process can help to reduce noise and unnecessary details in the image while preserving its main features. The downsampling process reduces the image size, which can be useful for reducing the computational resources needed for further image processing or analysis. The specific parameters for the Gaussian blur and the downsampling factor can be adjusted to achieve the desired level of detail and image size.

— — — — — — — —

Gaussian noise generation:

The gaussNoise_Calculator function generates Gaussian noise with a specified mean and standard deviation. This is done using the cv2.randn function, which fills an array with random numbers drawn from a normal distribution. The mathematical formula for generating Gaussian noise is given by:

$X=\mu+\sigma \cdot \text{randn}$

where X is the generated noise, mu is the mean, sigma is the standard deviation, and randn is a random number drawn from a standard normal distribution1.

Image Addition:

The cv2.add function is used to add the generated Gaussian noise to the original image. This effectively applies the noise to the image. The mathematical operation for image addition is simply adding the corresponding pixel values of the two images2.

Image Display:

The matplotlib.pyplot.imshow function is used to display the original image, the Gaussian noise, and the noisy image. The images are displayed in grayscale, as indicated by the 'gray' colormap.

Image Reading:

The cv2.imread function is used to read the image file. The '0' argument indicates that the image should be read in grayscale.

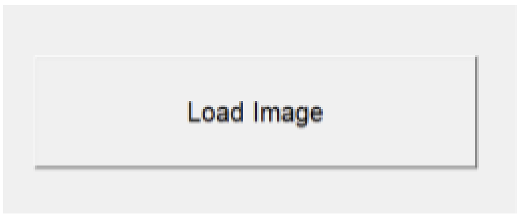
The overall algorithm implemented by this code is a form of image noise generation. It generates Gaussian noise and applies it to an image, then displays the original image, the noise, and the noisy image. This can be useful for testing image processing algorithms, as it allows you to see how they perform under noisy conditions. The specific parameters for the Gaussian noise (mean and standard deviation) can be adjusted as needed.

— — — — — — — —

Results and Discussion

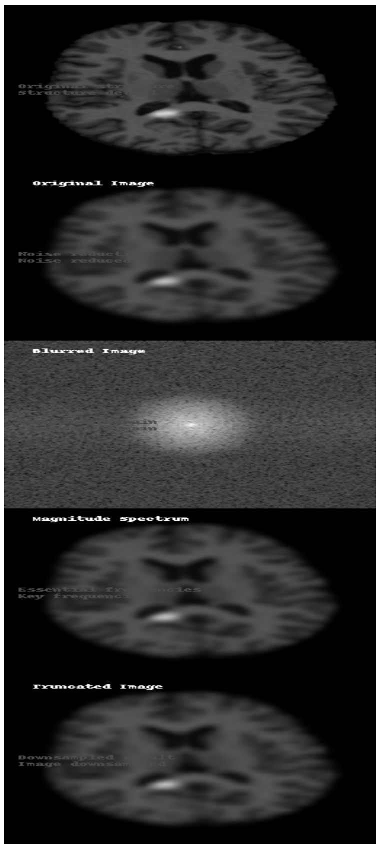
MRI Acquisition (Aim 1)

We simulate scanning an MRI by loading an image into the simulator.



K Space Truncation (Aim 2)

The primary goal of this method is to intricately downsample images by manipulating their frequency elements, thereby preserving a substantial amount of their energy via a carefully determined truncation radius within the frequency domain. This strategy elevates the process beyond conventional spatial-domain resizing techniques, placing a strong emphasis on maintaining the high quality and integrity of images even after they have been reduced in size. This approach not only ensures efficient downsizing but also safeguards the rich details and essential characteristics of the original images. The figure visualizes the assumptions made to meet the desired output.



Step 1: Image Preparation:

Gaussian blur is applied to the image using the ImageFilter.GaussianBlur function from the PIL library to reduce noise. This is done in the downsample_via_kspace_truncation function.

Step 2: Fourier Transform:

The Fourier Transform is applied to the blurred image using fft2 and fftshift functions from the scipy.fft library. This transformation occurs in both the calculation of the k_space variable in the downsample_via_kspace_truncation function and the subsequent calculation of k_space again.

Step 3: Frequency Selection:

The find_truncation_radius function calculates the truncation radius based on the energy distribution in the k-space, which is essential for selecting the significant frequencies.

Step 4: Truncation and Inverse Transform:

The apply_k_space_truncation function applies the truncation by creating a binary mask and setting frequencies outside the truncation radius to zero. It then uses inverse Fourier Transform to bring the image back to the spatial domain. This step involves using fft2, fftshift, ifft2, and ifftshift functions.

Step 5: Resizing:

The final resizing (downsampling) of the image is performed using the Image.resize method with Lanczos resampling to adjust the resolution. This resizing occurs in the downsample_via_kspace_truncation function.

Mathematical Rationale Behind the Simulation

Gaussian Blur: We apply this as a pre-processing step based on the convolution theorem, which in layman's terms, states that blurring in the spatial domain corresponds to simple multiplication in the frequency domain. It prepares the image for a cleaner transformation and downsampling.

Fourier Transform: We use this to transition into the frequency domain, enabling us to analyze and manipulate the image based on its frequency content. It's a reversible transformation, meaning we can go back and forth between domains without losing information.

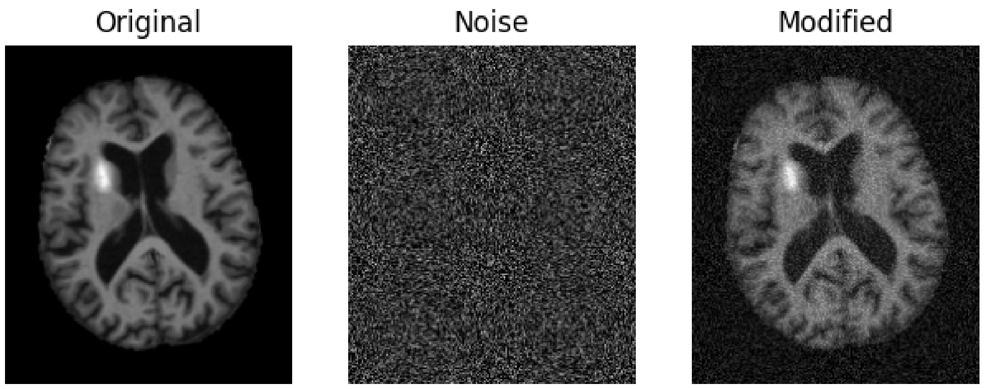
Energy Concentration: The rationale for focusing on areas with a high concentration of energy in k-space is rooted in the signal processing principle that the essential characteristics of an image are often contained within these frequencies.

Truncation and Masking: The decision to truncate is based on the premise that high-frequency components often represent noise or details that can be sacrificed when downsampling, allowing for a smaller image size without significantly impacting the visual content.

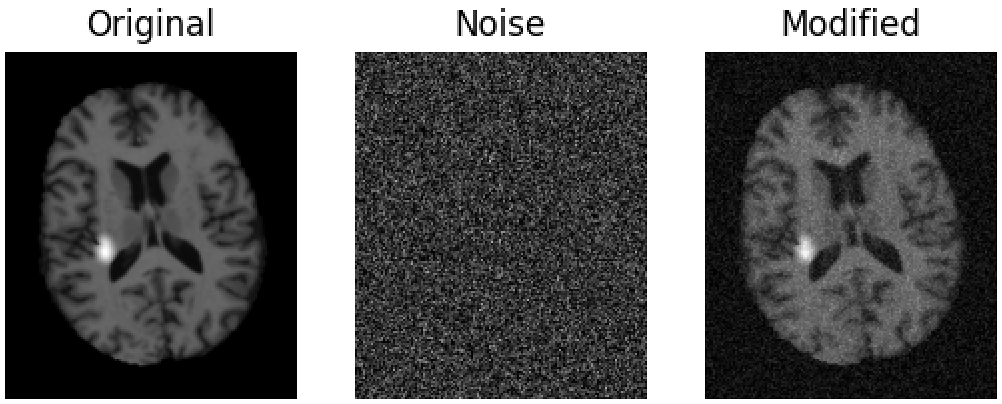
Gaussian Noise Addition (Aim 3)

Gaussian noise is a type of statistical noise characterized by its probability distribution, which follows the Gaussian or normal distribution. This is useful for our simulator because it allows us to simulate the inherent randomness and imperfections encountered during the image acquisition process.

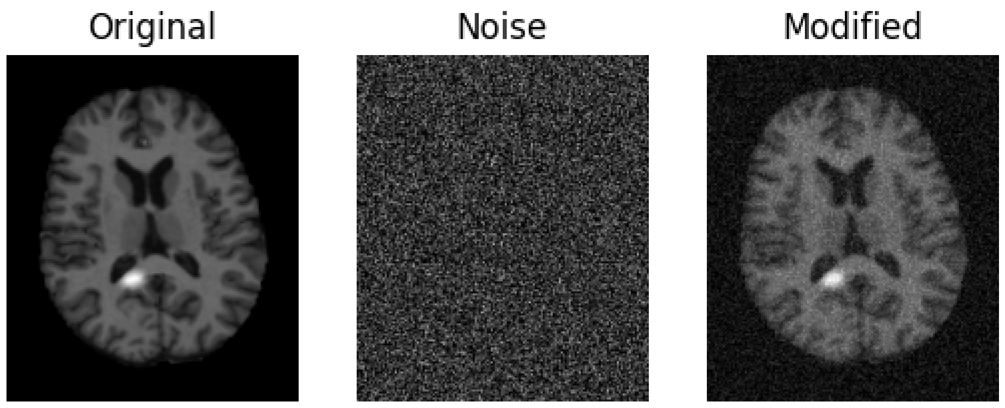
Gaussian Noise Image 1



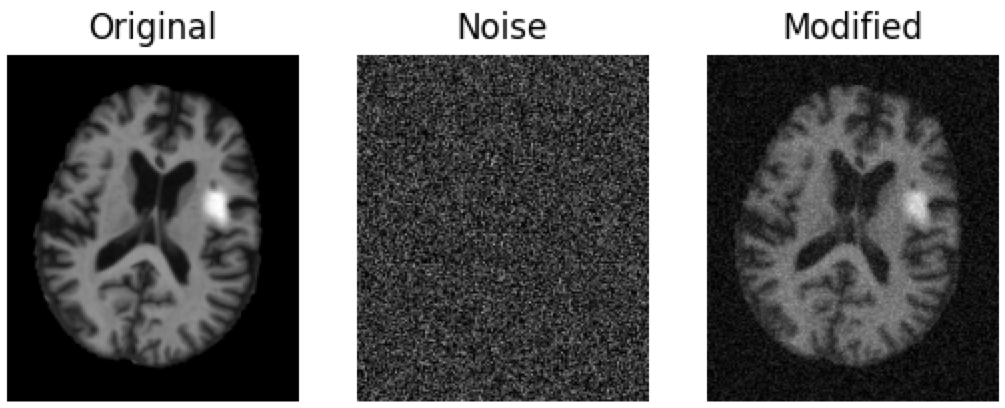
Gaussian Noise Image 2



Gaussian Noise Image 3

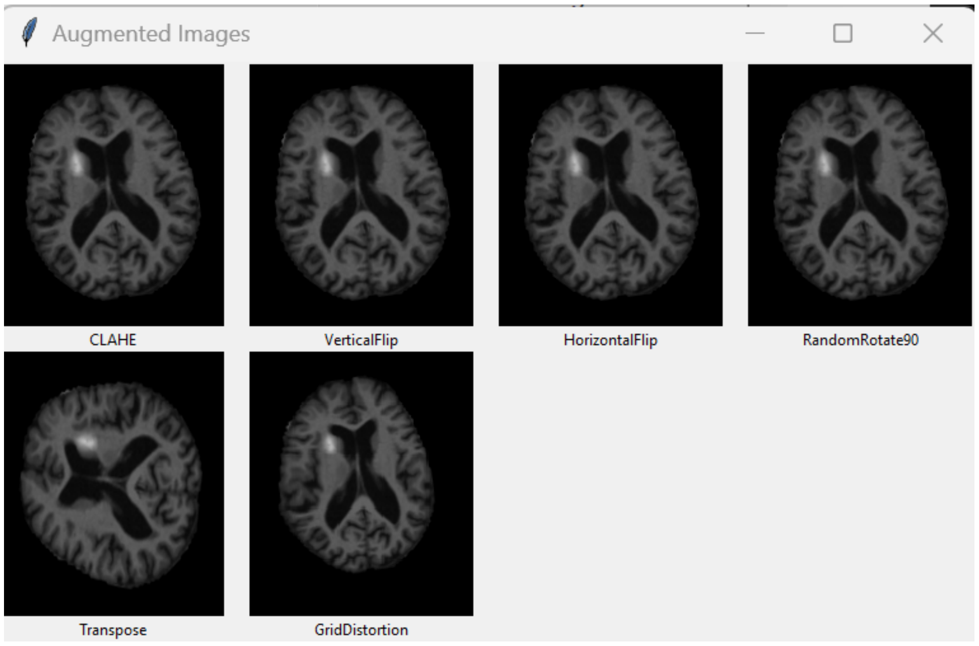


Gaussian Noise Image 4



The following images show step by step result of developing a modified image with noise.
We generate a random gaussian noise, and then add that noise to the original image. The modified image is our final result. We can see under “ Modified” how adding noise to the original image creates a lower quality sample and achieves our aim for simulating real world image acquisition.

Image Augmentation (Aim 4)

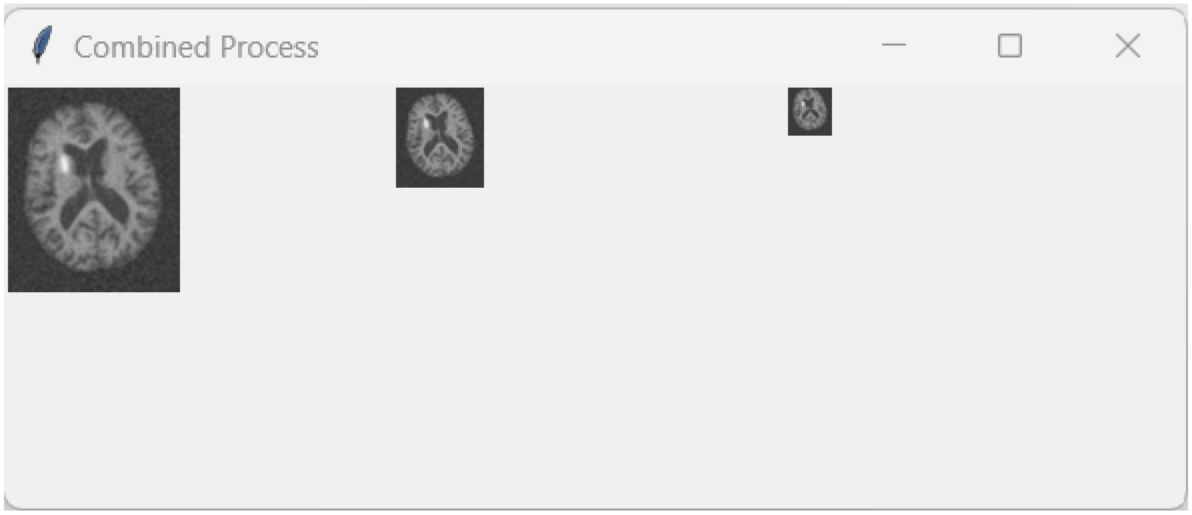


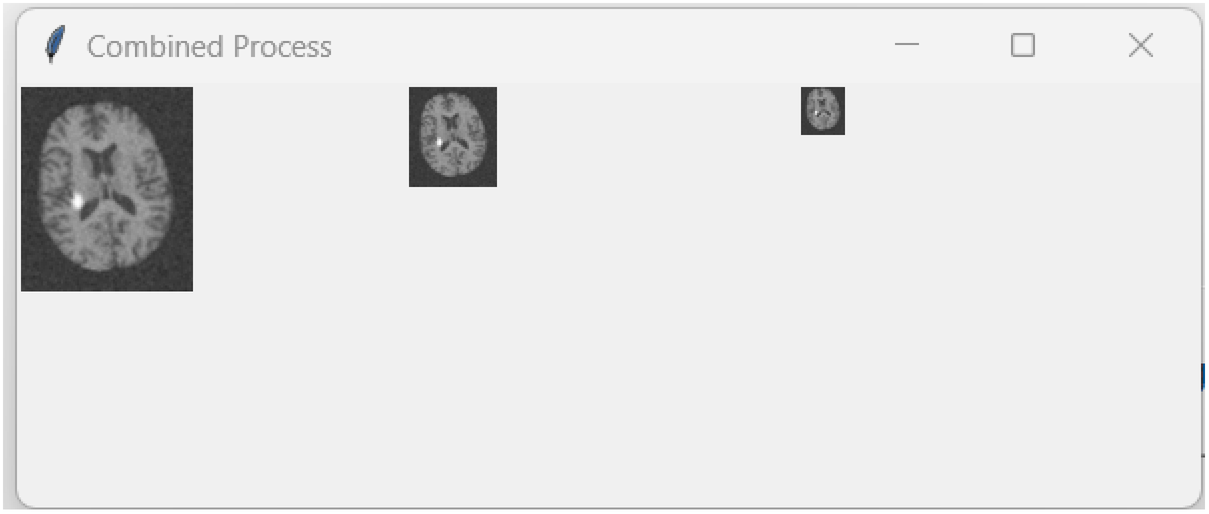
In order to generate the augmentation images, the image is loaded using the `cv2.imread` function and then converted from BGR to RGB color space using `cv2.cvtColor`. Once an image is received a list of transformations is defined, each represented by a tuple containing the name of the transformation and an instance of the corresponding `albumentations` class. For each transformation in the list, the transformation is applied to the image, and the result is added to the `augmented_images` list. After the results are obtained then we move on into displaying the results. This is done through the `display_augmented_images` function. This function creates a new window, calculates the size of the canvas based on the number of images and the spacing between them, and then creates a `tkinter.Canvas` widget of the calculated size. It then loops over the `augmented_images` list, converts each augmented image to a `tkinter`-compatible format.

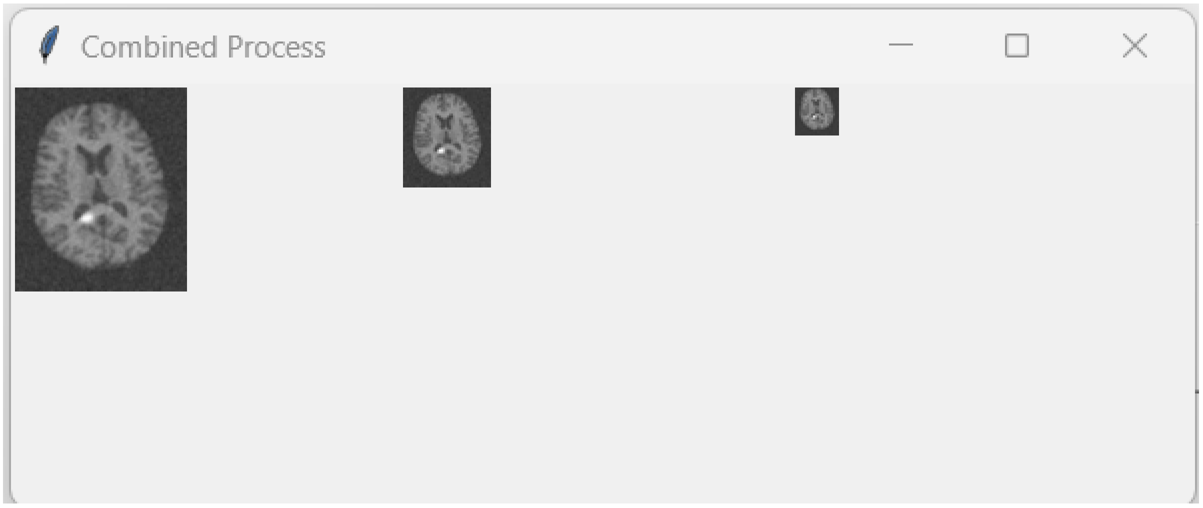
The results reflect an image being subjected to different transformations. This can be useful in machine learning, where augmented images are used to increase the size of the training set, thereby improving the model's ability to generalize from the training data to unseen data. The transformations used in this code include contrast adjustment (CLAHE), flipping, rotation, transposition, and distortion, all of which can help the model learn to recognize the relevant features under a variety of conditions. The displayed images give you a visual understanding of what each transformation does to the image.

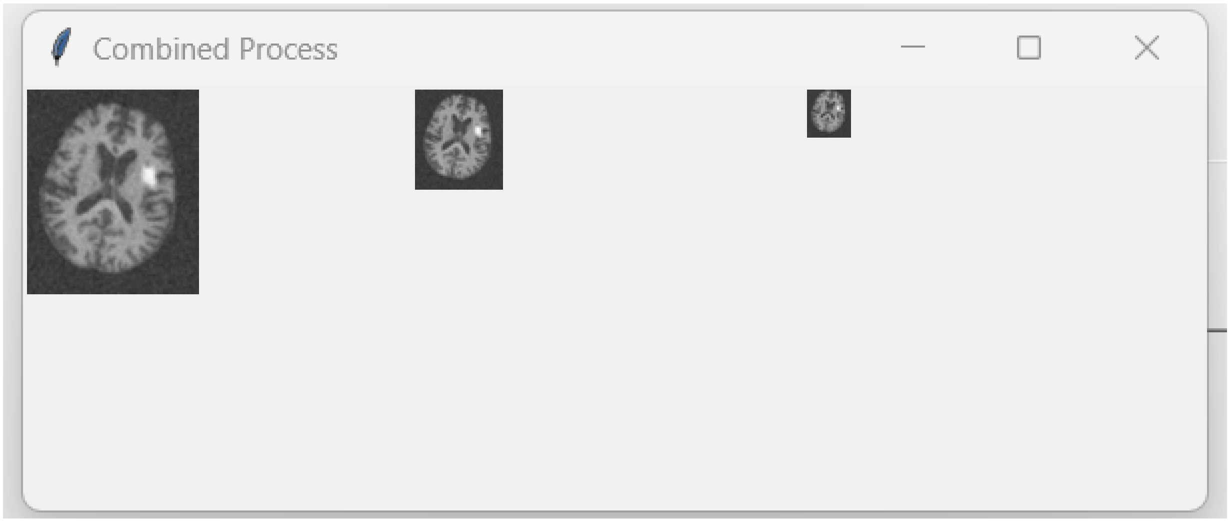
Final Processed Images (Aim 5)

The following images are the final processed groundtruth images that have been modified with both gaussian noise and k-space truncation.



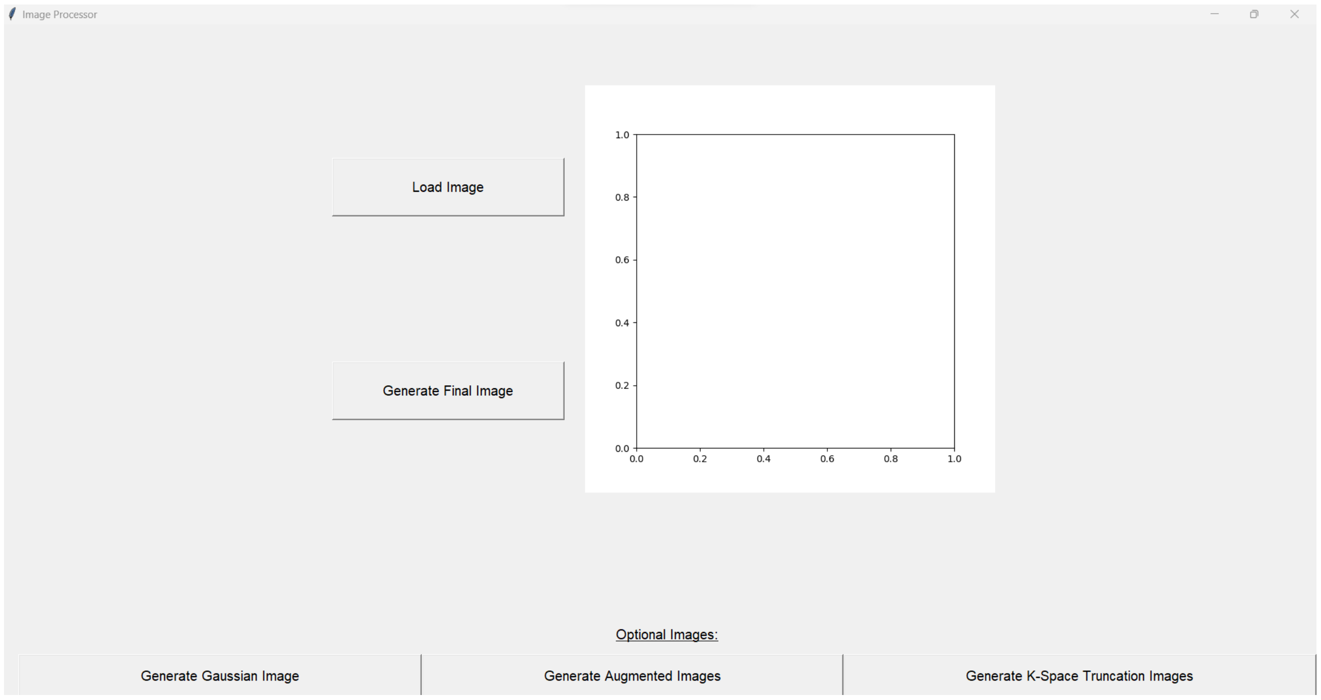






Graphical User Interface (Aim 6)

The Graphical User Interface (GUI) enables for k-space truncation, augmented images, and gaussian filtering using just one module, which the user uploads the desired image to. A separate window will open for each processed image.



Load Image Button:
For the user to add the desired image to the graph, the user will click the load image button, which opens the users file explorer.

Generate Final Image:
In this module that is located to the left of the graph and under the load image button we are allowing the user to apply the gaussian filter and k-space truncation in one step.

<u>Optional Process:</u>
Generate Gaussian Image:
The generate gaussian image button, which is aligned at the bottom of the GUI on the far left of the window, will generate a gaussian image. The user must load the image into the graph first before using the button.

Generate Augmented Image:
The generate augmented image button, which is aligned at the bottom of the GUI in the middle of the window, will generate a gaussian image. The user must load the image into the graph first before using the button.

Generate K-space Truncation Image:
The generate k-space truncation image button, which is aligned at the bottom of the GUI far right of the window, will generate a gaussian image. The user must load the image into the graph first before using the button.

Walkthrough:

1. To navigate to the project file in your favorite IDE, open the IDE and look for a "File" menu at the top. Click on "File" and then select "Open" or "Open Project." Browse to the location where your project file is stored, select the file, and click "Open" to load the project into the IDE
2. The user can then enter "python main.py" in the terminal to launch the program.
3. For the user to load the photo into the interface the user must press the “Load Image” button once. After that, the user can find the picture they want to use in the graph.
4. The user can now select the process they want to carry out once the image has been imported into the graph. The Generate Final Image, Generate Gaussian Image, Generate Augmented Images, and Generate K-space Truncation Image buttons are available for selection by the user.

Conclusion

In this study, we developed a software to simulate MRI data acquisition, incorporating advanced techniques like K-Space Truncation, Gaussian Noise Addition, and Data Augmentation, emulating a virtual Magnetic Resonance Scanner. This multifaceted approach includes simulating MRI data collection, applying K-Space truncation to original images, adding Gaussian noise to mimic real-world imaging conditions, and generating augmented images for a comprehensive analysis. Additionally, we crafted a user-friendly Graphical User Interface (GUI) to facilitate the execution of these complex processes.

Appendix A

The Python code is attached. A list of the files is as follows:

Main.py: Code contains GUI interface, K space truncation function, gaussian addition function, and image augmentation function. Run This file to start the program