Gamma Camera SPECT Uniformity

[redacted] Project

8th January - 22nd April, 2024

Word count: 3800 (excl. appendices)

Abstract: Gamma cameras are imaging tools used in hospitals to view physiological processes by imaging the gamma radiation emitted by radio-pharmaceuticals. As a part of quality control measures to ensure the functionality of gamma cameras, uniformity tests need to be completed to ascertain whether the devices are functioning correctly. This involves imaging a uniform body of radiation, also called a test phantom, and checking that the DICOM images produced by a gamma camera show a similarly uniform image. Traditionally, medical professionals have inspected DICOM images and measured the uniformity across limited parts of the image. We have aimed to create a piece of software as a proof of concept that allows uniformity to be calculated across an entire DICOM image at once, that has the potential to increase the efficiency and reliability of quality control tests for gamma cameras.

Lyall Stewart
Leah Wells
Zac Baker
Alex Toogood-Johnson
Josh Scates
[redacted] **School**

[redacted]

Contents

1 Introduction and Background

The RD&E Nuclear Medicine Department currently performs various types of quality control tests on Gamma cameras [3]. One of these tests involves calculating a measure of uniformity over a tomographically acquired SPECT image. Our task was to write a piece of software that allows a user to find the differential and integral uniformity of a DICOM image, by accepting an image from the user, applying a 9-point weighted convolution filter to smooth out the pixels, and using the formulae specified in the NEMA handbook provided to us to obtain the uniformity percentage values.

1.1 Introduction to Nuclear Medicine

1.1.1 What is Nuclear Medicine?

Nuclear medicine is a specialised field of medicine that uses radioactive tracers known as radio-pharmaceuticals to assess bodily functions and to diagnose and treat disease." Approved tracers are called radio-pharmaceuticals since they must meet the FDA's exacting standards for safety and appropriate performance for the approved clinical use" [3]. Some commonly used tracers used in nuclear medicine are Technetium-99m, chosen due to its short half-life of 6 hours, and Iodine-131 [2].

1.1.2 What is a Gamma Camera?

Gamma cameras are imaging tools used in hospitals to view physiological processes by imaging gamma radiation emitted by radio-pharmaceuticals [4]. A gamma camera produces SPECT images (see Section 1.2) from the gamma ray emissions, and these can be analysed to diagnose a condition. For example, a high radiation count in a specific area of the body, when the radio-pharmaceuticals have been attached to substances such as octreotide or dotatate, can be indicative of the presence of neuroendocrine tumours [15].

Figure 1: A modern duel-head gamma camera [9]

1.1.3 How Does a Gamma Camera Work?

A radio-pharmaceutical such as Technetium-99m can be attached to a drug so that it will target a specific area of the body that needs to be imaged [2]. As mentioned in section 1.1.2, drugs such as octreotide and dotatate can be attached to these radio-pharmaceuticals so that the radioactive sources are drawn to tumour cells, although many other possibilities exist [15].

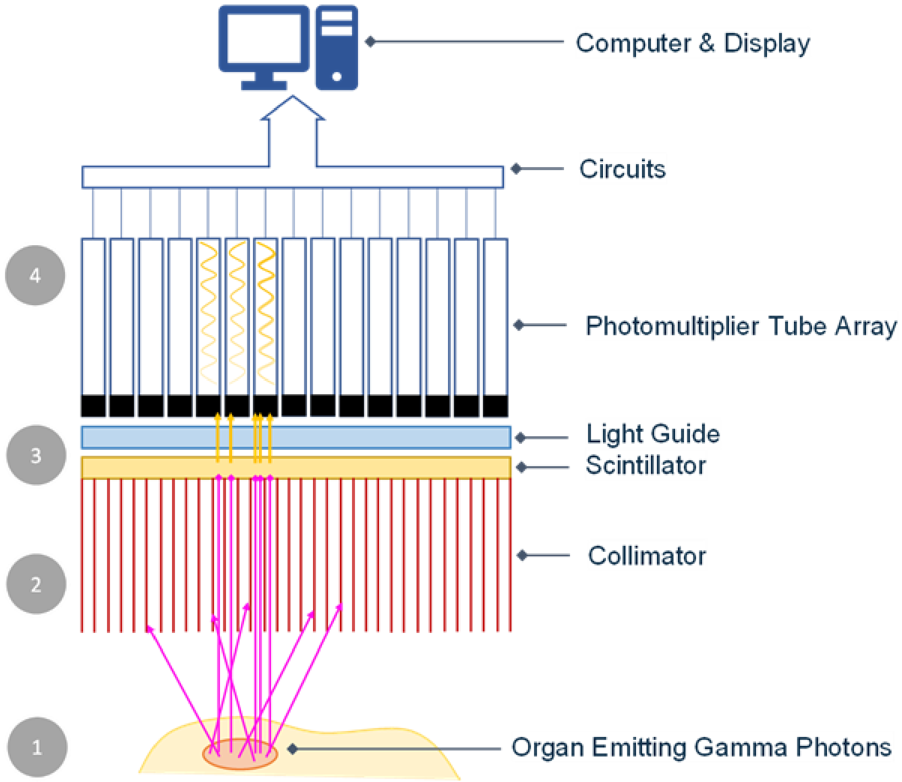

Figure 2: Gamma camera diagram [16]

Gamma cameras detect high-energy photons released from the radioactive atoms in a patient's body. These photons escape from the patient's body from all directions and are passed through a collimator. This ensures that only photons travelling vertically upwards will be detected, which reduces noise and measures with more accuracy the density of the radiation emitted from different areas. The photons then hit a scintillation crystal, which converts them into light. This light then has to be amplified in a photo-multiplier tube array and processed by a computer to produce a 2D image of the distribution of photons. The camera takes these images from 360°, so they can be layered to form a 3D image of the organ or process you are interested in [4]. The areas that you are interested in will have a higher concentration of photons.

1.2 SPECT

A SPECT scan is a type of imaging that uses a radioactive substance and a detector to produce 3D images to show physiological processes. A gamma camera produces SPECT scans [9]. When multiple 2D projections are acquired of an object from multiple angles, mathematical tools can then be used to reconstruct a 3D image of that object.

When using a gamma camera, the image reconstruction is created from planar (2D) images from every angle, which shows the number of gamma rays detected [10].

1.3 Quality Control

Quality control means ensuring that the level of performance of a system remains at a stable operating level within the most recent performance limits. A good quality control system ensures that the people closest to the work have the knowledge, tools, and procedures in place to review, reflect, and react, with immediacy, to discrepancies between observed and agreed levels of performance [6]. This is necessary because an error in the system could lead to the misdiagnosis of patients. In the case of gamma cameras, it could also lead to increased exposure to radiation if the test must be redone.

1.4 Uniformity

The uniformity of a gamma camera is normally defined as the difference between the maximum and minimum counts per field element (or count density) expressed as a percentage of the mean number of counts per element or of the average of the maximum and minimum values [8]. For example, the uniformity of a gamma camera is often tested using a cylinder of radiation, if the gamma camera is working perfectly this will produce one section of equal maximum density surrounded by equal minimum density; in reality this is not the case, so uniformity tests must be used to confirm that the variance in count density is due to background noise and not a fault with the gamma camera. There are two types of uniformity: differential uniformity and integral uniformity. If the value of differential uniformity exceeds 4% on a daily test, then it should be retested and considered as potentially being faulty [12].

$$\text{Integral uniformity} = \pm 100 \times \frac{(max - min)}{(max + min)} \tag{1}$$

From the NEMA handbook given to us for this project: ”For pixels within each area (CFOV and UFOV), the maximum and minimum values are identified from the smoothed data. The difference between the maximum and the minimum is divided by the sum of these two values, and multiplied by 100.” Here, CFOV and UFOV stand for Central Field Of View and Useful Field Of View. As we have cropped each DICOM image to a specific width, length, and height, we do not need to regard the fields of view, and can instead focus on values for the whole image. Thus, to find the integral uniformity of a DICOM image, we just smooth it with a convolution filter, crop it to the preferred size, and substitute the maximum and minimum pixel values across the whole image into the formula above.

$$\text{Differential uniformity} = \pm 100 \times \frac{(max - min)}{(max + min)} \tag{2}$$

From looking at the equation, it may seem that differential uniformity is the same as integral uniformity, however, the main difference here is which values are chosen as min and max to substitute into the equation. From the NEMA handbook: ”The smoothed data are treated as a number of rows (X slices) and columns (Y slices) Each slice is processed by starting at the beginning pixel for the respective FOV. A set of 5 contiguous pixels is examined to find the minimum and maximum pixels. The differential uniformity is calculated using these values. The next set of 5 pixels is analysed by stepping forward one pixel and again determining the percent uniformity. This is repeated until the outermost pixel is reached. The maximum differential uniformity is found in the slice. This process is then repeated for all the slices, and the maximum value for all slices is reported.” To summarise, each DICOM image is made of slices, e.g. a cropped, smoothed 8x8x8 DICOM image has 8 layers, each of which are 8 pixels wide by 8 pixels long.

7	8	4	9	6	3	10	9
5	4	6	2	3	7	6	7
5	7	3	1	5	4	2	3
3	2	9	9	6	7	8	6
2	3	4	3	10	11	9	6
7	2	3	5	6	2	3	4
1	4	1	2	7	5	4	5
5	4	6	2	3	7	6	7

For each of those slices in the image, take a set of 5 contiguous values, e.g.:

7	8	4	9	6

We can find the values of max and min from these values, being 9 and 4, and substitute these into the equation for differential uniformity to get a uniformity value of 38.46%, to 2 decimal places. Let us step forward one pixel and consider the values:

8	4	9	6	3

Here, we can take the values for max and min as 9 and 3, and get a differential uniformity of 50%. We would then repeat this for every row and column in the layer, 5 pixels at a time, to get a uniformity value for that layer (this being the maximum value for any 5 pixels in that layer). The maximum value out of all of the layers in the image is taken as the differential uniformity for that image. Due to each calculation requiring 4 additional pixels to the right or down, this leaves a 5 pixel thick band around the left and bottom of the circle in each layer, which cannot be calculated with and are ignored.

2 Methods

2.1 DICOM files

At first, we decided to use the Python package *pydicom* in order to read DICOM files, however, after that we decided to instead research DICOM files and the DICOM specification to try to manually parse the binary of a DICOM file in order to get the metadata and image data. We eventually succeeded in doing this to the most extent, although there are still a few flaws in the code.

2.1.1 The DICOM File Format

The results of gamma camera scans, as standard in medical imaging, are stored in the Digital Communications and Information in Medicine (DICOM) file format, allowing cross-compatibility with a number of different software tools. While there are many open-source, existing tools to parse these, which we did use at the start of the project, we later decided to do this step ourselves.

We researched the structure of DICOM files and found out that there are 2 main parts: the header and the dataset. The header is made of a preamble and a prefix. Firstly, the preamble is a sequence of 128 bytes, which are usually all set to '00H'. The prefix is a sequence of 4 ASCII-encoded bytes, which stand for the letters 'D', 'I', 'C' and 'M'. Thus, we ignore the first 132 bytes of any DICOM file [10].

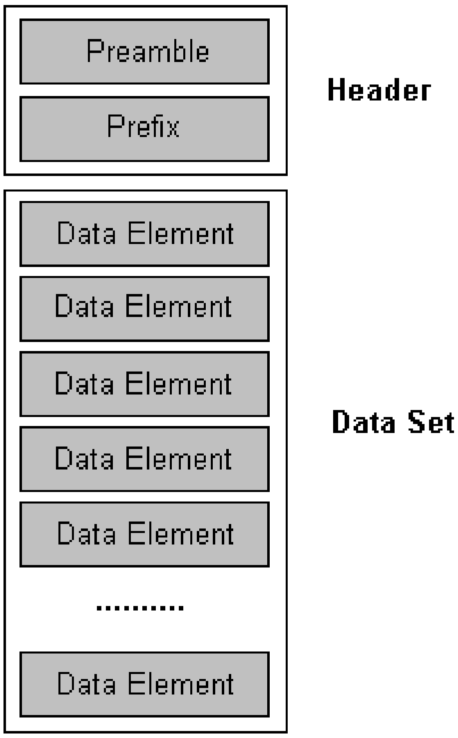

Figure 3: Structure of a DICOM file [10]

The dataset is comprised of repeating data elements, which are split further into either metadata or image data elements. Each element has 5 parts : the group ID, the element ID, the VR (Value Representation), the group length, and the element data. The group ID, the element ID, and the VR are always 2 bytes each. The size of the value length field is determined by the which VR is used, and the value of the element data field is determined by the data length field. There are 34 different VRs, each of which are a 2-byte ASCII-encoded string. Each VR holds a different type of data, for example the VR 'UL' is an unsigned 32-bit binary integer. The full list of VRs can be found in the DICOM specification [6].

Group Number	Element Number	Value Representation	Value Length	Value Field
2 bytes	2 bytes	2 bytes	2 bytes	"Value Length" bytes

Figure 4: Structure of a DICOM element [11]

Each group ID and element ID pairing, which take the form of 2 4-digit numbers e.g. (0100,0010), correspond to a specific tag, the full list of which, can be found from pages 23 to 183 of the registry of DICOM data elements [8]. In the image above, the VR is neither OB, OW, SQ or UN, so the value length is 2 bytes, however, if the VR is one of those 4, then a 4 byte value length shall be preceded by 2 reserved bytes, which can be ignored when we parse the file, meaning 6 bytes are used instead of 2. One of the challenges we faced when manually decoding the binary was when we encountered one of these 4 VRs. SQ, which stands for Sequence, has multiple data elements contained within, and we found this very difficult to parse.

After the metadata elements, there is the image data element. This stores the pixel data for the entire DICOM image, and we found in the DICOM files we used that this had a tag beginning '7FE0' [7]. We initially had trouble decoding this, but upon looking more closely at the DICOM specification, we eventually figured out how to convert the binary into a Python *numpy.ndarray* representing the image.

2.1.2 Obtaining Metadata

Our main success with manually parsing the DICOM files was obtaining most of the metadata for a DICOM file, as we could get the group ID, element ID, VR, length and data field for almost all of the metadata elements. We also web-scraped the registry of DICOM data elements into a JSON file, so that we could convert the group-element pair into a tag [5]. Comparing our result with the metadata output of the *pydicom* package, we found out we were almost completely accurate, with our only issue being that our code couldn't decode any SQ (Sequence) elements.

```
File Meta Information Group Length      02000000   212
File Meta Information Version           02000100   b'\x00\x01'
Media Storage SOP Class UID             02000200   1.2.840.10008.5.1.4.1.1.20
Media Storage SOP Instance UID          02000300   1.2.840.113619.2.184.31108.192168116
Transfer Syntax UID                     02001000   1.2.840.10008.1.2.1
Implementation Class UID                02001200   1.2.840.113619.6.281
Implementation Version Name             02001300   Xeleris 3.1108
Source Application Entity Title         02001600   XELERIS-R
Group Length                            00080000   544
Image Type                              00080008   DERIVED\PRIMARY\RECON TOMO\EMISSION
Instance Creation Date                  00080012   20231124
Instance Creation Time                  00080013   152143.0000
Instance Creator UID                    00080014   1.2.840.113619.6.281
SOP Class UID                           00080016   1.2.840.10008.5.1.4.1.1.20
SOP Instance UID                        00080018   1.2.840.113619.2.184.31108.192168116
Study Date                              00080020   20230728
Series Date                             00080021   20231124
Acquisition Date                        00080022   20230728
Content Date                            00080023   20231124
Study Time                              00080030   103856.00
Series Time                             00080031   152054.00
Acquisition Time                        00080032   103856.00
Content Time                            00080033   152054.00
Modality                                00080060   OT
Manufacturer                            00080070   GE MEDICAL SYSTEMS
Institution Name                        00080080   Royal Devon and Exeter Hospital
Station Name                            00081010   XELERIS-R
Study Description                       00081030   User&Basic&Tomo
Series Description                      0008103e   Volumetrix MI RESULTS AC
Manufacturer's Model Name               00081090   INFINIA
Private Creator                         00090000   470
Private Tag Data                        00090010   GEMS_GENIE_1
UNKNOWN                                 00091001   GEMS_GENIE
Study Name                              00091010   Tomo
Study Flags                             00091011   17825792
Study Type                              00091012   0
Dataset UID                             0009101e   1.2.840.113619.2.184.31108.192168116
Series Object Name                      00091020   Tomo
Series Flags                            00091021   16777216
Initiation Type                         00091023   0
```

Figure 5: Part of the metadata obtained through manually parsing the binary

```
Group Length: 4194316
Image Type: ['DERIVED', 'PRIMARY', 'RECON TOMO', 'EMISSION']
Instance Creation Date: 20231124
Instance Creation Time: 152143.0000
Instance Creator UID: 1.2.840.113619.6.281
SOP Class UID: 1.2.840.10008.5.1.4.1.1.20
SOP Instance UID: 1.2.840.113619.2.184.31108.192168116118.1700839067.58997240
Study Date: 20230728
Series Date: 20231124
Acquisition Date: 20230728
Content Date: 20231124
Study Time: 103856.00
Series Time: 152054.00
Acquisition Time: 103856.00
Content Time: 152054.00
Accession Number:
Modality: OT
Manufacturer: GE MEDICAL SYSTEMS
Institution Name: Royal Devon and Exeter Hospital
Referring Physician's Name:
Station Name: XELERIS-R
Study Description: User&Basic&Tomo
Series Description: Volumetrix MI RESULTS AC
Name of Physician(s) Reading Study:
Operators' Name:
Manufacturer's Model Name: INFINIA
Private Creator: 1052
Private tag data: GEMS_GENIE_1
[Unknown]: [(0013, 0010) Private Creator                    LO: 'GEMS_GENIE_1'
(0013, 1014) [Original Image Num]                SL: Array of 128 elements]
[Study Name]: Tomo
[Study Flags]: 4097
[Study Type]: 0
[Dataset UID]: 1.2.840.113619.2.184.31108.192168116118.1700839067.58997240
[Series Object Name]: Tomo
[Series Flags]: 1
[User Orientation]:
[Initiation Type]: 0
[Initiation Delay]: 0
[Initiation Count Rate]: 0
```

Figure 6: Part of the metadata obtained via the *pydicom* package

Here in the images above and below, you can see a list of the metadata that you would find in a standard DICOM file. There are elements such as the acquisition date, the manufacturer of the gamma camera, and the name of the institution who owns the gamma camera. We had just over 5,200 entries in our JSON file, which meant our program was capable of identifying that many elements of metadata, however there are still a few unknown group-element ID pairings, which have just been left as 'UNKNOWN' in the metadata output.

Although our program did not end up using a majority of the metadata we obtained through reading the binary, we feel that coding this helped us gain a better understanding of the structure of DICOM files, and it helped us understand how to get the image data from the binary, which our program did use.

2.1.3 Obtaining Image Data

Once we had managed to obtain the metadata from the SPECT DICOM files that we were given, we found it easy to to gain the raw data for the image, that being a Python list some 4,194,304 bytes in length, however it then took us quite a long time to figure out how to convert this into an image, although in the end it turned out to be relatively easy.

The metadata with group (0028, 0100) contains the value for the bits allocated per pixel [7]. For all the DICOM files we tested, this turned out to be 16 bits, or 2 bytes per pixel. From here, the groups with ID (0028, 0008), (0028, 0010) and (0028, 0011) in the metadata contained the values for the number of rows, columns and planes in the SPECT image. From this point, we converted the Python list to a *Numpy* array, and then used the *Numpy .reshape()* function to convert the *Numpy* array into a 3D array with the same dimensions as the image.

2.2 What is a Convolution?

"A convolution is essentially a filter that can be applied to a set of pixels in order to emphasize or smooth a particular feature. A convolution uses a matrix called a kernel, and each pixel in the resultant image is a function of the nearby pixels, and the input image, with a kernel as the function" [5]. Mathematically :

$$g(x,y) = \omega(x,y) * f(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \omega(i,j) f(x-i, y-i)$$

Where $g(x,y)$ is the filtered image, $f(x,y)$ is the original image, and ω is the filter kernel [14]. For more information, we recommend watching the 3Blue1Brown YouTube video on convolutions [1]. Different kernels can create different effects, such as sharpening or blurring an image, but the kernel we were given is intended to smooth a layer of a DICOM image. In the NEMA handbook provided to us, we were given the following kernel to use.

1	2	1
2	4	2
1	2	1

In order to understand how a convolution kernel works, let us take a small section of the arbitrary layer in section 1.4, focusing specifically on this '6' pixel:

7	8	4	9
5	4	**6**	2
5	7	3	1
3	2	9	9

In order to obtain the new value for the '6' pixel, we must first multiple each pixel in the 3x3 section centered around '6' by the corresponding pixel in the convolution kernel, obtaining the following matrix:

8	8	9
8	24	4
7	6	1

From this, we can take the mean value of this matrix, 8.33, to 2 significant figures. This is the new value for the '6' pixel. This process is repeated for every pixel in the layer, and every layer in the DICOM image, to get the smoothed image. We then crop the image to a cylinder, discarding the outer regions.

2.3 Development Tools

2.3.1 Programming language

We decided to use Python in order to code our program, as it is a language that we were all familiar with before starting the project. We did contemplate using a combination of JavaScript, HTML and CSS for a web based design, but we decided the extra work of learning components of a new programming language in addition to completing the project would have been too much to complete. Python also came with the most pre-written libraries involving DICOM files, however, we decided not to use these.

2.3.2 GUI framework

Some of the GUI frameworks we considered using were.

- Tkinter
- Kivy
- PyQT5
- CustomTkinter [13]

In the end we decided that Kivy and PyQT5 had a steep learning curve, as we did not previously know how to use them, which left us with a choice between Tkinter and CustomTkinter. Although CustomTkinter had slightly different functionality, it was sufficiently similar to Tkinter, which we had used before, that we felt confident in using it. The advantages of CustomTkinter over Tkinter are a nicer-looking GUI, support for light / dark mode and colour themes, and dynamic scaling when resizing an app window.

3 Results

`https://github.com/AlexToogood-Johnson/Gamma-Camera-Uniformity`
This is the GitHub repository where we have put all our code. If you have Git installed on your system, you can run the command:

```
git clone github.com/AlexToogood-Johnson/Gamma-Camera-Uniformity.git
```

Which will download all the files, or else you can download them manually. The appropriate Python libraries can be installed with `pip install -r requirements.txt`.

3.1 Final Program

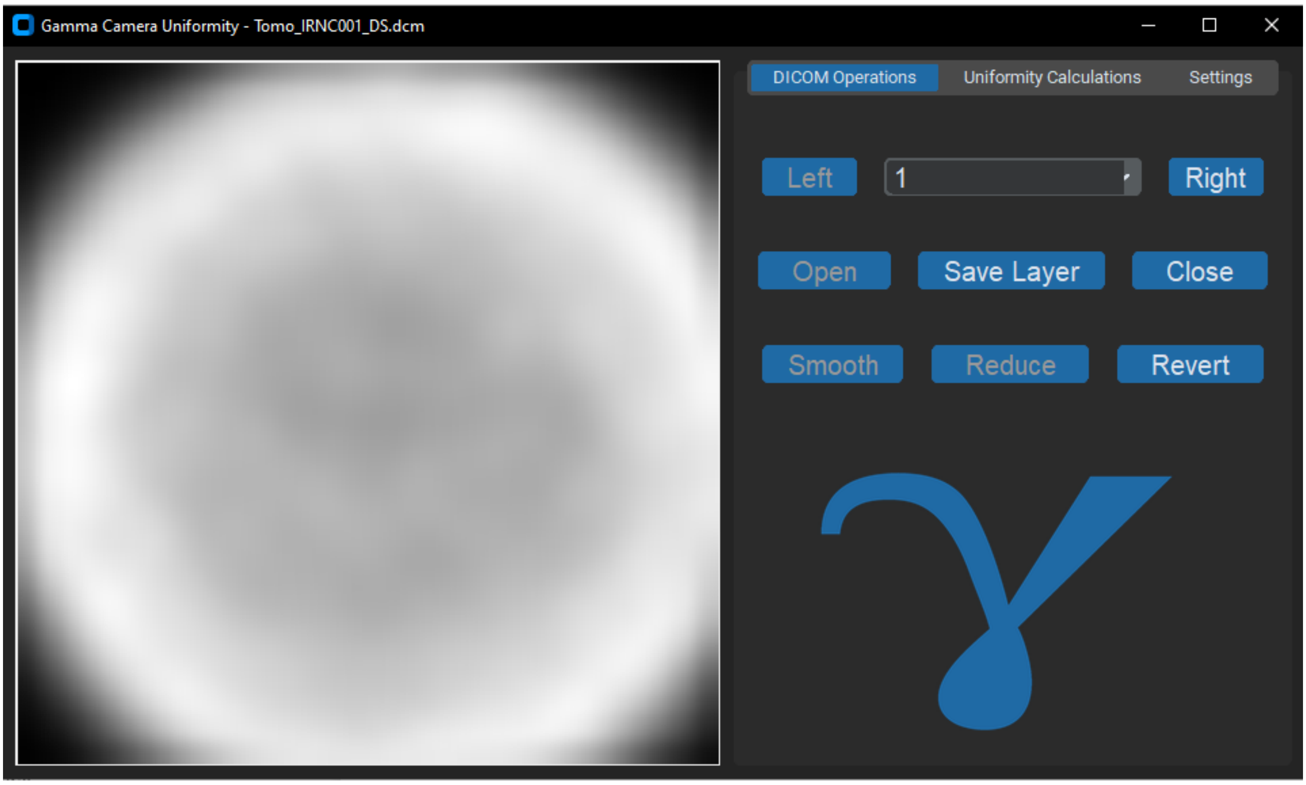

Figure 7: Our program showing a smoothed and cropped DICOM image

Note the following features:

- **Left & Right**: Increases or decreases the index of the layer shown in the preview
- **Open & Close**: Select a DICOM file for processing, or reset the program to the beginning ready for a new file.
- **Save Layer**: Export the current layer as shown in the preview to an image file.
- **Smooth**: Applies the convolution to the image, as specified in Section 2.2.
- **Reduce**: Removes the top and bottom n layers of the image, by default will remove all but the middle 40.
- **Revert**: Undo all processing to the DICOM file.

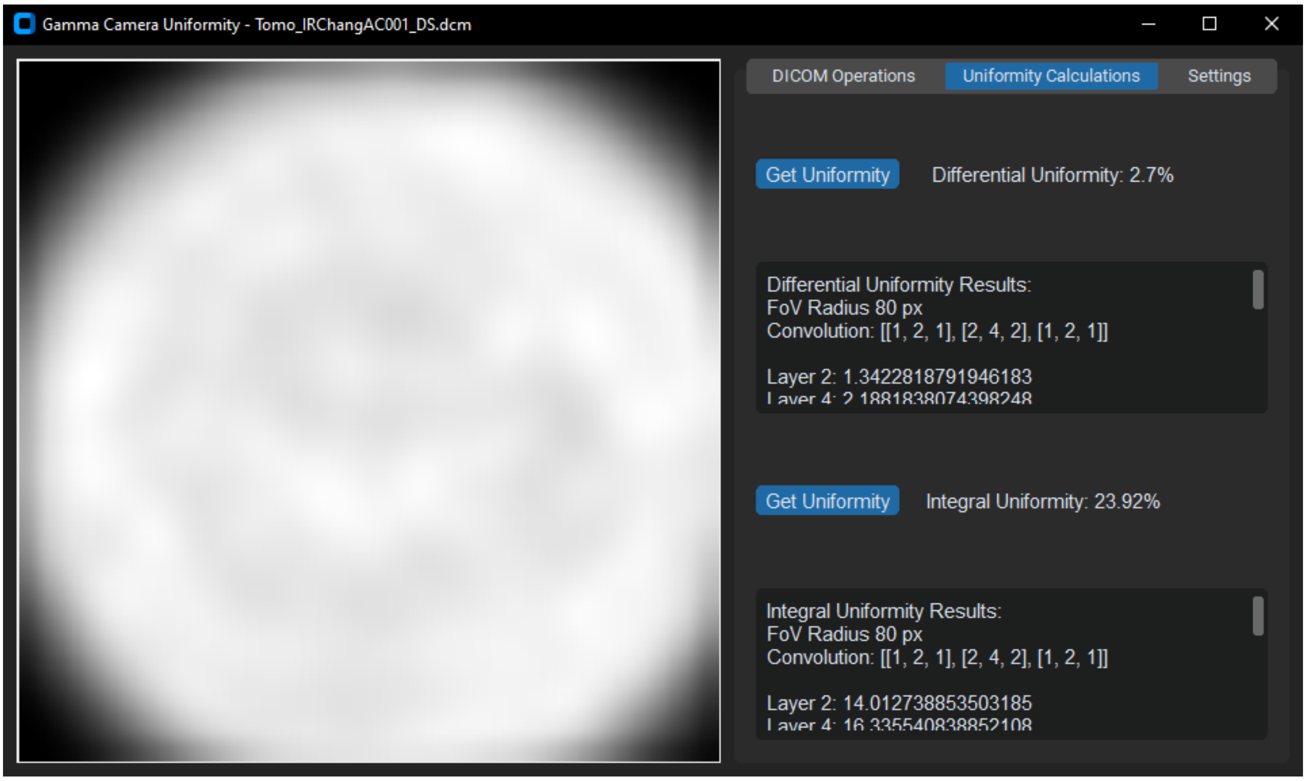

Figure 8: Our program showing the result of a uniformity calculation

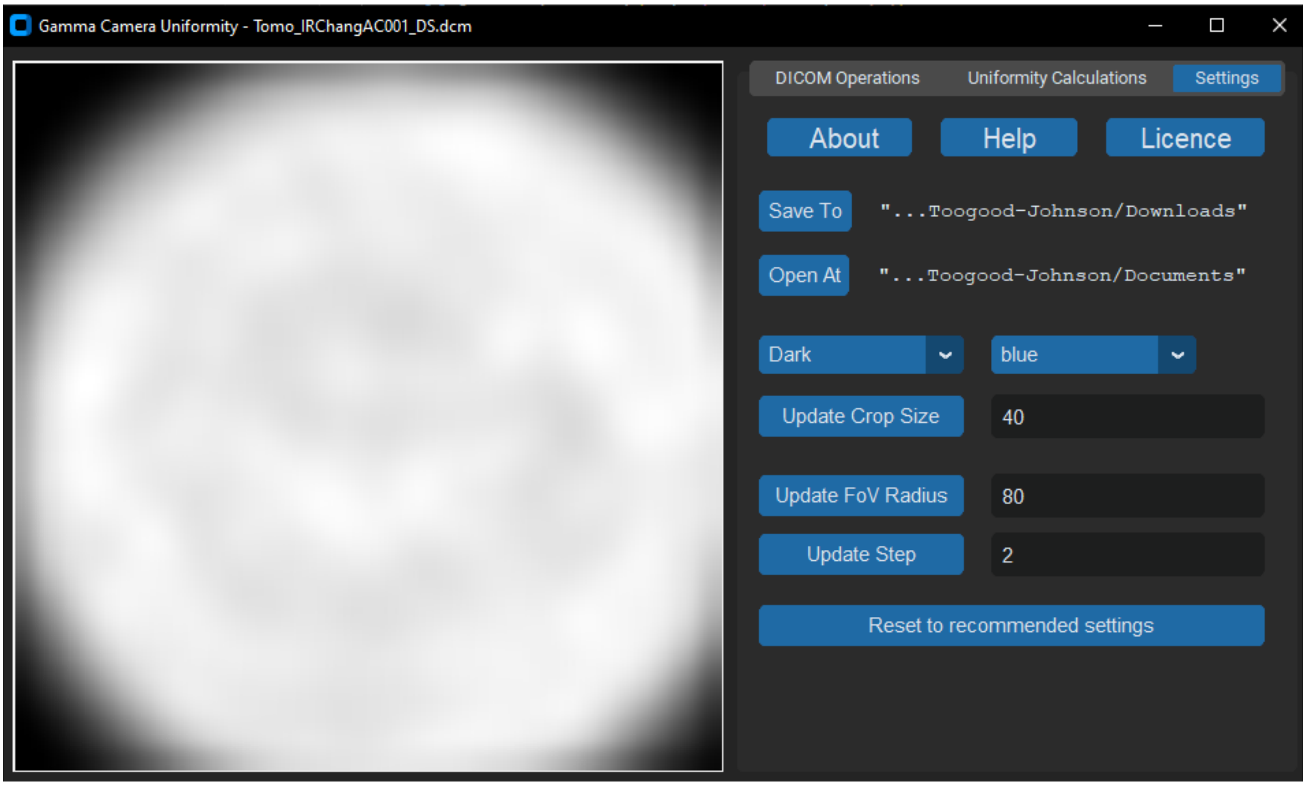

Figure 9: Our program's settings page

Again note the following:

- **Crop Size**: affects the number of layers that will be removed from the top and the bottom.
- **FoV Radius**: Controls the radius of the circle which will be considered for uniformity.
- **Step**: See Section 4.1. A step of 1 calculates the uniformity for every layer, a step of 2 considers every other layer etc. Higher values will execute faster, but will be less accurate.

3.2 Testing

We were provided with two sample DICOM scans of standard phantoms for testing, from which we obtained the following results:

File Name	Differential Uniformity (%)	Integral Uniformity (%)
TOMO_IRNC001_DS.dcm	3.37	30.14
TOMO_IRChangAC001_DS.dcm	2.70	23.92

Table 1: Testing values for provided DICOM images.

The values for differential uniformity are within the acceptable $< 3 - 4\%$ range (although, the first value is the high level of what would be acceptable), however, calculated values for integral uniformity seem particularly high, with a large range [12]. In absence of an online source confirming what an acceptable threshold for integral uniformity would be, we have decided to accept the calculated values as accurate. Existing sources only calculated the integral uniformity over small regions of interest, so values they suggest as maximum thresholds are not applicable to us. Because we consider the whole image, our values are expected to be much larger.

4 Discussions

4.1 Execution Speed

Uniformity calculations, particularly for differential uniformity, are highly computationally intensive tasks. This, without proper optimisation, can result in very slow execution times, on average approximately 2.5 minutes per DICOM image.

This was unacceptably slow, and at 3.65 seconds per layer (for the central 40 layers considered by the program), this is likely slower than manually inspecting each layer for major anomalies. To optimise this, we adjusted the code to allow for multiprocessing, where different calculations could be executed simultaneously by taking advantage of modern multi-core CPUs. This immediately gave an approximately 85% reduction in execution speeds with no impact on the final calculated uniformity values.

To further optimise the program, we decided to only calculate the uniformity for every other layer in the cropped region. Our reasoning was that any significant anomalies would likely affect multiple layers and therefore only considering intermediate layers would be sufficient. This gave us a further approximately 39% reduction when tested. While this did not affect the final uniformity values for the two tested images, this was by chance and it would cause value changes for some images. However, consecutive layers tend to have similar values, so it was decided that this loss of information was justified by the improvement in speed. We did, however, leave an option in settings for this stepping to be disabled completely, or for a custom value (e.g. every third layer) to be used.

File Name	Original (s)	Threaded (s)	Thread & Layer Optimised (s)
TOMO_IRNC001_DS.dcm	145.34	23.86	14.40
TOMO_IRChangAC001_DS.dcm	142.55	22.00	13.85

Table 2: Table of execution times for the uniformity calculations for provided DICOM images.

4.2 Code Structure Improvements

As visible in Appendix A, our code has three major Python files, `dicom_functions.py`, `uniformity_functions.py` and `gui.py`. Due to the complexity of the task and CustomTkinter naturally being quite verbose, this has resulted in some files of considerable length, for example `gui.py` at approximately 600 lines. While file length has no impact on the final result, this length combined with `gui.py` carrying out a large number of tasks made development difficult at times, as it was sometimes challenging to find (for example) the location of one function relative to the other. While we decided that overhauling the structure of the program late in development would not be a justified use of effort, if we had the opportunity to do this again we would pay more careful consideration to how we structured the project, for example a larger number of smaller, more modularised, files that carry out a much smaller number of tasks.

5 Acknowledgements

Section modified for anonymity in public copy

6 References

[1] 3Blue1Brown. *But what is a convolution?* URL: https://www.youtube.com/watch?v=KuXjwB4LzSA&t=1178s (visited on 04/15/2024).

[2] Robert Alley. *What Are Radioactive Tracers?* URL: https://sciencing.com/radioactive-tracers-8330110.html (visited on 04/15/2024).

[3] National Insitute of Biomedical Imaging and Engineering. *Nuclear Medicine*. URL: https://www.nibib.nih.gov/science-education/science-topics/nuclear-medicine (visited on 04/15/2024).

[4] Wikipedia contributors. *Gamma camera — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Gamma_camera&oldid=1216441321 (visited on 04/15/2024).

[5] Wikipedia contributors. *Kernel (image processing) — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Kernel_(image_processing)&oldid=1180652863 (visited on 04/15/2024).

[6] DICOM. *6.2 Value Representation (VR)*. URL: https://dicom.nema.org/medical/dicom/current/output/chtml/part05/sect_6.2.html#table_6.2-1 (visited on 04/15/2024).

[7] DICOM. *8 Encoding of Pixel, Overlay and Waveform Data*. URL: https://dicom.nema.org/medical/dicom/current/output/chtml/part05/chapter_8.html (visited on 04/17/2024).

[8] DICOM. *PS3.6*. URL: https://dicom.nema.org/medical/dicom/current/output/pdf/part06.pdf (visited on 04/15/2024).

[9] Sequoia Healthcare. *MiE Ecam Scintron Single Dual Head Gamma Camera System*. URL: https://www.sqhpl.com/mie-ecam-scintron-single-dual-head-camma-camera.php (visited on 04/15/2024).

[10] LEAD Technologies Inc. *Overview: Basic DICOM File Structure*. URL: https://www.leadtools.com/help/sdk/v21/dicom/api/overview-basic-dicom-file-structure.html (visited on 04/15/2024).

[11] LEAD Technologies Inc. *Overview: Data Element Structure*. URL: https://www.leadtools.com/help/sdk/v21/dicom/api/overview-data-element-structure.html (visited on 04/15/2024).

[12] Michael K.O'Connor. *Quality Control of Scintillation Cameras (Planar and SPECT)*. URL: https://www.aapm.org/meetings/99am/pdf/2741-51264.pdf (visited on 04/19/2024).

[13] Pypi. *customtkinter* 5.2.2. URL: https://pypi.org/project/customtkinter/ (visited on 01/08/2024).

[14] tbd. *Example of 2D Convolution*. URL: http://www.songho.ca/dsp/convolution/convolution2d_example.html (visited on 04/21/2024).

[15] Cancer Research UK. *Radioactive scans for neuroendocrine tumours (NETs)*. URL: https://www.cancerresearchuk.org/about-cancer/neuroendocrine-tumours-nets/getting-diagnosed/tests/radioactive-scans (visited on 04/15/2024).

[16] Vaia. *Gamma Camera*. URL: https://www.vaia.com/en-us/explanations/physics/medical-physics/gamma-camera/ (visited on 04/15/2024).

A Code Appendix

https://github.com/AlexToogood-Johnson/Gamma-Camera-Uniformity

A.1 dicom_functions.py

```python
# dicom_functions.py
"""Functions relating to the manpulation of DICOM images"""

########## IMPORTS ##########

import numpy as np
import os
import json
from typing import NoReturn
from scipy.signal import convolve2d
import struct

########## FUNCTIONS ##########


def get_dicom_data(filepath: str) -> list:
    """Reads the binary of a DICOM file"""

    with open(filepath, "rb") as dicom_file:
        data = dicom_file.read()
    return data


def decode_value(vr, value):
    """Decodes the value of a DICOM element based on its VR"""

    try:
        if vr in "AE AS CS DA DS DT LO LT PN SH ST TM UC UI UR UT UV":
            return value.decode("ascii")
        match vr:
            case "AT": return value.hex()
            case "FL": return struct.unpack(">f", value)[0]
            case "FD": return struct.unpack(">d", value)[0]
            case "IS": return int(value.decode("ascii"))
            case "SL": return struct.unpack(">i", value)[0]
            case "SS": return struct.unpack(">h", value)[0]
            case "SV": return struct.unpack(">q", value)[0]
            case "UL": return int.from_bytes(value, byteorder="little")
            case "US": return int.from_bytes(value, byteorder="little")
            case _: return value

    except Exception:
        return decode_value(vr, value[:-2])


def rearrange_tag(tag: str) -> str:
    """When reading tags from binary, they are in the wrong order"""

    tag = tag[2:4] + tag[0:2] + tag[6:] + tag[4:6]
    return tag


def parse_binary(data) -> dict:
```

```python
    """Parses the raw binary from a DICOM file, and returns the metadata and raw image data"""

    parsed_data = {}
    image_data = []
    data = data[128:]  # Remove preamble
    data = data[4:]  # Remove prefix

    while len(data) > 0:
        tag = rearrange_tag(data[:4].hex())
        data = data[4:]
        vr = data[:2].decode("ascii")
        data = data[2:]
        if vr in "AE AS AT CS DA DS DT FL FD IS LO LT PN SH SL ST SS TM UI UL US":
            length = int.from_bytes(data[:2], byteorder="little")
            data = data[2:]
        else:
            data = data[2:]
            length = int.from_bytes(data[:4], byteorder="little")
            data = data[4:]
        value = decode_value(vr, data[:length])
        data = data[length:]

        if tag == "7fe00010":
            image_data = value
            data = []
            break
        else:
            parsed_data[tag] = [vr, length, value]

    elements = json.load(open("dicom_elements.json"))

    for key, value in parsed_data.items():
        if key in elements:
            parsed_data[key].append(elements[key])
        else:
            parsed_data[key].append("UNKOWN")

    return parsed_data, image_data


def decode_image_data(parsed_data: dict, image_data: list) -> np.ndarray:
    """Converts the raw image data into a 3D Numpy array"""

    bits_allocated = parsed_data["00280100"][2] // 8

    image_data = [int.from_bytes(image_data[i:i + bits_allocated], byteorder="little") for i in range(0, len(image_data), bits_allocated)]
    rows = parsed_data["00280010"][2]
    columns = parsed_data["00280011"][2]
    planes = parsed_data["00280008"][2]
    image_data = np.array(image_data).reshape((planes, rows, columns))

    return image_data


def load_dicom_image(filepath: str) -> tuple:
    """Returns the metadata and image data of a DICOM file"""

    if not os.path.isfile(filepath):
```

```
        raise FileNotFoundError(f"Error: {filepath} not found")
    if not filepath.endswith(".dcm"):
        raise ValueError(f"Error: {filepath} is not a DICOM file")

    parsed_data, image_data = parse_binary(get_dicom_data(filepath))
    image_data = decode_image_data(parsed_data, image_data)

    return parsed_data, image_data


def read_config_file(key: str) -> list | NoReturn:
    """Returns the contents of config.json"""

    if not os.path.isfile("config.json"):
        raise FileNotFoundError(f"Error: config.json not found, please run setup.
    py")

    with open("config.json", "r") as config_file:
        data = json.load(config_file)
    if key not in data:
        raise KeyError(f"Error: key {key} not found in config.json")
    return data[key]


def edit_config_file(key: str, value: list) -> NoReturn:
    """Edits the contents of config.json"""

    if not os.path.isfile("config.json"):
        raise FileNotFoundError(f"Error: config.json not found, please run setup.
    py")

    with open("config.json", "r") as config_file:
        data = json.load(config_file)
    if key not in data:
        raise KeyError(f"Error: key {key} not found in config.json")
    data[key] = value
    with open("config.json", "w") as config_file:
        json.dump(data, config_file, indent=4)


def crop(array: np.ndarray, crop_size: int) -> np.ndarray | NoReturn:
    """Returns the central n x n x n crop of the input array"""

    if not len(array.shape) == 3:
        raise ValueError(f"Expected 3D array. Got {len(array.shape)}D array
    instead")

    center = array.shape[0] // 2
    start = center - crop_size // 2
    end = start + crop_size
    return array[start:end, start:end, start:end]


def apply_convolution(array: np.ndarray, convolution: list) -> np.ndarray |
    NoReturn:
    """Applies a smoothing convolution to a 2D Numpy array"""

    if not len(array.shape) == 3:
        raise ValueError(f"Expected 3D array. Got {len(array.shape)}D array
    instead")
    new_array = np.zeros_like(array)
```

```
    for i in range(array.shape[0]):
        new_array[i] = convolve2d(array[i], convolution, mode='same', boundary='
    fill', fillvalue=0)
    return new_array


def remove_image_edges(array: np.ndarray) -> np.ndarray | NoReturn:
    """Removes the edges of a 3D Numpy array"""

    if not len(array.shape) == 3:
        raise ValueError(f"Expected 3D array. Got {len(array.shape)}D array
    instead")

    radius = array.shape[0] // 2
    for layer in range(len(array)):
        for row in range(len(array[layer])):
            for column in range(len(array[layer][row])):
                if (radius - column) ** 2 + (radius - row) ** 2 > radius ** 2:
                    array[layer][row][column] = 10000

    return array


def max_pixel(array: list) -> int | NoReturn:
    """Returns the maximum value in a list, ignoring None values"""

    return max([value for value in array if value != 10000])


def min_pixel(array: list) -> int | NoReturn:
    """Returns the minimum value in a list, ignoring None values"""

    return min([value for value in array if value != 10000])
```

A.2 uniformity_functions.py

```
# uniformity_functions.py
"""Functions relating to uniformity calculations"""

import numpy as np
from PIL import Image, ImageDraw
import concurrent.futures


class UniformityLayer:
    def __init__(self, bin_data: list, circ_rad: int) -> None:
        self.bin_data = np.array(bin_data, dtype=np.uint8)
        self.circ_rad = circ_rad
        self.cropped_data = None

    def crop_to_circle(self) -> None:
        image = Image.fromarray(self.bin_data)
        if image.mode != 'RGBA':
            image = image.convert('RGBA')
        transparent_image = Image.new('RGBA', image.size)
        transparent_image.paste(image, (0, 0))
        mask = Image.new('L', image.size, 0)
        draw = ImageDraw.Draw(mask)
        center_x, center_y = image.width // 2, image.height // 2
        draw.ellipse((center_x - self.circ_rad, center_y - self.circ_rad,
```

```
                        center_x + self.circ_rad, center_y + self.circ_rad), fill=255)
        transparent_image.putalpha(mask)
        self.cropped_data = np.array(transparent_image)

    def _calculate_uniformity(self, slices):
        max_uniformity = 0
        for s in slices:
            if np.isnan(s).all():
                continue  # Skip entirely transparent slices
            valid_slice = s[~np.isnan(s)]
            if valid_slice.size >= 2:
                max_brightness = np.nanmax(valid_slice)
                min_brightness = np.nanmin(valid_slice)
                if max_brightness + min_brightness != 0:
                    uniformity = abs((max_brightness - min_brightness) / ((max_brightness + min_brightness) / 2) * 100)
                    max_uniformity = max(max_uniformity, uniformity)
        return max_uniformity

    def differential(self) -> float:
        if self.cropped_data is None:
            print("No data to process.")
            return 0

        alpha = self.cropped_data[:, :, 3] > 0
        grayscale = 0.299 * self.cropped_data[:, :, 0] + \
            0.587 * self.cropped_data[:, :, 1] + \
            0.114 * self.cropped_data[:, :, 2]
        grayscale[~alpha] = np.nan

        horizontal_slices = [grayscale[y, x:x + 5] for y in range(grayscale.shape[0]) for x in range(grayscale.shape[1] - 4)]
        vertical_slices = [grayscale[y:y + 5, x] for x in range(grayscale.shape[1]) for y in range(grayscale.shape[0] - 4)]

        with concurrent.futures.ThreadPoolExecutor() as executor:
            futures = [
                executor.submit(self._calculate_uniformity, horizontal_slices),
                executor.submit(self._calculate_uniformity, vertical_slices)
            ]
            results = [future.result() for future in concurrent.futures.as_completed(futures)]

        return max(results)

    def integral(self) -> float:
        if self.cropped_data is None:
            print("No data to process.")
            return 0

        alpha = self.cropped_data[:, :, 3]
        grayscale = 0.299 * self.cropped_data[:, :, 0] + \
            0.587 * self.cropped_data[:, :, 1] + \
            0.114 * self.cropped_data[:, :, 2]
        grayscale[alpha == 0] = np.nan
        valid_grayscale = grayscale[~np.isnan(grayscale)]

        if valid_grayscale.size == 0:
            return 0

```

```python
        max_brightness = np.nanmax(valid_grayscale)
        min_brightness = np.nanmin(valid_grayscale)

        integral_uniformity = abs((max_brightness - min_brightness) / ((max_brightness + min_brightness) / 2) * 100)
        return integral_uniformity
```

A.3 gui.py

```python
# gui.py
"""The GUI for the gamma camera uniformity program
   Copyright (c) 2024 Alex Toogood-Johnson, Lyall Stewart, Josh Scates, Leah Wells, Zac Baker"""

########## IMPORTS ##########

import customtkinter as ctk
from tkinterdnd2 import TkinterDnD, DND_ALL
from PIL import Image, ImageTk, ImageDraw
import numpy as np
from dicom_functions import *
from uniformity_functions import UniformityLayer
from tkinter import filedialog
import os
import ast
from concurrent.futures import ProcessPoolExecutor
import time

########## CONSTANTS ##########

HEIGHT, WIDTH = 540, 960
FILEPATH = os.path.dirname(os.path.abspath(__file__))

ctk.set_default_color_theme(read_config_file("colour"))
ctk.set_appearance_mode(read_config_file("colour_theme"))

########## CLASSES ##########


class Tk(ctk.CTk, TkinterDnD.DnDWrapper):
    """Sourced from: https://stackoverflow.com/questions/75526264/using-drag-and-drop-files-or-file-picker-with-customtkinter
       Allows for drag and drop file opening within customtkinter"""

    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)
        self.TkdndVersion = TkinterDnD._require(self)


class Gui(Tk):
    """The main GUI class for the Gamma Camera Uniformity program"""

    def __init__(self, *args, **kwargs) -> None:
        Tk.__init__(self, *args, **kwargs)

        BUTTON_WIDTH = 150
        TEXTBOX_WIDTH = 200
        COMPONENT_HEIGHT = 30

        self.bind('<Left>', lambda event: self.left_button_callback())
```

```
        self.bind('<Right>', lambda event: self.right_button_callback())
        self.bind('<Up>', lambda event: self.left_button_callback())
        self.bind('<Down>', lambda event: self.right_button_callback())
        self.bind('<o>', lambda event: self.open_button_callback())
        self.bind('<s>', lambda event: self.apply_convolution())
        self.bind('<BackSpace>', lambda event: self.revert_changes())

        self.title("Gamma Camera Uniformity")
        self.geometry(f"{WIDTH}x{HEIGHT}")
        self.resizable(True, True)

        self.photo_frame = ctk.CTkFrame(self, height=HEIGHT - 20, width=HEIGHT - 20)
        self.photo_frame.place(x=10, y=10)
        self.canvas = ctk.CTkCanvas(self.photo_frame, width=HEIGHT - 20, height=HEIGHT - 20)
        self.canvas.place(x=0, y=0)
        self.canvas_image = None
        self.tk_image = None

        self.dnd_box = ctk.CTkEntry(self.photo_frame, width=HEIGHT - 20, height=HEIGHT - 20, state="readonly")
        self.dnd_box.place(x=0, y=0)
        self.dnd_box.drop_target_register(DND_ALL)
        self.dnd_box.dnd_bind("<<Drop>>", self.get_path)

        self.tab_view = ctk.CTkTabview(self, width=WIDTH - HEIGHT - 10, height=HEIGHT - 10)
        self.tab_view.place(x=HEIGHT, y=0)
        self.tab_1 = self.tab_view.add("    DICOM Operations    ")
        self.tab_2 = self.tab_view.add("   Uniformity Calculations   ")
        self.tab_3 = self.tab_view.add("     Settings     ")

        self.frame_tab_1 = ctk.CTkFrame(self.tab_1, width=WIDTH - HEIGHT - 20)
        self.frame_tab_1.pack(side='top', pady=40)
        self.combobox_values = [str(i) for i in range(1, 40)]
        self.left_button = ctk.CTkButton(self.frame_tab_1, text="Left", width=70, height=10, command=self.left_button_callback, font=("", 20), state="disabled")
        self.select_layer = ctk.CTkComboBox(self.frame_tab_1, width=190, height=10, state="disabled", font=("", 20), values=self.combobox_values, command=self.combobox_callback)
        self.right_button = ctk.CTkButton(self.frame_tab_1, text="Right", width=70, height=10, command=self.right_button_callback, font=("", 20), state="disabled")
        self.left_button.pack(side='left', padx=10)
        self.select_layer.pack(side='left', padx=10)
        self.right_button.pack(side='left', padx=10)

        self.frame_tab_2 = ctk.CTkFrame(self.tab_1, width=WIDTH - HEIGHT - 20)
        self.frame_tab_2.pack()
        self.open_button = ctk.CTkButton(self.frame_tab_2, text="   Open   ", width=30, height=20, font=("", 20), command=self.open_button_callback)
        self.close_button = ctk.CTkButton(self.frame_tab_2, text="   Close   ", width=30, height=20, font=("", 20), state="disabled", command=self.close_button_callback)
        self.save_button = ctk.CTkButton(self.frame_tab_2, text="  Save Layer  ", width=30, height=20, font=("", 20), state="disabled", command=self.save_button_callback)
        self.open_button.pack(side='left', padx=10)
        self.save_button.pack(side='left', padx=10)
```

```
        self.close_button.pack(side='left', padx=10)
        self.open_button.configure(state='normal')

        self.frame_tab_4 = ctk.CTkFrame(self.tab_1, width=WIDTH - HEIGHT - 20)
        self.frame_tab_4.pack(pady=40)
        self.apply_convolution_button = ctk.CTkButton(self.frame_tab_4, text="
Smooth  ", width=30, height=20, font=("", 20), state="disabled", command=self.
apply_convolution)
        self.reduce_image_button = ctk.CTkButton(self.frame_tab_4, text="
Reduce   ", width=30, height=20, font=("", 20), state="disabled", command=self
.reduce_image)
        self.revert_changes_button = ctk.CTkButton(self.frame_tab_4, text="
Revert   ", width=30, height=20, font=("", 20), state="disabled", command=self
.revert_changes)
        self.apply_convolution_button.pack(side='left', padx=10)
        self.reduce_image_button.pack(side='left', padx=10)
        self.revert_changes_button.pack(side='left', padx=10)

        self.frame_tab_5 = ctk.CTkFrame(self.tab_1, width=WIDTH - HEIGHT - 20,
height=190)
        self.frame_tab_5.pack(side='bottom', pady=20)
        self.logo = Image.open(os.path.join(FILEPATH, "logo.png"))
        self.logo = self.logo.resize((WIDTH - HEIGHT - 20, 190), Image.LANCZOS)
        self.tk_logo = ImageTk.PhotoImage(self.logo)
        self.logo_label = ctk.CTkLabel(self.frame_tab_5, image=self.tk_logo, text
="")
        self.logo_label.pack()

        self.frame_tab_6 = ctk.CTkFrame(self.tab_3, width=WIDTH - HEIGHT - 20)
        self.frame_tab_6.pack(pady=10)
        self.about_button = ctk.CTkButton(self.frame_tab_6, text="    About   ",
width=30, height=20, font=("", 20), command=lambda: AboutGUI().mainloop())
        self.help_button = ctk.CTkButton(self.frame_tab_6, text="    Help    ",
width=30, height=20, font=("", 20), command=lambda: HelpGUI().mainloop())
        self.licence_button = ctk.CTkButton(self.frame_tab_6, text="   Licence
", width=30, height=20, font=("", 20), command=lambda: LicenceGUI().mainloop()
)
        self.about_button.pack(side='left', padx=10)
        self.help_button.pack(side='left', padx=10)
        self.licence_button.pack(side='left', padx=10)

        self.frame_tab_7 = ctk.CTkFrame(self.tab_3, width=WIDTH - HEIGHT - 20)
        self.frame_tab_7.pack(pady=(10, 2), anchor='w')
        self.change_saving_directory_button = ctk.CTkButton(self.frame_tab_7,
text="Save To", width=30, height=30, font=("", 15), command=self.
save_directory_button_callback)
        dirname = read_config_file("default_file_saving_directory")
        self.saving_directory_name = ctk.CTkLabel(self.frame_tab_7, text=self.
normalize_directory(dirname), font=("courier", 15))
        self.change_saving_directory_button.pack(side='left', padx=10)
        self.saving_directory_name.pack(side='left', padx=10, pady=5)

        self.frame_tab_8 = ctk.CTkFrame(self.tab_3, width=WIDTH - HEIGHT - 20)
        self.frame_tab_8.pack(pady=(2, 10), anchor='w')
        self.change_opening_directory_button = ctk.CTkButton(self.frame_tab_8,
text="Open At", width=30, height=30, font=("", 15), command=self.
open_directory_button_callback)
        dirname = read_config_file("default_file_opening_directory")
        self.opening_directory_name = ctk.CTkLabel(self.frame_tab_8, text=self.
normalize_directory(dirname), font=("courier", 15))
        self.change_opening_directory_button.pack(side='left', padx=10)
```

```
        self.opening_directory_name.pack(side='left', padx=10, pady=10)

        self.frame_tab_9 = ctk.CTkFrame(self.tab_3, width=WIDTH - HEIGHT - 20)
        self.frame_tab_9.pack(pady=10, anchor='w')
        self.theme_menu = ctk.CTkOptionMenu(self.frame_tab_9, font=("", 15), width=BUTTON_WIDTH, command=lambda event: self.theme_changed(event), values=["Light", "Dark", "System"])
        self.theme_menu.pack(side='left', padx=10)
        self.theme_menu.set(read_config_file("colour_theme"))
        self.colour_menu = ctk.CTkOptionMenu(self.frame_tab_9, font=("", 15), width=BUTTON_WIDTH, command=lambda event: self.colour_changed(event), values=["blue", "green", "dark-blue"])
        self.colour_menu.pack(side='left', padx=10)
        self.colour_menu.set(read_config_file("colour"))

        self.frame_tab_10 = ctk.CTkFrame(self.tab_3, width=WIDTH-HEIGHT-20)
        self.frame_tab_10.pack(pady=5, anchor='w')

        self.change_default_crop_size_button = ctk.CTkButton(self.frame_tab_10, text="Update Crop Size", width=BUTTON_WIDTH, height=COMPONENT_HEIGHT, font=("", 15), command=self.change_crop_size_callback)
        self.change_default_crop_size_button.pack(side='left', padx=10)
        self.crop_size_textbox = ctk.CTkTextbox(self.frame_tab_10, width=TEXTBOX_WIDTH, height=1, font=("", 15))
        self.crop_size_textbox.insert("1.0", str(read_config_file("crop_amount")))
        self.crop_size_textbox.pack(side='left', padx=10)

        self.frame_tab_11 = ctk.CTkFrame(self.tab_3, width=WIDTH - HEIGHT - 20)
        self.frame_tab_11.pack(pady=(10, 0), anchor='w')
        self.change_fov_radius_button = ctk.CTkButton(self.frame_tab_11, text="Update FoV Radius", width=BUTTON_WIDTH, height=COMPONENT_HEIGHT, font=("", 15), command=self.change_fov_radius_callback)
        self.change_fov_radius_button.pack(side='left', padx=10)
        self.fov_radius_textbox = ctk.CTkTextbox(self.frame_tab_11, width=TEXTBOX_WIDTH, height=1, font=("", 15))
        self.fov_radius_textbox.insert("1.0", str(read_config_file("fov_radius")))

        self.fov_radius_textbox.pack(side='left', padx=10, pady=10)

        self.frame_tab_14 = ctk.CTkFrame(self.tab_3, width=WIDTH-HEIGHT-20)
        self.frame_tab_14.pack(pady=(0, 10), anchor='w')
        self.draw_crop_size_button = ctk.CTkButton(self.frame_tab_14, text="Draw FoV Radius", width=BUTTON_WIDTH + TEXTBOX_WIDTH + 20, height=COMPONENT_HEIGHT, font=("", 15), command=self.draw_fov_radius_callback)
        self.draw_crop_size_button.pack(side='left', padx=10, pady=10)
        self.draw_crop_size_button.configure(state="disabled")

        self.frame_tab_13 = ctk.CTkFrame(self.tab_3, width=WIDTH-HEIGHT-20)
        self.frame_tab_13.pack(anchor='w')

        self.change_step_button = ctk.CTkButton(self.frame_tab_13, text="Update Step", width=BUTTON_WIDTH, height=COMPONENT_HEIGHT, font=("", 15), command=self.change_step)
        self.change_step_button.pack(side='left', padx=10)
        self.step_textbox = ctk.CTkTextbox(self.frame_tab_13, width=TEXTBOX_WIDTH, height=1, font=("", 15))
        self.step_textbox.insert("1.0", str(read_config_file("step")))
        self.step_textbox.pack(side='left', padx=10)

```

```
        self.frame_tab_15 = ctk.CTkFrame(self.tab_3, width=WIDTH - HEIGHT - 20)
        self.frame_tab_15.pack(pady=10, anchor='w')
        self.reset_button = ctk.CTkButton(self.frame_tab_15, text="Reset to recommended settings", width=BUTTON_WIDTH + TEXTBOX_WIDTH + 20, height=30, font=("", 15), command=self.reset_settings)
        self.reset_button.pack(side='left', padx=10, pady=10)

        self.differential_uniformity_frame = ctk.CTkFrame(self.tab_2, width=WIDTH - HEIGHT - 20)
        self.differential_uniformity_frame.pack(side='top', pady=40, anchor='w')

        self.get_differential_uniformity_button = ctk.CTkButton(self.differential_uniformity_frame, text="Get Uniformity", width=30, height=20, font=("", 15), command=self.uniformity_callback)
        self.get_differential_uniformity_button.pack(side='left', padx=10, anchor='w')
        self.differential_uniformity_label = ctk.CTkLabel(self.differential_uniformity_frame, text="Differential Uniformity: ", font=("", 15))
        self.differential_uniformity_label.pack(side='left', padx=10)
        self.differential_text_output = ctk.CTkTextbox(self.tab_2, width=30, height=10, font=("", 15))
        self.differential_text_output.pack(padx=10, pady=10, expand=True, fill='both')

        self.integral_uniformity_frame = ctk.CTkFrame(self.tab_2, width=WIDTH - HEIGHT - 20)
        self.integral_uniformity_frame.pack(pady=40, anchor='w')
        self.get_integral_uniformity_button = ctk.CTkButton(self.integral_uniformity_frame, text="Get Uniformity", width=30, height=20, font=("", 15), command=self.uniformity_callback)
        self.get_integral_uniformity_button.pack(side='left', padx=10, anchor='w')
        self.integral_uniformity_label = ctk.CTkLabel(self.integral_uniformity_frame, text="Integral Uniformity: ", font=("", 15))
        self.integral_uniformity_label.pack(side='left', padx=10)
        self.integral_text_output = ctk.CTkTextbox(self.tab_2, width=30, height=10, font=("", 15))
        self.integral_text_output.pack(padx=10, pady=10, expand=True, fill='both')

        self.get_differential_uniformity_button.configure(state="disabled")
        self.get_integral_uniformity_button.configure(state="disabled")

    def change_step(self) -> None:
        step_size = self.step_textbox.get("1.0", "end-1c")
        if step_size.isdigit() and int(step_size) > 0 and int(step_size) < 40:
            edit_config_file("crop", int(step_size))
        else:
            self.step_size.delete("1.0", "end")
            self.step_size.insert("1.0", str(read_config_file("step")))

    def reset_settings(self) -> None:
        self.crop_size_textbox.delete("1.0", "end")
        self.crop_size_textbox.insert("1.0", "40")
        self.fov_radius_textbox.delete("1.0", "end")
        self.fov_radius_textbox.insert("1.0", "80")
        self.step_textbox.delete("1.0", "end")
        self.step_textbox.insert("1.0", "2")
        self.change_crop_size_callback()

```

```python
    def uniformity_callback(self) -> None:
        self.cropped_dicom_image = apply_convolution(self.current_dicom_image, read_config_file("convolution"))
        self.cropped_dicom_image = crop(self.current_dicom_image, read_config_file("crop_amount"))

        self.differential_text_output.insert("1.0", "Loading...")
        self.integral_text_output.insert("1.0", "Loading...")

        differential_vals = []
        integral_vals = []

        with ProcessPoolExecutor() as executor:
            future_to_type = {}
            for layer_index in range(0, read_config_file("crop_amount") - 1, 2):
                dicom_slice = self.current_dicom_image[layer_index]
                normalized_slice = ((dicom_slice - np.min(dicom_slice)) / (np.max(dicom_slice) - np.min(dicom_slice)) * 255).astype(np.uint8)

                image = Image.fromarray(normalized_slice)
                image = image.convert("RGB")
                image = image.resize((HEIGHT - 20, HEIGHT - 20), Image.LANCZOS)

                layer = UniformityLayer(np.array(image), read_config_file("fov_radius"))
                layer.crop_to_circle()

                future_to_type[executor.submit(layer.differential)] = 'differential'
                future_to_type[executor.submit(layer.integral)] = 'integral'

            start_time = time.time()
            for future, type_label in future_to_type.items():
                result = future.result()
                if type_label == 'differential':
                    differential_vals.append(result)
                else:
                    integral_vals.append(result)
            end_time = time.time()

        if differential_vals:
            self.display_uniformity(integral_vals, differential_vals)
        else:
            print("No differential values to display.")

        print(f"Total Execution Time: {end_time - start_time} seconds")

    def display_uniformity(self, integral: int, differential: int) -> None:
        self.differential_uniformity_results = differential
        self.differential_uniformity_label.configure(text=f" Differential Uniformity: {round(max(self.differential_uniformity_results), 2)}%")
        fov = read_config_file("fov_radius")
        convolution = read_config_file("convolution")
        textbox_text = f"Differential Uniformity Results:\nFoV Radius {fov} px \nConvolution: {convolution}\n\n"
        for i, result in enumerate(self.differential_uniformity_results):
            textbox_text += f"Layer {(i+1) * read_config_file('step')}: {result}\n"
        self.differential_text_output.delete("1.0", "end")
        self.differential_text_output.insert("1.0", textbox_text)

```

```
        self.integral_uniformity_results = integral
        self.integral_uniformity_label.configure(text=f"Integral Uniformity: {round(max(self.integral_uniformity_results), 2)}%")
        fov = read_config_file("fov_radius")
        convolution = read_config_file("convolution")
        textbox_text = f"Integral Uniformity Results:\nFoV Radius {fov} px \nConvolution: {convolution}\n\n"
        for i, result in enumerate(self.integral_uniformity_results):
            textbox_text += f"Layer {(i+1) * read_config_file('step')}: {result}\n"
        self.integral_text_output.delete("1.0", "end")
        self.integral_text_output.insert("1.0", textbox_text)

    def change_crop_size_callback(self) -> None:
        """Changes the crop size stored in the config file"""

        crop_size = self.crop_size_textbox.get("1.0", "end-1c")
        if crop_size.isdigit() and int(crop_size) > 0 and int(crop_size) < 100:
            edit_config_file("crop_amount", int(crop_size))
        else:
            self.crop_size_textbox.delete("1.0", "end")
            self.crop_size_textbox.insert("1.0", str(read_config_file("crop_amount")))

    def change_fov_radius_callback(self) -> None:
        """Changes the field of view radius stored in the config file"""
        fov_rad = self.fov_radius_textbox.get("1.0", "end-1c")
        if fov_rad.isdigit() and int(fov_rad) > 0 and int(fov_rad) < 100:
            edit_config_file("fov_radius", int(fov_rad))
        else:
            self.fov_radius_textbox.delete("1.0", "end")
            self.fov_radius_textbox.insert("1.0", str(read_config_file("fov_radius")))

    def draw_fov_radius_callback(self) -> None:
        """Draws the field of view radius on the image"""
        self.display_image(int(self.select_layer.get()), int(self.fov_radius_textbox.get("1.0", "end")))

    def colour_changed(self, event) -> None:
        """Changes the colour of the GUI"""
        colour = self.colour_menu.get()
        ctk.set_default_color_theme(colour)
        edit_config_file("colour", colour)

    def theme_changed(self, event) -> None:
        """Changes the colour theme of the GUI"""
        theme = self.theme_menu.get()
        ctk.set_appearance_mode(theme)
        edit_config_file("colour_theme", theme)

    def save_directory_button_callback(self) -> None:
        """Changes the directory stored in the config file for saving files"""
        directory = filedialog.askdirectory()
        if directory:
            if os.path.isdir(directory):
                self.saving_directory_name.configure(text=self.normalize_directory(directory))
                edit_config_file("default_file_saving_directory", directory)

    def normalize_directory(self, directory: str) -> str:
```

```
        """Makes all directory names 30 character long for display purposes"""
        if len(directory) >= 25:
            directory = '"...' + directory[-25:] + '"'
        else:
            directory = '"' + directory + '"' + ' ' * (28 - len(directory))
        return directory

    def open_directory_button_callback(self) -> None:
        """Changes the directory stored in the config file for opening files"""
        directory = filedialog.askdirectory()
        if directory:
            if os.path.isdir(directory):
                self.opening_directory_name.configure(text=self.normalize_directory(directory))
                edit_config_file("default_file_opening_directory", directory)

    def reduce_image(self) -> None:
        """Reduces the image by the amount specified in the config file, and displays in the GUI"""
        if self.reduce_image_button.cget("state") != 'normal': return
        self.current_dicom_image = crop(self.current_dicom_image, read_config_file("crop_amount"))
        self.reduce_image_button.configure(state="disabled")
        self.apply_convolution_button.configure(state="disabled")
        self.revert_changes_button.configure(state="normal")
        self.select_layer.set("1")
        self.left_button.configure(state="disabled")
        self.combobox_values = [str(i) for i in range(1, read_config_file("crop_amount") - 1)]
        self.display_image(int(self.select_layer.get()))

    def apply_convolution(self) -> None:
        """Applies a convolution to the image, and displays in the GUI"""
        if self.apply_convolution_button.cget("state") != 'normal': return
        self.current_dicom_image = apply_convolution(self.current_dicom_image, read_config_file("convolution"))
        self.display_image(int(self.select_layer.get()))
        self.apply_convolution_button.configure(state="disabled")
        self.reduce_image_button.configure(state="normal")
        self.revert_changes_button.configure(state="normal")

    def revert_changes(self) -> None:
        """Reverts the visual changes made to the image - convolution and/or cropping"""
        if self.revert_changes_button.cget("state") != 'normal': return
        self.current_dicom_image = self.original_dicom_image
        self.display_image(int(self.select_layer.get()))
        self.reduce_image_button.configure(state="disabled")
        self.apply_convolution_button.configure(state="normal")
        self.revert_changes_button.configure(state="disabled")

    def combobox_callback(self, choice: str) -> None:
        """Callback for selection of layer number from combobox"""
        if int(choice) == 1:
            self.left_button.configure(state="disabled")
            self.right_button.configure(state="normal")
        elif choice == self.combobox_values[-1]:
            self.left_button.configure(state="normal")
            self.right_button.configure(state="disabled")
        else:
            self.left_button.configure(state="normal")
```

```
            self.right_button.configure(state="normal")
        self.display_image(int(choice))

    def left_button_callback(self) -> None:
        """Callback for left button press on layer selection"""
        if self.left_button.cget("state") != 'normal': return
        self.select_layer.set(int(self.select_layer.get()) - 1)
        if int(self.select_layer.get()) == 1:
            self.left_button.configure(state="disabled")
        if self.select_layer.get() != self.combobox_values[-1]:
            self.right_button.configure(state="normal")
        self.display_image(int(self.select_layer.get()))

    def right_button_callback(self):
        """Callback for the right button press on layer selection"""
        if self.right_button.cget("state") != 'normal': return
        self.select_layer.set(int(self.select_layer.get()) + 1)
        if int(self.select_layer.get()) > 1:
            self.left_button.configure(state="normal")
        if self.select_layer.get() == self.combobox_values[-1]:
            self.right_button.configure(state="disabled")
        self.display_image(int(self.select_layer.get()))

    def open_button_callback(self) -> None:
        """Callback for the open button press, opens a file dialog to select a
DICOM file to open"""
        if self.open_button.cget("state") != 'normal': return
        default = read_config_file("default_file_opening_directory")
        if os.path.isdir(default):
            file_path = filedialog.askopenfilename(filetypes=[("DICOM Files", "*.
dcm")], initialdir=default)
        else:
            file_path = filedialog.askopenfilename(filetypes=[("DICOM Files", "*.
dcm")], initialdir=os.getcwd())
        if file_path:
            self.title(f"Gamma Camera Uniformity - {file_path.split('/')[-1]}")
            self.fit_dicom_image(file_path)

    def close_button_callback(self) -> None:
        """Callback for the close button press, resets the GUI to its initial
state"""
        self.current_dicom_image = None
        self.original_dicom_image = None
        self.combobox_values = []
        self.select_layer.set("")
        self.left_button.configure(state="disabled")
        self.right_button.configure(state="disabled")
        self.select_layer.configure(state="disabled")
        self.open_button.configure(state="normal")
        self.save_button.configure(state="disabled")
        self.close_button.configure(state="disabled")
        self.reduce_image_button.configure(state="disabled")
        self.apply_convolution_button.configure(state="disabled")
        self.revert_changes_button.configure(state="disabled")
        self.get_integral_uniformity_button.configure(state="disabled")
        self.get_differential_uniformity_button.configure(state="disabled")
        self.differential_text_output.delete("1.0", "end")
        self.integral_text_output.delete("1.0", "end")
        self.differential_uniformity_label.configure(text="Uniformity: ")
        self.integral_uniformity_label.configure(text="Uniformity: ")
        self.title("Gamma Camera Uniformity")
```

```
        self.draw_crop_size_button.configure(state="disabled")
        self.canvas.delete(self.canvas_image)
        self.dnd_box.lift()

    def save_button_callback(self) -> None:
        """Callback for the save button press, saves the currently viewed layer as a PNG file"""
        dicom_slice = self.current_dicom_image[int(self.select_layer.get()) - 1]
        normalized_slice = ((dicom_slice - np.min(dicom_slice)) / (np.max(dicom_slice) - np.min(dicom_slice)) * 255).astype(np.uint8)
        default = read_config_file("default_file_saving_directory")
        if os.path.isdir(default):
            file_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG Files", "*.png")], initialdir=default)
        else:
            file_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG Files", "*.png")], initialdir=os.getcwd())
        if file_path:
            Image.fromarray(normalized_slice).save(file_path)

    def get_path(self, event) -> None:
        image_path = event.data.replace("{", "").replace("}", "")
        if image_path.endswith("dcm"):
            self.fit_dicom_image(image_path)

    def fit_dicom_image(self, image_path) -> None:
        self.current_dicom_image = load_dicom_image(image_path)[1]
        if not len(self.current_dicom_image.shape) == 3:
            self.current_dicom_image = None
            return  # Rejects 2D images
        self.original_dicom_image = self.current_dicom_image
        self.combobox_values = [str(i) for i in range(1, self.current_dicom_image.shape[0] + 1)]
        self.right_button.configure(state="normal")
        self.select_layer.configure(state="normal")
        self.select_layer.set("1")
        self.open_button.configure(state="disabled")
        self.save_button.configure(state="normal")
        self.close_button.configure(state="normal")
        self.apply_convolution_button.configure(state="normal")
        self.get_integral_uniformity_button.configure(state="normal")
        self.get_differential_uniformity_button.configure(state="normal")
        self.draw_crop_size_button.configure(state="normal")
        self.display_image(1)

    def display_image(self, layer: int, fov_rad: int = 0) -> None:
        dicom_slice = self.current_dicom_image[layer - 1]
        normalized_slice = ((dicom_slice - np.min(dicom_slice)) / (np.max(dicom_slice) - np.min(dicom_slice)) * 255).astype(np.uint8)

        image = Image.fromarray(normalized_slice)
        image = image.convert("RGB")
        image = image.resize((HEIGHT - 20, HEIGHT - 20), Image.LANCZOS)
        if fov_rad > 0:  # Providing a FoV of 0 indicates that the user does not want a FoV drawn
            draw = ImageDraw.Draw(image)
            center_x, center_y = (HEIGHT - 20) // 2, (HEIGHT - 20) // 2
            top_left = (center_x - fov_rad, center_y - fov_rad)
            bottom_right = (center_x + fov_rad, center_y + fov_rad)
            draw.ellipse([top_left, bottom_right], outline="red", width=2)

```

```
        self.tk_image = ImageTk.PhotoImage(image)
        if self.canvas_image:
            self.canvas.delete(self.canvas_image)
        self.canvas_image = self.canvas.create_image(-1, -1, anchor="nw", image=self.tk_image, tags="image")
        self.canvas.lift(self.canvas_image)
        self.dnd_box.lower()


class LicenceGUI(Tk):
    def __init__(self, *args, **kwargs) -> None:
        Tk.__init__(self, *args, **kwargs)

        self.title("Licence")
        self.geometry("500x300")
        self.resizable(False, False)

        self.licence_text = ctk.CTkTextbox(self, width=480, height=280)
        self.licence_text.place(x=10, y=10)
        self.licence_text.insert("1.0", open(os.path.join(FILEPATH, "LICENCE")).read())
        self.licence_text.configure(state="disabled")


class AboutGUI(Tk):
    def __init__(self, *args, **kwargs) -> None:
        Tk.__init__(self, *args, **kwargs)
        self.title("About")
        self.geometry("500x300")
        self.resizable(False, False)
        text = """
Gamma Camera Uniformity Program
Developed at Exeter Maths School as part of the Exeter Maths Certificate
in 2024.
The purpose of this software is to perform calculations to determine the
uniformity values of DICOM SPECT images taken from a gamma camera.

For more information, and to view the source code, visit the GitHub page:
https://github.com/ExeMS/NHS-EMC2024

This software is distributed under the MIT License.
Copyright (c) 2024 Alex Toogood-Johnson, Lyall Stewart, Josh Scates, Leah Wells, Zac Baker.

               """
        self.about_text = ctk.CTkTextbox(self, width=480, height=280)
        self.about_text.place(x=10, y=10)
        self.about_text.insert("1.0", text)
        self.about_text.configure(state="disabled")


class HelpGUI(Tk):
    def __init__(self, *args, **kwargs) -> None:
        Tk.__init__(self, *args, **kwargs)
        self.title("Help")
        self.geometry("500x300")
        self.resizable(False, False)
        text = """
This section provides some information on how to use this program, although for further details please contact the authors. To start, open a DICOM file by selecting it from the file directory, which you can access by typing 'o' on
```

```
your keyboard or by clicking the 'open' button. You can also directly open a file by holding it over the left half of the program, which functions as a drag and drop box as long as there isn't currently a file open.

The chosen file must be a .dcm file, and must be a SPECT image in 3 dimensions in order to be opened by the program. Once an image has been opened, you can close the image by clicking the 'close' button. In order to view each layer of the DICOM image, you can either click the left, right, up or down buttons on your keyboard, select the left or right buttons on the gui, or manually select a layer from the combobox. Please note that the first and last 20-40 layers in a DICOM uniformity image may be blank or incomplete.

The 'reduce' button, which you can either click, or press 'r' on your keyboard, fixes this by removing the first n and last n layers of the image. The default and recommended value is 44, although this can be changed in the settings tab.

The smooth button, which you can either click or press 's' on your keyboard, applies a 9 point weighted convolution in order to smooth each layer of the DICOM image. The convolution used is [[1,2,1],[2,4,2],[1,2,1]] as specified in the NEMA handbook. We strongly recommend not changing this, however if necessary than this can be manually changed inside 'config.json'.

The revert button, which you can either click, or press backspace on your keyboard, visually reverts the smoothing and / or cropping changes to the DICOM image. The save layer button saves the currently selected layer of the DICOM image as a PNG file. The directory at which this opens, as well as the directory which is opened when you want to open a DICOM file, can be specified in the settings tab.

In the middle tab of the program, there are options to calculate integral and differential uniformity, although please note that pressing one of these buttons also triggers the other. Due to a high number of calculations in order to find the uniformities, this may take up to a minute, during which time the program may not respond. In the labels you will be able to see the overall differential or integral uniformity, whereas in the larger text box there will be details about each layer in the DICOM image. In the third tab, which contains the settings for the program, there are buttons linking to the about, help and licence sections. Add a bit more stuff here.
"""
        self.about_text = ctk.CTkTextbox(self, width=480, height=280)
        self.about_text.place(x=10, y=10)
        self.about_text.insert("1.0", text)
        self.about_text.configure(state="disabled")

########## MAIN ##########


if __name__ == "__main__":
    app = Gui()
    app.mainloop()
```

A.4 config.json

```
{
    "default_file_opening_directory": "C:/Users/Alex Toogood-Johnson/Documents",
    "default_file_saving_directory": "C:/Users/Alex Toogood-Johnson/Downloads",
    "colour_theme": "System",
    "colour": "blue",
    "convolution": [
```

```
        [
            1,
            2,
            1
        ],
        [
            2,
            4,
            2
        ],
        [
            1,
            2,
            1
        ]
    ],
    "crop_amount": 44,
    "fov_radius": 80,
    "step": 2
}
```

A.5 dicom_elements.json

A small sample of this file has been included to demonstrate it's format, however due to it's length of approximately 5000 lines, it has been heavily truncated.

```
{
    "00020000": "File Meta Information Group Length",
    "00020001": "File Meta Information Version",
    "00020002": "Media Storage SOP Class UID",
    "00020003": "Media Storage SOP Instance UID",
    "00020010": "Transfer Syntax UID",
    "00020012": "Implementation Class UID",
    "00020013": "Implementation Version Name",
    "00020016": "Source Application Entity Title",
    ...
}
```