Assignment 1

Ahmed Sharshar
22010334
ahmed.sharshar@mbzuai.ac.ae

1. Introduction

Medical imaging is an important field that focuses on the analysis of medical images to find features that can be important for diagnosis, prognosis, or treatment. Nowadays, there are many different aspects of work that can be done on these types of images. However, the basic image operations are very important to understand the data and improve its quality.

In this assignment, we used different simple operations on images to show their effect as following:

1. We tested the histogram equalization effect along with the operation cost (time complexity) on the Digital Imaging and Communications in Medicine (DICOM) image format.
2. we made a simple graphical user interface (GUI) to allow the user to manipulate images. We also used the Slicer application to construct 3D renderings of the lungs only.
3. Finally, we used the peak signal-to-noise ratio (PSNR) as a measurement for image quality, then used different filters to examine their effects on image quality.

2. Methodology

This assignment is divided into three main tasks as following:

2.1. Task1

In this task, we have to load 20 DICOM images; we used glob library for that.
Then perform, histogram equalization, a computer image processing method used to enhance contrast in photographs. This is achieved by extending the intensity range of the image and spreading out the most common intensity levels. When the useful data is represented by close contrast values, this method typically boosts the overall contrast of the photos. This makes it possible for regions with less local contrast to acquiring more contrast. [2] To achieve this, we made a simple function that makes the equalization:

```
def histogram_equalize(img):
    img_cdf, bin_centers = exposure.
    cumulative_distribution(img)
    new_image = np.interp(img, bin_centers,
    img_cdf)
    return new_image
```

After that, we normalized the output images to be between -1,1 by just applying a simple equation as follows:

```
normalized = (2 *((hist_img - hist_img.min()) /
    ( hist_img.max() - hist_img.min())) - 1)
```

Further, we computed the mean of each image pixel intensity using numpy mean function, then multiplied negative values by 0.9 and positive values by 0.8 and recomputed the mean.
In order to compute the complexity of the code, we measured the time it takes to run; using Time library, we ran the code 10 times and computed the mean and standard deviation of the total processing time for all images. We also computed the amount of CPU and RAM the program takes during processing and the specifications of CPU and RAM for the used device.

2.2. Task2

2.2.1 Task 2.1

In this task, We tried to make life easier for users by making a simple GUI to allow the user to choose one of the three orientation options (Axial, Coronal, Sagittal) so that the user can choose the orientation and the slices show as a gif with 0.1 seconds latency between image slices.
First, we showed only one image from the axial view using this code:

```
xray = pydicom.read_file("data/IMG0108.dcm")
dicom_img = xray.pixel_array
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.imshow(dicom_img, cmap='gray')
ax.axis('off')
```

To make the interface, we made a function called process() that takes only one parameter, a flag; that we will use later, this function contains loading the images as a stack using imageio.volread as follows:

```
vol = imageio.volread("path", "DICOM")
n0, n1, n2 = vol.shape
```

Then we used the number of slices we got from n0,n1, and n2 to make a loop that separated each of the three orientations on a list, we repeated it for Coronal and Sagittal.

```
axial = []
for i in range(0,n0-1):
    ax = vol[i,:,:]
    axial.append(ax)
```

Then we create a gif based on the signal from the button that the user sends; for example, if axial is selected, a flag with value 1 is now activated, making a gif for the axial view and save it as follows:

```
if x ==1:
    imgs = axial
    imgs = [Image.fromarray(i) for i in imgs]
    imgs[0].save("axial.gif", save_all=True,
    append_images=imgs[1:], duration=100, loop=0)
```

Then for the GUI, we used tkinter library; we made a function for each view that is related to each button, each function load gif and view it on a window wafter reciving a signal from its button, for example axial function is as follows:

```
def play_axial():
    process(1)
    global img
    img = Image.open("axial.gif")
    lbl = Label(root)
    lbl.place(x=0 , y=0)
    for img in ImageSequence.Iterator(img):
        img = ImageTk.PhotoImage(img)
        lbl.config(image=img)
        root.update()
        time.sleep(0.1)
    root.after(0,play_axial)
```

We made four buttons, one for each view and one for exit, a sample button with some decoration as follows:

```
Button(root , text = 'Axial' , command =
    play_axial, height = 1 , width = 7, bg = '
    black', fg = 'white', font = "Times 14 bold "
     ).place(x = 400 , y = 600)
```

This GUI depends on creating a gif, saving it then viewing it. There are smarter ways as taking the gif without explicitly saving it or showing images one by one using a timer show, but these did not work for me.

For the test, please run the code locally (on PC) or in a directory that allows you to save the gif (shared folders need permissions). The full code is attached to the assignment.

2.2.2 Task 2.2

The remaining part of the task required creating a 3D rendering of the lungs-only region using Slicer program. To learn more about the program, I tried a different method than the one we slightly discussed in the lab [1], the steps are:

1. Open slicer; from "File", choose "add data", then choose the data file.
2. From the navigation bar, choose "volume rendering", which allows us to deal with the data and generate 3d data.
3. Under "Display" in the left bar, choose preset; it contains predefined sets suitable for parts of the human body; here, as the data for lungs, we choose "CT-Air".
4. Under "Display" in the left bar, awake "Display ROI" eye, which shows a box that represents the Region of interest (ROI) in each view.
5. Select ROI in each image that only shows the lunges; the closer you choose the ROI around lunges, the clearer the output.
6. save the project from "File" select "save data".

The output project is attached along with the assignment.

2.3. Task3

In this task, we use peak signal-to-noise ratio (PSNR), which calculates the peak signal-to-noise ratio between two pictures, expressed in decibels. This ratio is used to compare the original and compressed images' quality. The quality of the compressed or rebuilt image improves with increasing PSNR. for example, if we compare an image with itself, it should give us the highest PSNR. [3]. First, we need to load 10 videos and a text file containing video names along with an index for frames one and 2. We need to match video names with the text to get frames indices the extract these frames from the videos. To do so, first, we loaded the videos:

```
import os
filelist=os.listdir('path')
for fichier in filelist[:]:
    if not(fichier.endswith(".avi")):
        filelist.remove(fichier)
print(filelist)
```

Then we loaded the text file as pandas datarame to make it easy to deal with; we used "FN" column, which contains video names, as an index for the dataframe.

```
import pandas as pd
df = pd.read_csv('Frames.txt')
df = df.set_index('FN')
```

Then we do the matching, get the index of the frame and load the corresponding frame from the video; we also use PNSR from Open-cv library to compute the PSNR between frames 1 and 2:

```
for i in filelist:
    for name, row in df.iterrows():
        if i == name:
            print (name)
            cap = cv2.VideoCapture(name)
            first_frame = (df.loc[name][' Frame1'
    ])
```

```
            second_frame = (df.loc[name][' Frame2'])
            shot_1 = cap.set(cv2.CAP_PROP_POS_FRAMES, first_frame)
            res1, frame1 = cap.read()
            shot_2 = cap.set(cv2.CAP_PROP_POS_FRAMES, second_frame)
            res2, frame2 = cap.read()
            psnr = cv2.PSNR(frame1, frame2)
            print("Frame1 & Frame2 PSNR  = " , psnr)
```

Then we applied different filter kernels on the second frame; we used the gauss filter, median filter, box filter and mode filter to examine the effect of each filter on enhancing image quality than increasing PSNR.

Guass filter is a two-dimensional convolution operator used to 'blur' pictures and gaussian noise with kernel values taken from a normal distribution.

The Box Filter is a low-pass filter that removes features, noise, and borders from pictures by making each output pixel the average of the surrounding ones.The Mode filter removes noise from a picture by replacing pixels with the most commonly occurring pixel value from a given window size.

The median filtering technique is a nonlinear approach for removing noise from photographs. It is computed by first sorting all of the pixel values in the window into numerical order and then replacing the pixel under consideration with the middle (median) pixel value.
All the kernel sizes used in this experiment are $3 * 3$ as this produced the best results. The code for this part as follows:

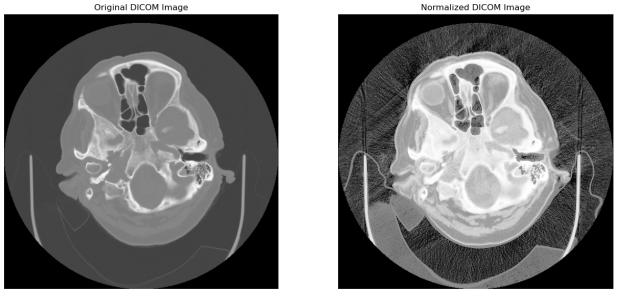
```
for i in filelist:
    for name, row in df.iterrows():
        if i == name:
            print (name)
            cap = cv2.VideoCapture(name)
            second_frame = (df.loc[name][' Frame2'])
            shot_2 = cap.set(cv2.CAP_PROP_POS_FRAMES, second_frame)
            res2, frame2 = cap.read()
            image = Image.fromarray(frame2)
            shot_1 = cap.set(cv2.CAP_PROP_POS_FRAMES, first_frame)
            res1, frame1 = cap.read()

            gauss = image.filter(ImageFilter.GaussianBlur(radius=3)) #gauss
            gauss = np.array(gauss)
            psnr = cv2.PSNR(frame1, gauss)
            print("Gaussian Filter PSNR  = " , psnr)

            median = image.filter(ImageFilter.MedianFilter(size = 3)) #median
            median = np.array(median)
            psnr = cv2.PSNR(frame1, median)
            print("Median Filter PSNR  = " , psnr)

            box = image.filter(ImageFilter.BoxBlur(1)) #3*3 box
            box = np.array(box)
            psnr = cv2.PSNR(frame1, box)
            print("Box Filter PSNR  = " , psnr)

            mode = image.filter(ImageFilter.ModeFilter(size=3)) #mode
            mode = np.array(mode)
            psnr = cv2.PSNR(frame1, mode)
            print("mode Filter PSNR  = " , psnr)
```

3. Results

In this section, we show the results of each task.

3.1. Task1

After loading the images, we did histogram equalization; figure 1 shows a sample output of the equalized images. It shows that the pixel intensities are now stretched along the image, and it becomes clearer.



(a) Original DICOM Image (b) Equalized DICOM Image

Figure 1. Histogram Equalization Sample Output

Figures 2,3,4 represent the histogram of a sample image of the original images, equalized images and the images after normalizing the pixel values to range between -1 and 1, respectively. It shows how the pixel intensities stretched over the range after equalization, improving the image. Then the pixel intensity range became between -1 and 1 after normalizing it.

Next, we computed the mean of each image by using numpy mean, we sorted the mean values to be: [-0.2408, -0.2396, -0.2370, -0.2122, -0.2110, -0.2082, -0.2024, -0.2003, -0.1799, 0.0018, 0.0058, 0.0071, 0.0085, 0.0104, 0.0118, 0.0123, 0.0137, 0.0158, 0.0165, 0.0608]
After that, we multiplied negative values with 0.9 and positive values with 0.8 and recomputed the mean again; we sorted the values to be: [-0.2356, -0.2347, -0.2324, -0.2106, -0.2096, -0.2073, -0.2021, -0.2000, -0.1818, -0.0235, -0.0197, -0.0187, -0.0174, -0.0157, -0.0145, -0.0139, -0.0127, -0.0108, -0.0102, 0.02962]
There are some notes on this point; when we reproduced these values using different loading libraries as open-cv read method or math.mean to calculate the mean, the mean

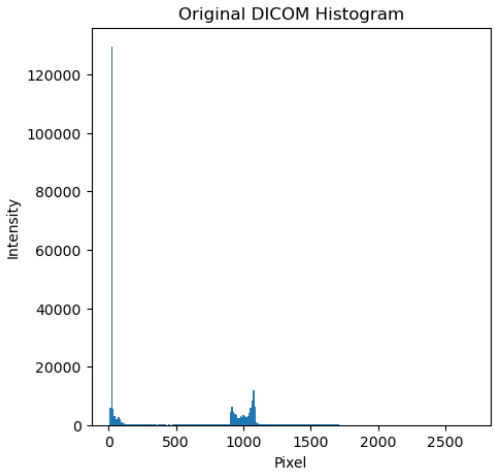


Figure 2. Original Histogram

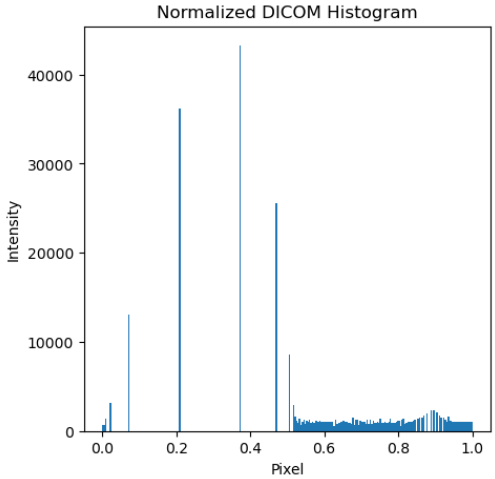


Figure 3. Equalized Histogram

values got different values. The reason may be because of data types, as different data types have different sensitivity and ranges. This can happen while reading the images, calculating the mean, or both.

We calculated the time the code takes after loading the images and till the end. We repeated this process 10 times and got the time's average and standard deviation(Sd). The mean was around 4 seconds with sd = $\pm$ 0.1. As the processing time also depends on the hardware specs and the background running programs, we measured the amount of memory (RAM) the program needs = 82 MB, out of the device RAM, which = 16 GB. For the CPU, it takes around 12% from the CPU, which is 11th Gen Intel(R) Core(TM) i7-11800H with base frequency = 2.30 GHz. This is a reasonable performance for such a program.

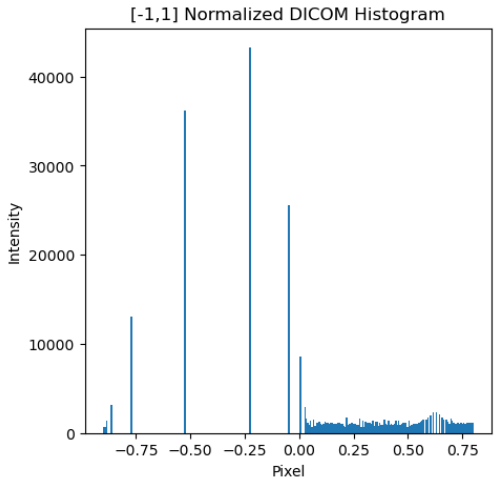


Figure 4. [-1,1] Normalized Histogram

3.2. Task2

3.2.1 Task2.1

Figure 5 shows the interface for task 2 with axial slides runs automatically on it, with 0.1 seconds latency between slides.

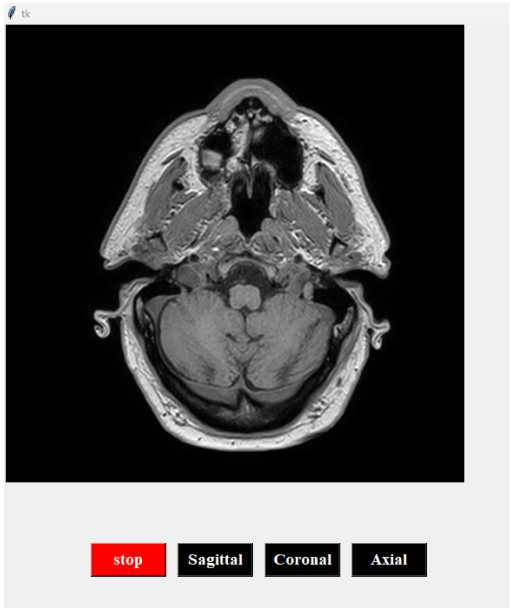
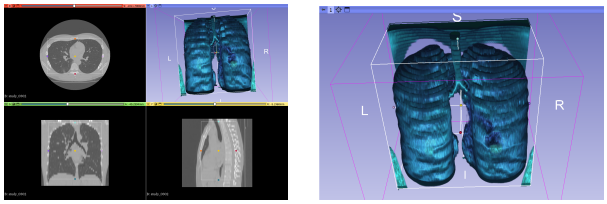


Figure 5. Screenshot of The GUI

3.2.2 Task2.2

Figure 6 shows all the lunge's perspectives along with 3d rendered lungs and another perspective for only 3d Lungs.



(a) All The Views (b) 3d Lungs Only

Figure 6. Slicer Output

3.3. Task3

As Table 1 shows, the PSNR value is the largest between frames 1 and 2 in video 55 with PSNR = 19.52, then video 87 with PSNR = 19.44. This is because these video frames have more similarities than the others.

Table 1. PSNR Between Frame 1 and Frame 2 In Each Video

Video Name	PSNR
video4.avi	16.49
video7.avi	17.90
video18.avi	18.40
video27.avi	16.88
video49.avi	16.07
video55.avi	19.52
video59.avi	10.90
video78.avi	17.27
video87.avi	19.44
video88.avi	15.40

For the second task, we filtered the second frame using different filters and calculated PSNR between the filtered frame and frame 1. From Table 2, we can discover that the mode filter is decreasing PSNR values for all values with respect to the first experiment. The reason is that mode will almost substitute pixels with 0 as the frames are almost black, reducing the similarities between frames.

We can find that the Gaussian filter is almost the best filter as it increases the PSNR in all 10 videos and does the best in 6 of them. This is because most of the noise in the frame is gaussian noise which this filter can reduce its effectiveness.

Box filter did great as well; kernel size = 3 can reduce the noise effectively without blurring the images much. The median filter did the same but not with the same effectiveness.

Increasing the kernel size for both gaussian and box filters increases the images' smoothness and reduces the noise. However, it also adds blur to the images, which decreases PSNR.

Table 2. PSNR Between Original Frame 2 and Filtered One In Each Video

Video Name	Gaussian	Median	Box	Mode
video4.avi	**17.18**	16.84	16.98	16.23
video7.avi	21.24	21.51	**21.61**	21.19
video18.avi	19.08	19.21	**19.51**	18.72
video27.avi	**19.56**	19.13	19.25	18.56
video49.avi	22.58	24.20	**24.44**	23.96
video55.avi	**20.53**	19.75	19.92	19.36
video59.avi	**12.91**	12.20	12.37	12.04
video78.avi	18.47	18.51	**18.60**	18.28
video87.avi	**21.75**	21.35	21.71	20.94
video88.avi	**16.18**	15.59	15.69	15.29

References

[1] Slicer Documentation. Modules:roimodule-documentation-3.6. 2

[2] Wu Zhihong and Xiao Xiaohong. Study on histogram equalization, 2011. 1

[3] Zhendong Zhuang and Yang Xue. Image quality metrics: Psnr vs. ssim. pages 2366–2369, 2010. 2