Task 1: Normalization and counting (3/10 marks)
Write a Python code to load a set of 20 DICOM images. Normalize pixel values to [0 - 255]. Print the number of pixels in each image which has a pixel intensity greater than 250. Report the list of 20 numbers sorted in ascending order

Code

```
def load_file_paths(path_to_dir, file_extension):
    return glob.glob(os.path.join(path_to_dir, "*" + file_extension))

def load_dicom(dcm_path):
    img = pydicom.read_file(dcm_path)
    img = img.pixel_array
    return img

def normalize_image(img):
    img = (img - np.min(img)) / (np.max(img) - np.min(img))
    img = img * 255
    return img


PATH_TO_DIR = "./assignment_1_data/assignment_1/Task_1"

# load dicom file paths from directory
file_paths = load_file_paths(PATH_TO_DIR, ".dcm")

float_counts_greater_than_250 = []
for file_path in file_paths:
    img = load_dicom(file_path)

    # normalize and count pxls larger than 250 for datatype float64
    float_img = normalize_image(img)
    float_count_greater_than_250 = len(float_img[float_img>250])
    float_counts_greater_than_250.append(float_count_greater_than_250)

sort_float_counts_greater_than_250 = sorted(float_counts_greater_than_250)
print("Data type:", float_img.dtype)
print(sort_float_counts_greater_than_250)

### EXTRA (data type investigation) ###
int_counts_greater_than_250 = []
round_int_counts_greater_than_250 = []
ceil_int_counts_greater_than_250 = []

for file_path in file_paths:
    img = load_dicom(file_path)
    float_img = normalize_image(img)

    # normalize and count pxls larger than 250 for datatype uint8
    int_img = normalize_image(img).astype("uint8")
    int_count_greater_than_250 = len(int_img[int_img>250])
```

```
        int_counts_greater_than_250.append(int_count_greater_than_250)

        # normalize and count pxls larger than 250 for datatype uint8
        # (after rounding float values to the nearest int)
        round_int_img = np.round(float_img, 0).astype("uint8")
        round_int_count_greater_than_250 = len(round_int_img[round_int_img>250])
        round_int_counts_greater_than_250.append(round_int_count_greater_than_250)

        # normalize and count pxls larger than 250 for datatype uint8
        # (after rounding UP float values to the nearest int)
        ceil_int_img = np.ceil(float_img).astype("uint8")
        ceil_int_count_greater_than_250 = len(ceil_int_img[ceil_int_img>250])
        ceil_int_counts_greater_than_250.append(ceil_int_count_greater_than_250)

sort_int_counts_greater_than_250 = sorted(int_counts_greater_than_250)
sort_round_int_counts_greater_than_250 =
sorted(round_int_counts_greater_than_250)
sort_ceil_int_counts_greater_than_250 =
sorted(ceil_int_counts_greater_than_250)

print("\nEXTRA")
print("Data type:", int_img.dtype)
print(sort_int_counts_greater_than_250)
print("\nData type:", round_int_img.dtype, "(round)")
print(sort_round_int_counts_greater_than_250)
print("\nData type:", ceil_int_img.dtype, "(ceil)")
print(sort_ceil_int_counts_greater_than_250)
```

Brief description

Code summary:

1. Load all dicom file paths from directory.
2. For each file path:
    - Read the dicom file
    - Normalize the image to [0, 255]
    - Count the number of pixels greater than 250
3. Sort the number of pixels greater than 250 in ascending order.

Note: I repeated Steps 2-3 for different data types. Initially, I found that the number of pixels greater than 250 was different (i.e. higher) for float64 than uint8. I thought this was due to a simple rounding of the floats to integer. However, the pixel counts for uint8 (after rounding to the nearest integer) was still different from the pixel counts of float64. Upon investigation, I found that in Python, uint8 rounds up the pixel values to the nearest integer (i.e. float_array.astype("uint8") == np.ceil(float_array).astype("uint8"); a behavior different from MATLAB).

Results or screen shots

```
Data type: float64
[2, 3, 5, 5, 6, 8, 8, 9, 9, 11, 12, 15, 20, 20, 21, 23, 23, 34, 132, 154]

EXTRA
Data type: uint8
```

```
[2, 2, 3, 4, 5, 5, 5, 6, 7, 8, 11, 15, 16, 18, 20, 20, 21, 27, 129, 152]

Data type: uint8 (round)
[2, 3, 5, 5, 5, 6, 7, 7, 8, 10, 11, 15, 18, 19, 20, 21, 22, 31, 131, 153]

Data type: uint8 (ceil)
[2, 3, 5, 5, 6, 8, 8, 9, 9, 11, 12, 15, 20, 20, 21, 23, 23, 34, 132, 154]
```

Task 2: 3D image visualization (3/10 marks)
Task 2.1 Write a Python function to:
- load one brain MRI.
- Visualize slices of the MRI scan one by one with 0.2 second between each slice.
- Report your code and one screen shot of a frame.

Code

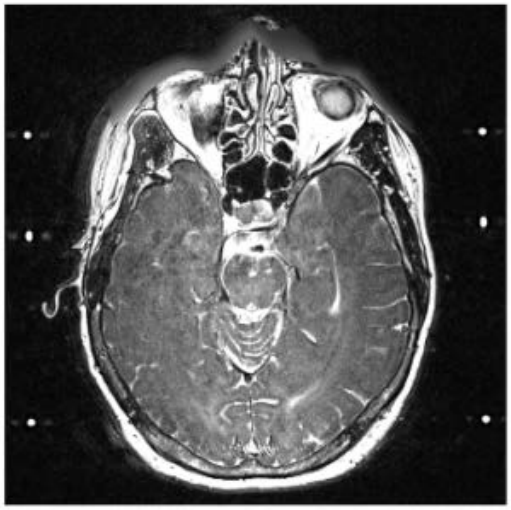
```
from matplotlib import animation, rc
rc("animation", html="jshtml")

def create_animation(ims, interval):
'''
    inputs:
        ims: image slices as an array
        interval: frame interval in milliseconds

    output: frame animation
'''

    fig = plt.figure(figsize=(6, 6))
    plt.axis('off')
    im = plt.imshow(ims[0], cmap = 'gray')
    plt.close()

    def animate_func(i):
        im.set_array(ims[i])
        return im

    return animation.FuncAnimation(fig, animate_func, frames = len(ims),
interval = interval)


PATH_TO_MRI =
"./assignment_1_data/assignment_1/Task_2/crossmoda_229_hrT2.nii.gz"

mri = nb.load(PATH_TO_MRI)
mri = mri.get_fdata()
mri = np.transpose(mri, (2, 1, 0))
create_animation(mri, 200)
```

Brief description
Code summary:
1. Load NIfTI file.
2. Get pixel array.
3. Reshape array from (y,x,z) to (z,x,y).
4. Visualize slices with 0.2 second (200 milliseconds) between each slice.

Note: From the NIfTI file header, we can see that the MRI grid spacing is 0.546875 mm in the x- and y-directions, and 1.5 mm in the z-direction. No information on time is provided.

	Value	**Description**[1]
pixdim	[1. 0.546875 0.546875 1.5 0. 0. 0. 0.]	Grid spacings (unit per dimension) ***pixdim[0]:*** qfac value used in the Hadamard product to define the transformation from voxel to world space via a rotation matrix R ***pixdim[1:4]:*** voxel dimension in x,y,z,t
xyzt_units	2	Units for pixdim[1:4] ***Code*** ***Unit*** 1 Meter (m) 2 Milimeter (mm) 3 Micron (µm) 8 Seconds (s) 16 Miliseconds (ms) 24 Microseconds (µs) 32 Hertz (Hz) 40 Parts-per-million (ppm) 48 Radians per second (rad/s)

Results or screen shots



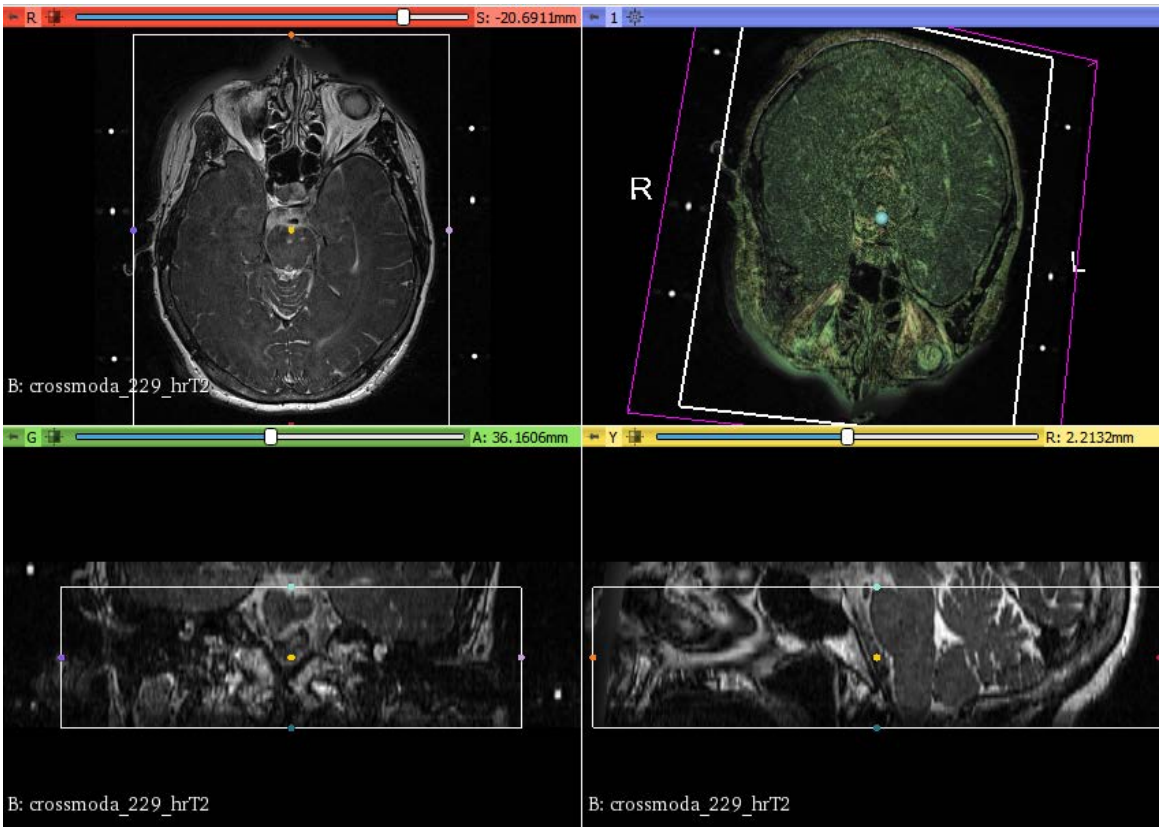
[1] https://brainder.org/2012/09/23/the-nifti-file-format/

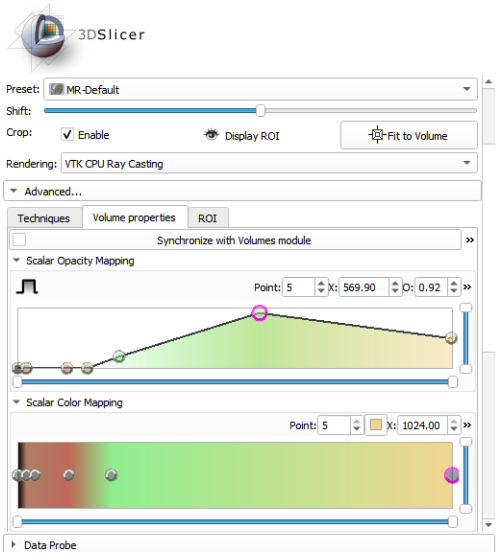
Task 2.2: Load the MRI scan using Slicer, investigate different rendering approaches to visualize the brain internal structures. Report two screen shots for 3D visualization of the brain.

Brief description

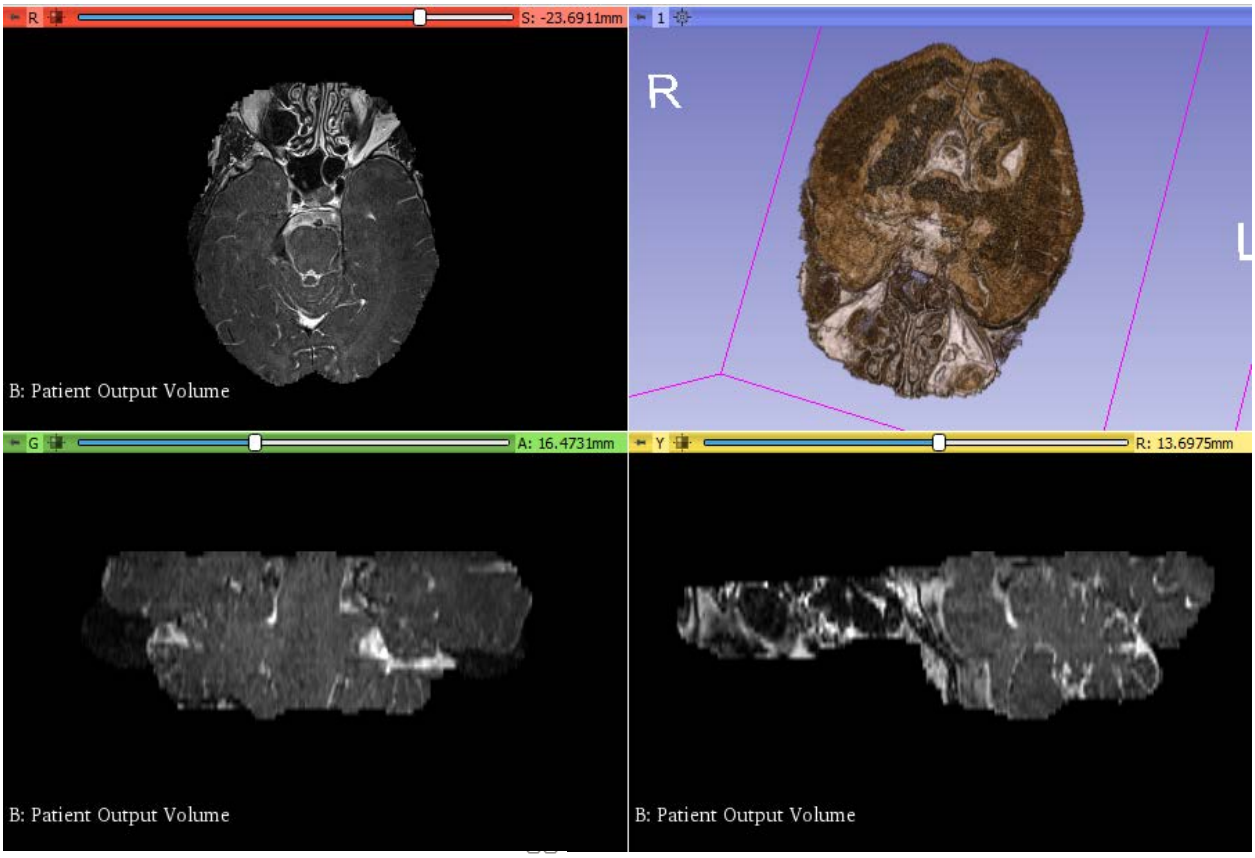
1. Brain visualization using MR-Default volume rendering with a modified color scheme.
2. Brain segmentation using the Swiss Skull Stripper module[2].

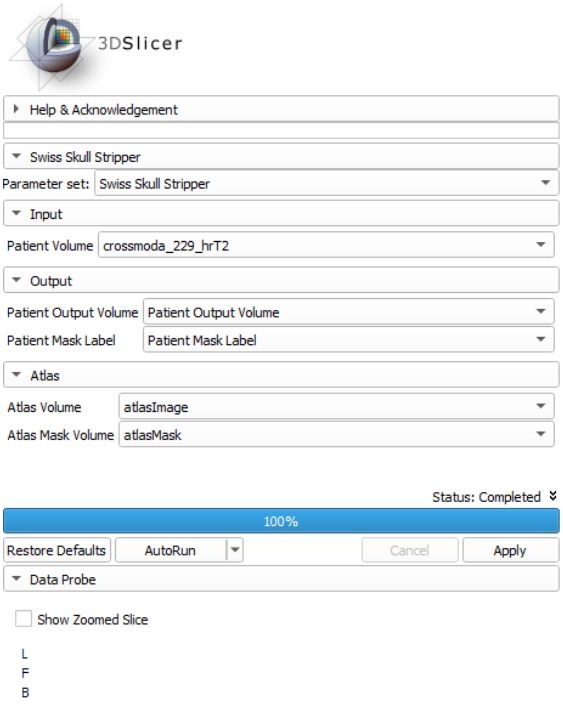
Results or screen shots





[2] https://www.slicer.org/wiki/Documentation/Nightly/Modules/SwissSkullStripper


R
S: -23.6911mm
B: Patient Output Volume
1
R
G
A: 16.4731mm
B: Patient Output Volume
Y
R: 13.6975mm
B: Patient Output Volume


3DSlicer
Help & Acknowledgement
Swiss Skull Stripper
Parameter set: Swiss Skull Stripper
Input
Patient Volume crossmoda_229_hrT2
Output
Patient Output Volume Patient Output Volume
Patient Mask Label Patient Mask Label
Atlas
Atlas Volume atlasImage
Atlas Mask Volume atlasMask
Status: Completed
100%
Restore Defaults
AutoRun
Cancel
Apply
Data Probe
Show Zoomed Slice
L
F
B

Task 3: Image filtering and quality assessment (4/10 marks)
Write a Python code to load a set of 2D+time echocardiography scans provided as 10 video files. You are also given a text file called “Frames.txt” containing the name of each video and the indices of 2 frames within the video (first frame is at position zero).

Task 3.1: Compute PSNR between the two frames on each video.
Report the code and the name of the video with the highest PSNR.

Code

```
PATH_TO_DIR =
"/home/muhammadridzuan/Documents/assignment_1_data/assignment_1/Task_3/"

def get_frame_from_vid(vid_name, frame):
    vid_path = glob.glob(os.path.join(PATH_TO_DIR, vid_name))
    vidcap = cv2.VideoCapture(vid_path)

    vidcap.set(cv2.CAP_PROP_POS_FRAMES, frame)
    _, img = vidcap.read()

    # convert img to grayscale dimension (X,Y) from (X,Y,3)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    return img

def PSNR(vid_name, frame1, frame2):
    img1 = get_frame_from_vid(vid_name, frame1)
    img2 = get_frame_from_vid(vid_name, frame2)

    return cv2.PSNR(img1, img2)

def plot(vid_name, frame1, frame2):
    fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize=(10, 10))
    fig.subplots_adjust(top = 1, bottom = 0.5)
    fig.suptitle(vid_name)

    img1 = get_frame_from_vid(vid_name, frame1)
    img2 = get_frame_from_vid(vid_name, frame2)

    ax1.imshow(img1, cmap="gray")
    ax1.set_title("Frame {}".format(frame1))
    ax2.imshow(img2, cmap="gray")
    ax2.set_title("Frame {}".format(frame2))


# read the txt file as a dataframe
frame_df =
pd.read_csv("/home/muhammadridzuan/Documents/assignment_1_data/assignment_1/T
ask_3/Frames.txt")
```

```
# calculate PSNR between the two frames, and update the result in the
dataframe
for idx, row in frame_df.iterrows():
    frame_df.loc[idx, "PSNR"] = PSNR(row["FN"], row[" Frame1"], row["
Frame2"])

# print the row where frame_df["PSNR"] is equal to the maximum
frame_df["PSNR"]
print(frame_df[frame_df["PSNR"] == np.max(frame_df["PSNR"])])

# extra: plot the frames for each image for visualization
for idx, row in frame_df.iterrows():
    plot(row["FN"], row[" Frame1"], row[" Frame2"])
```

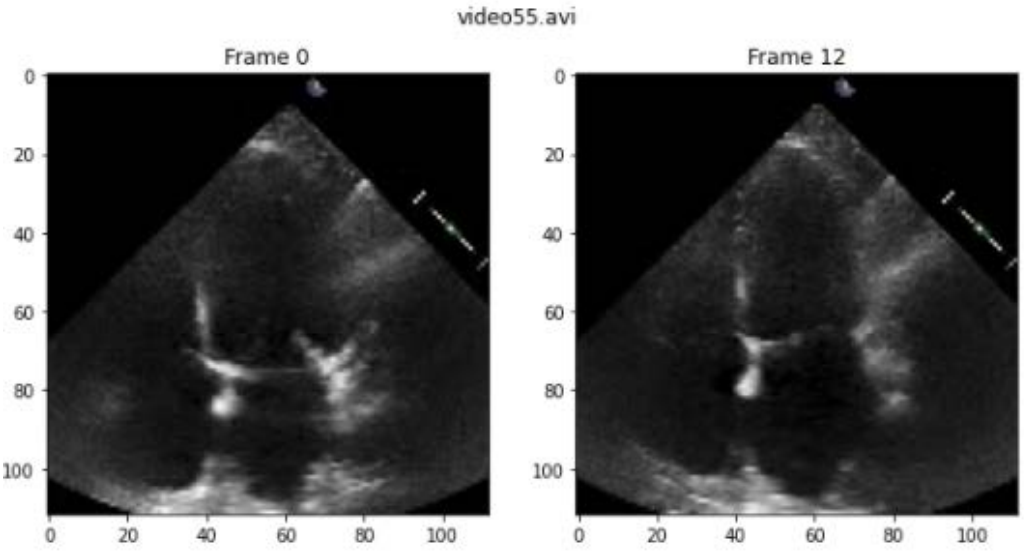
Brief description

Code summary:

1. Load Frames.txt as a dataframe.
2. Capture the relevant frames from each video as arrays. Convert the frames to grayscale.
3. Calculate the PSNR between the two frames. Update the result in the dataframe.
4. Print the row with the highest PSNR value.

Results or screen shots

	FN	Frame1	Frame2	PSNR
5	video55.avi	0	12	19.516534



Note: The code plots the frames for all files, but only video55.avi is shown here.

Task 3.2: Improve the quality on the first frame only (in Frames.txt) in each video by

- Add Salt and Pepper noise to 0.005 of pixels in each frame
- Then, add Speckle noise with variance of 0.01 to the noisy frame
- Filter the noisy frames using different filtering methods such as median, Gaussian, etc.…
- Compute PSNR between the reference frame and the filtered frame from each video.
- Report your code and the experiments you did. Discuss briefly which filtering method provided the best PSNR and why?

Code

```
frame_df2 = frame_df.copy()

for idx, row in frame_df2.iterrows():
    frame1 = get_frame_from_vid(row["FN"], row[" Frame1"])
    frame1 = frame1.astype(np.uint8) #need to ensure ori & noisy image are of
the same type

    # - Add Salt and Pepper noise to 0.005 of pixels in each frame
    salt_pepper = random_noise(frame1, mode='s&p', seed=37, clip=True, amount
= 0.005, salt_vs_pepper = 0.5)

    # - Then, add Speckle noise with variance of 0.01 to the noisy frame
    speckle_salt_pepper = random_noise(salt_pepper, mode='speckle', seed=37,
clip=True, var = 0.01)
    speckle_salt_pepper *= 255
    speckle_salt_pepper = speckle_salt_pepper.astype(np.uint8)

    # - Filter the noisy frames using different filtering methods such as
median, Gaussian, etc.…
    med = cv2.medianBlur(speckle_salt_pepper, ksize=3).astype(np.uint8)
    gauss3 = cv2.GaussianBlur(speckle_salt_pepper, ksize=(3,3), sigmaX=1,
sigmaY=1).astype(np.uint8)
    gauss5 = cv2.GaussianBlur(speckle_salt_pepper, ksize=(5,5), sigmaX=1,
sigmaY=1).astype(np.uint8)
    bilateral = cv2.bilateralFilter(speckle_salt_pepper, d=9, sigmaColor=75,
sigmaSpace=75)
    mean = cv2.blur(speckle_salt_pepper, ksize=(3,3)).astype(np.uint8)
    # selective median
    selective_med = speckle_salt_pepper.copy()
    speckle_salt_pepper_region = (speckle_salt_pepper == 0) |
(speckle_salt_pepper == 255)
    selective_med[speckle_salt_pepper_region] =
med[speckle_salt_pepper_region]

    # - Compute PSNR between the reference frame and the filtered frame from
each video.
    frame_df2.loc[idx, "PSNR_NOISY"] = cv2.PSNR(frame1, speckle_salt_pepper)
    frame_df2.loc[idx, "PSNR_BILATERAL"] = cv2.PSNR(frame1, bilateral)
    frame_df2.loc[idx, "PSNR_MED"] = cv2.PSNR(frame1, med)
    frame_df2.loc[idx, "PSNR_GAUSS3"] = cv2.PSNR(frame1, gauss)
    frame_df2.loc[idx, "PSNR_GAUSS5"] = cv2.PSNR(frame1, gauss5)
```

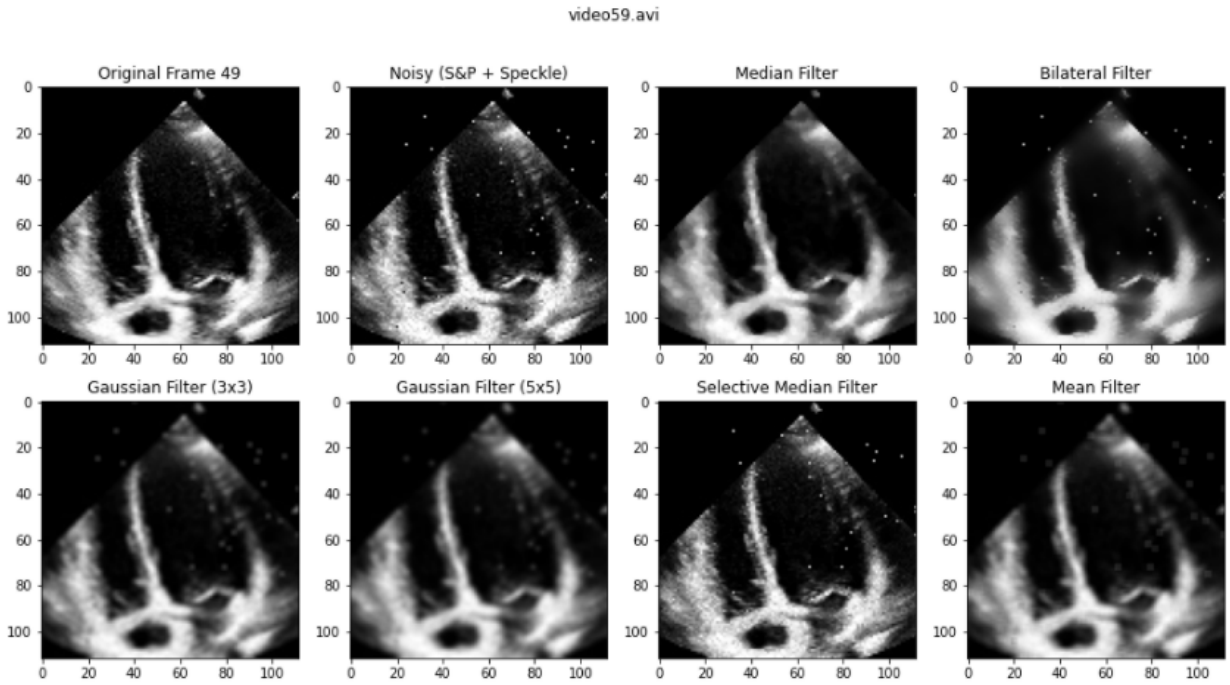
```
    frame_df2.loc[idx, "PSNR_SELECTIVE_MED"] = cv2.PSNR(frame1, selective_med)
    frame_df2.loc[idx, "PSNR_MEAN"] = cv2.PSNR(frame1, mean)

    # Plot for qualitative inspection
    fig, ax = plt.subplots(nrows = 2, ncols = 4, figsize=(16,8))
    fig.suptitle(row["FN"])

    ax[0,0].imshow(frame1, cmap="gray")
    ax[0,0].set_title("Original Frame {}".format(row[" Frame1"]))
    ax[0,1].imshow(speckle_salt_pepper, cmap="gray")
    ax[0,1].set_title("Noisy (S&P + Speckle)")
    ax[0,2].imshow(med, cmap="gray")
    ax[0,2].set_title("Median Filter")
    ax[0,3].imshow(bilateral, cmap="gray")
    ax[0,3].set_title("Bilateral Filter")

    ax[1,0].imshow(gauss3, cmap="gray")
    ax[1,0].set_title("Gaussian Filter (3x3)")
    ax[1,1].imshow(gauss5, cmap="gray")
    ax[1,1].set_title("Gaussian Filter (5x5)")
    ax[1,2].imshow(selective_med, cmap="gray")
    ax[1,2].set_title("Selective Median Filter")
    ax[1,3].imshow(mean, cmap="gray")
    ax[1,3].set_title("Mean Filter")
```

Brief description

I used 5 classical filters: bilateral, median, mean, Gaussian (3x3), Gaussian (5x5), and selective median. The best PSNR value was obtained using the selective median filter, where the median filter is applied only to regions with pixel values 0 or 255. This filter helps to eliminate salt and pepper noises, however it does not eliminate the speckles. Qualitatively, the median filter does a better job at eliminating the speckle noise, however it does so at the expense of blurring the borders.

Results or screen shots

	FN	Frame1	Frame2	PSNR	PSNR_NOISY	PSNR_BILATERAL	PSNR_MED	PSNR_GAUSS3	PSNR_GAUSS5	PSNR_SELECTIVE_MED	PSNR_MEAN
0	video59.avi	49	65	10.906271	25.031039	23.684072	24.641509	10.275835	24.215457	25.990168	24.015168
1	video4.avi	28	43	16.49099	25.548365	24.205725	24.535163	11.764612	24.312799	26.269433	24.132884
2	video88.avi	26	43	15.406468	25.915996	24.703328	25.661344	13.926028	25.877890	27.217755	25.454171
3	video27.avi	80	97	16.879856	26.042401	24.822374	25.891797	14.645953	26.282756	27.903802	25.815721
4	video87.avi	44	63	19.445555	26.129635	25.441638	27.010954	15.266163	27.131929	27.986180	26.734889
5	video55.avi	0	12	19.516537	26.380885	26.052383	27.522949	15.553751	27.558783	27.722874	27.266836
6	video18.avi	45	58	18.40073	25.913408	24.144351	25.379062	13.956587	25.264416	27.480202	24.988719
7	video7.avi	44	53	17.909802	26.070789	25.074095	25.284121	13.611577	25.776403	26.986816	25.279502
8	video78.avi	135	146	17.278485	25.923906	25.076892	27.244533	13.306137	26.449353	27.719392	26.198874
9	video49.avi	156	176	16.074259	26.118514	24.810005	25.967573	26.526453	25.909506	27.477735	25.682747


video59.avi
Original Frame 49
Noisy (S&P + Speckle)
Median Filter
Bilateral Filter
Gaussian Filter (3x3)
Gaussian Filter (5x5)
Selective Median Filter
Mean Filter