

FACULTY OF COMPUTING

SEMESTER 1/2223

SECV3213- FUNDAMENTAL OF IMAGE PROCESSING

SECTION 1

Assignment 2: Image Restoration

Lecturer: Dr Md Sah Hj Salam

Group 3

Name	Matric No
Fatin Aimi Ayuni Binti Affindy	A20EC0190
Bilkis Musa	A20EC0233

TABLE OF CONTENT

Introduction	1
Problem Analysis	2
Proposed Solution 1	3
Process flow of proposed solution	3
Explanation	4
Results	6
Analysis	8
Proposed Solution 2	9
Process flow of proposed solution	9
Explanation	11
Results	14
Analysis	16
Analysis of Proposed Solution	17
Conclusion	20
References	21

Introduction

Image restoration is a technique of image processing which is used to recover and solve the degraded image whether by deblurring or removing the noise. There are many techniques that can be used to remove the noise and restore the image such as median blur, mean, gaussian blur, high pass filter and low pass filter. Based on the knowledge we had learnt along this semester, in this assignment, we proposed 2 solutions to restore the noise image into the original image by using the set of images provided, comparing and evaluating the similarity and analyzing those 2 proposed solutions. These 2 solutions have different combinations of filters to produce the highest possible similarity of those three images.

Problem Analysis

We were assigned to restore three different images in this assignment. The first image is the brain, the second image is the handgun and the third image is the baby.

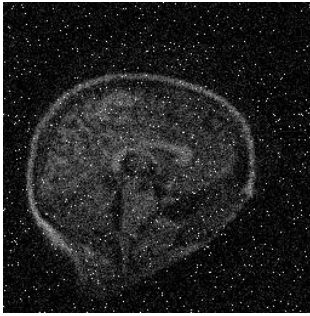


Figure 1: The brain

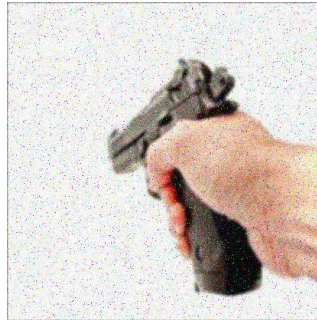


Figure 2: The handgun

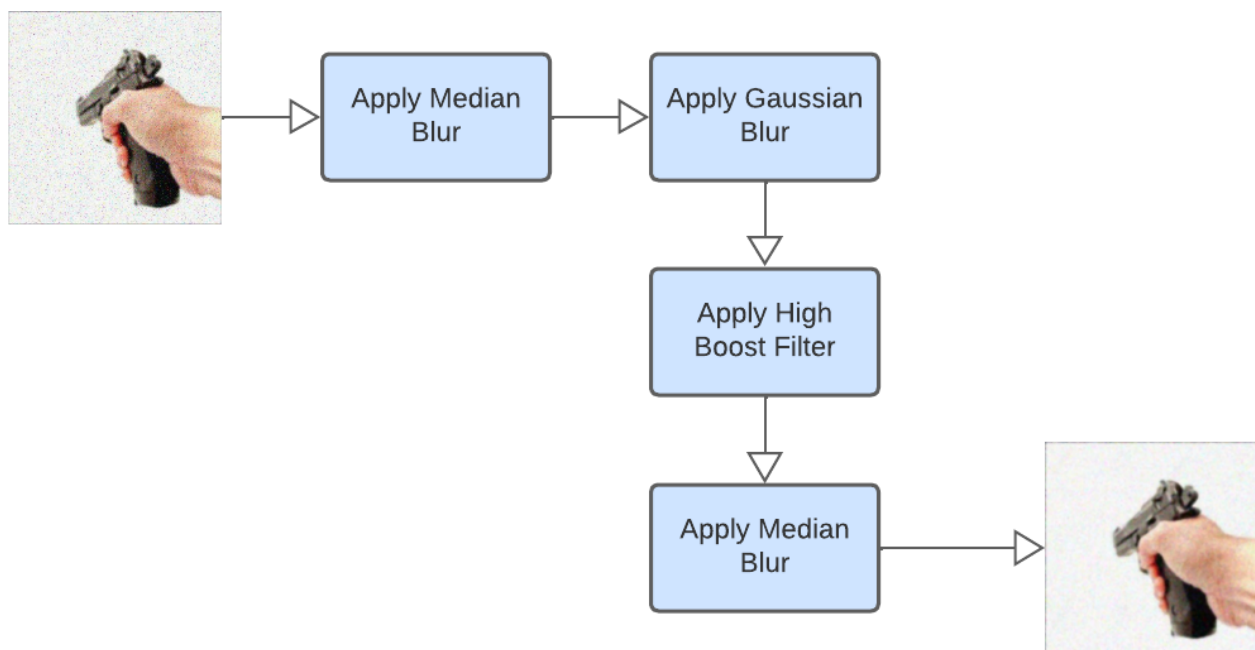
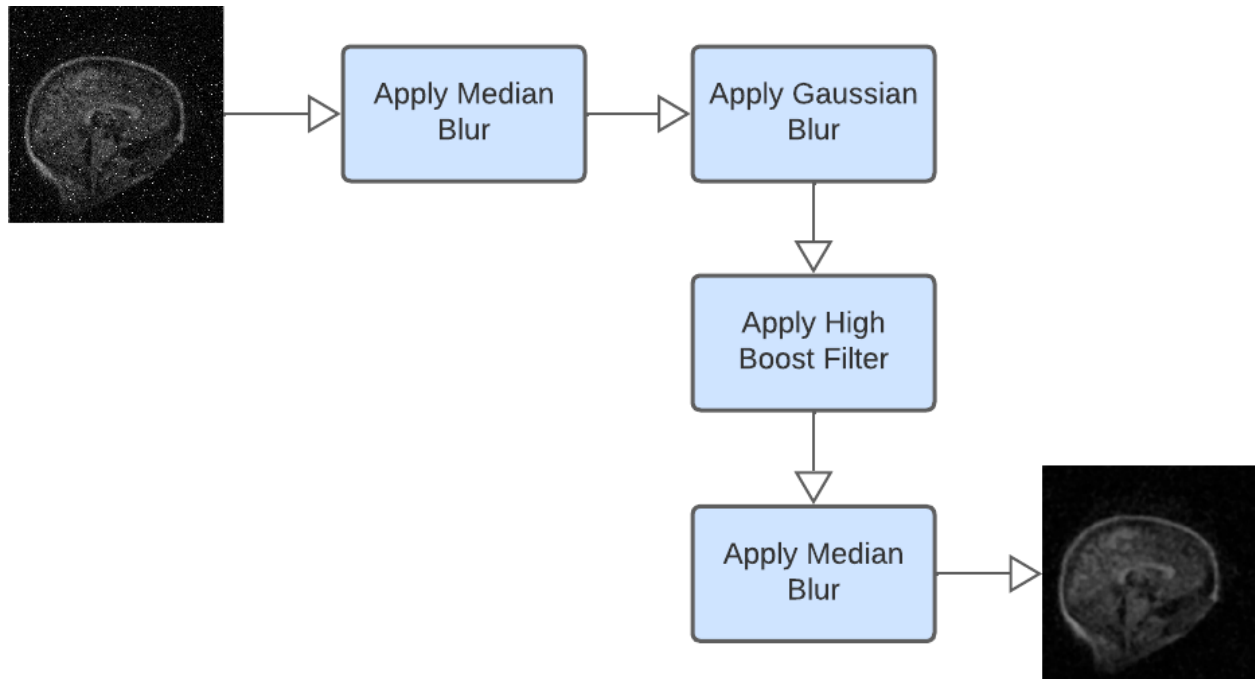


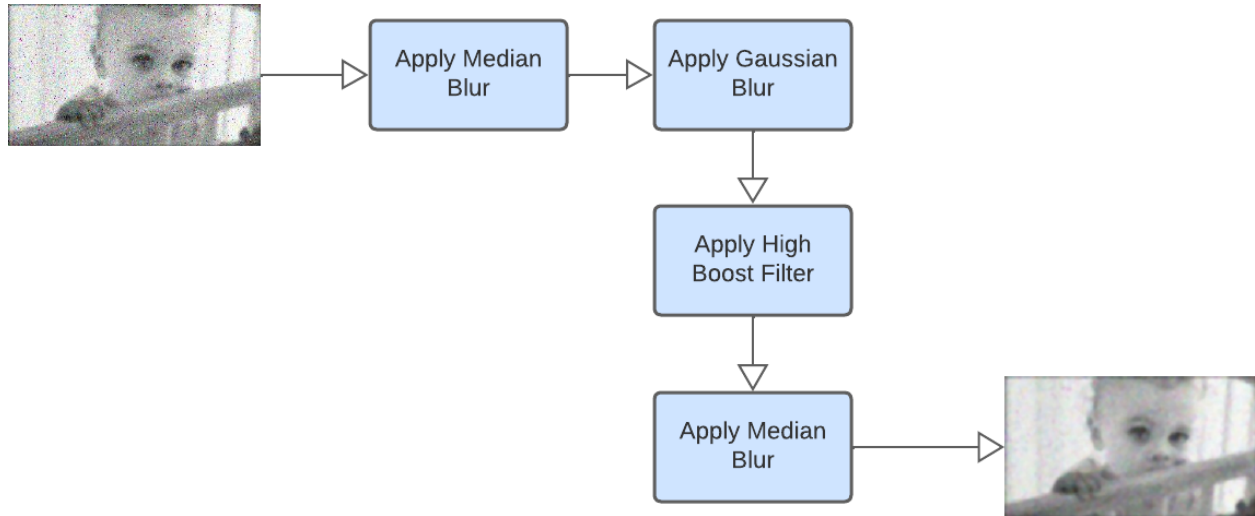
Figure 3: The baby

These three images contain salt and pepper noises and the task for this assignment is to restore the image as close as possible to the original. Salt noise is white noise that can be seen in the picture while pepper noise is black noise. In figure 1, it can be seen clearly that salt noise has an obvious presence compared to pepper noise because the image itself is a dark image, thus the pepper noise cannot be seen clearly. In figure 2, it can be seen that pepper noise or black noise has the most obvious presence compared to white noise because the image itself is a bright image which causes the salt noise cannot be seen clearly. In figure 3, both of the noises which are salt and pepper can be seen clearly because the image is in grayscale. The solution that we proposed needs to be able to restore these three images using the same 2 solutions, which means, the solution needs to be able to restore binary image, colored image and grayscale image in the same python file. The solution needs to be able to restore the image as close as possible to the original images using combinations of filters.

Proposed Solution 1

Process flow of proposed solution





Explanation

```
#high boost filter c = 9
kernel = np.array([[ -1, -1, -1], [-1, 9, -1], [-1, -1, -1]])

median = cv.medianBlur(img, 5) #using median blur filter
gaus = cv.GaussianBlur(median, (3,3), 1, 1, cv.BORDER_DEFAULT) #using Gaussian blur filter
img1 = cv.filter2D(gaus, -1, kernel) #using high boost filter
dst = cv.medianBlur(img1, 7) #using median blur filter
```

In solution 1, to restore the noise image to the original image we have apply 4 steps:

1. Median Blur

- `median = cv.medianBlur(img, 5)`
- Use to remove noise from the image

2. Gaussian Blur

- `gaus = cv.GaussianBlur(median, (3,3), 1, 1, cv.BORDER_DEFAULT)`
- Use to blur/smooth the image by reduce noise and details of the image

3. High Boost Filter

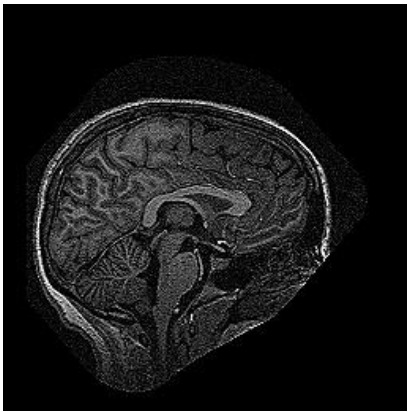
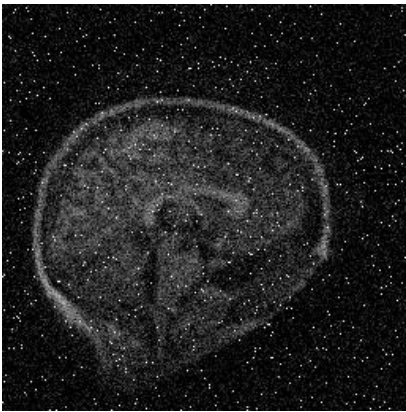
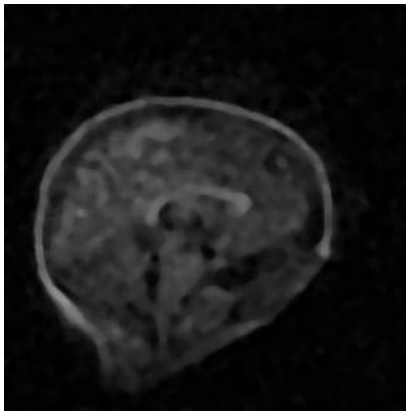
- `kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])`
- `img1 = cv.filter2D(gaus, -1, kernel)`







- Use to enhance image components, sharpen the edges.

4. Median Blur

- `dst = cv.medianBlur(img1, 7)`
- After we sharpen the image, we need to make it blur a bit to reduce noise that might appear after the sharpening process and there is some area to move out from the original pixel.

Results

Reference Image (Original Image)	Noised Image	Restored Image	Pixel Similarity Rate
			63.64%

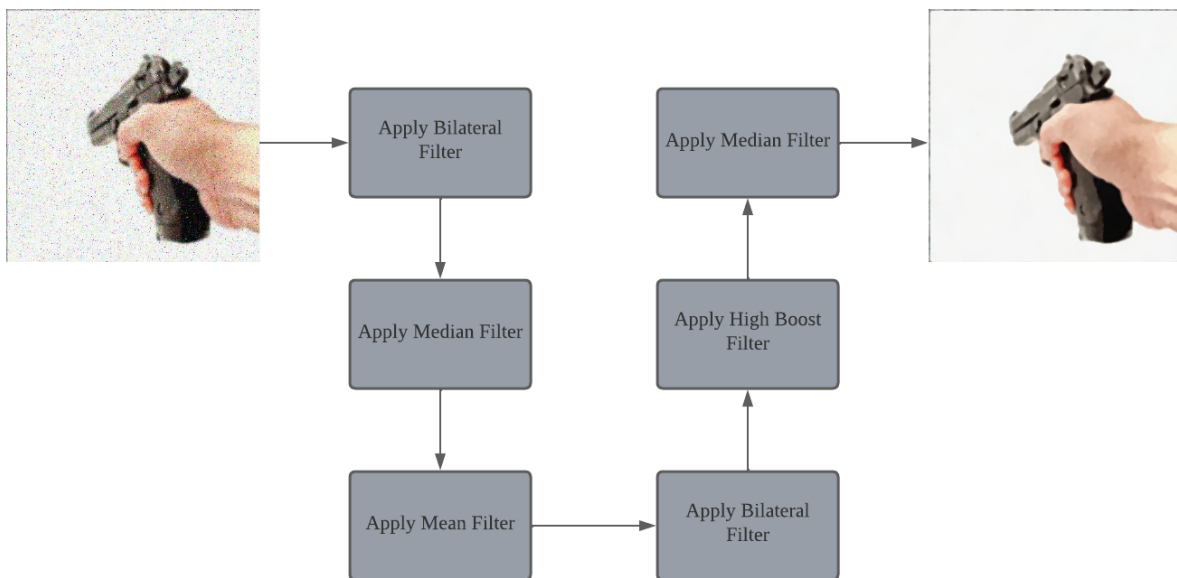
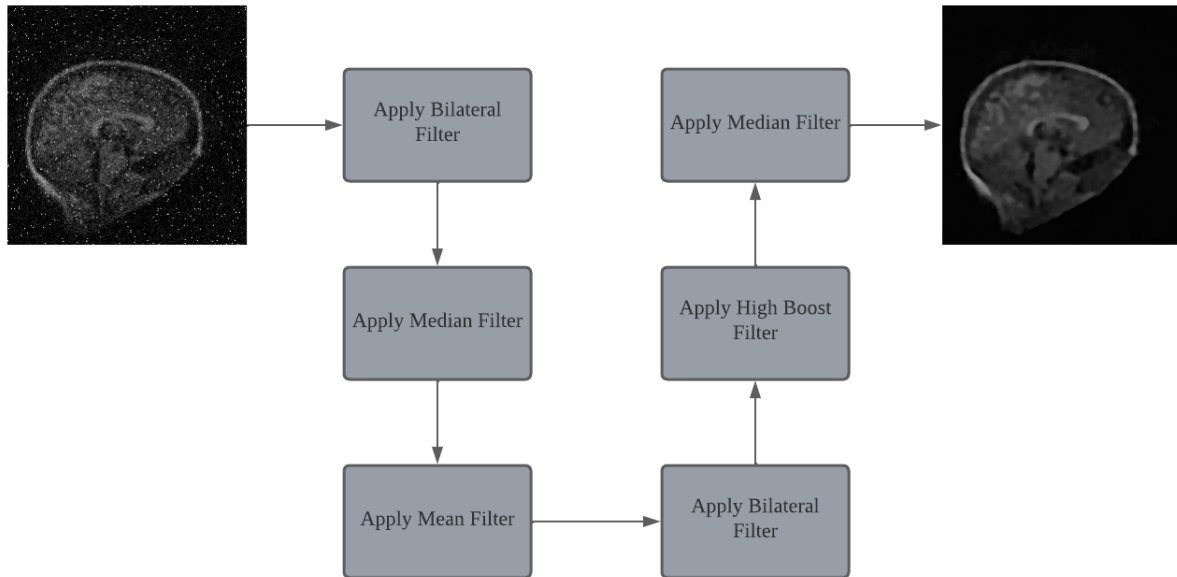
			<p>64.26%</p>
			<p>72.31%</p>

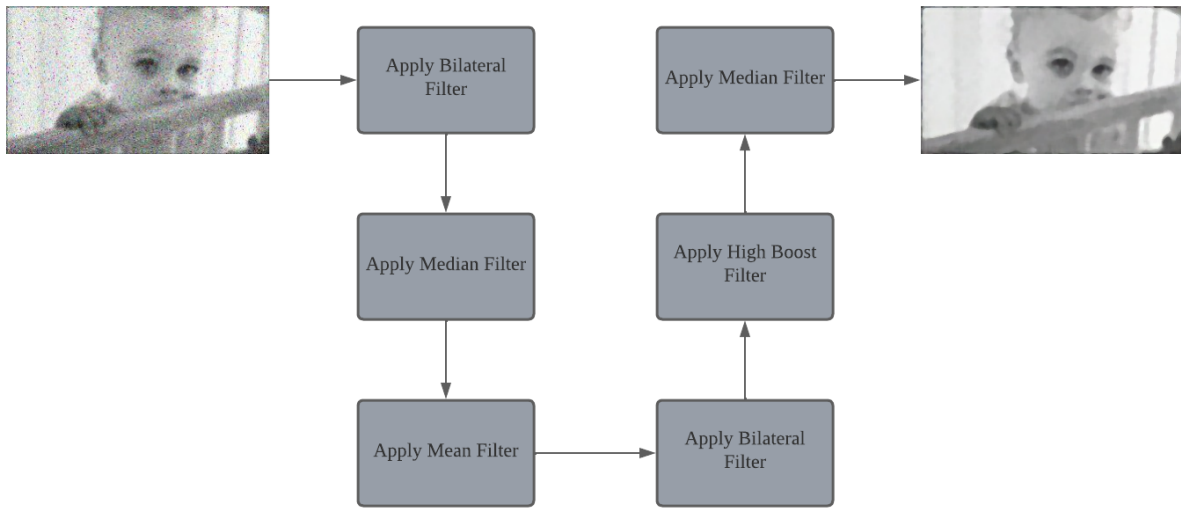
Analysis

By applying the same filters with the same steps for 3 sample images, it obviously shows that the third image (baby) obtains a higher similarity rate compared to the first (brain) and second (pistol) image. The concept of this solution that we use to remove noise is by blurring the image. Therefore, from that the noise and the details of the image is reduced. From the restored image 1, the output is quite blurry, we cannot see the blood vessels and details of the brain. However, there is no more noise. The similarity rate of the first image is 63.64%, the lowest one compared to another 2 images. It can be seen by our eyes that the quality of the image is quite low. Next, the second image obtains a 64.26% similarity rate. It actually is not much different compared with the first image. It is an only coloring sample image, but the solution is not too much give the impact to the output. After we applied with the filter in solution 1, noise was reduced. At the same time, the quality of the image also decreases. For the third image, it obtained 72.31% of similarity. Meaning that this solution is most suitable with the third image. Even though it has a high rate, the quality of the image is still low, since the output is not sharp compared to the original. Hence, by using the solution 1, it cannot restore the image totally as the original image. It just can reduce some noise.

Proposed Solution 2

Process flow of proposed solution





Explanation

```
kernel1 = np.ones((3,3),np.float32)/9 #mean filter
kernel2 = np.array([[ -1, -1, -1], [ -1,  9, -1], [ -1, -1, -1]])
           #high boost filter c = 9

img1 = cv.bilateralFilter(img, 100, 20, 120) #using bilateral filter
img2 = cv.medianBlur(img1, 3) #using median blur filter
img3 = cv.filter2D(img2, -1, kernel1) #using mean filter

img4 = cv.bilateralFilter(img3, 31, 20, 10) #using bilateral filter
img5 = cv.filter2D(img4, -1, kernel2) #using high boost filter
dst = cv.medianBlur(img5, 7) #using median blur filter
```

In proposed solution 2, there are four different filters used to remove the salt and pepper on the pictures as well as sharpening the pictures to increase the pixel similarity rate between the restored images and original images.

The following are the filters used in the proposed solution 2:

1. Bilateral Filter

cv.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[borderType]])

- **src**: Source of 8-bit or floating-point image that will be filtered.
- **d**: Diameter of each pixel neighborhood that will be used during the filtration. If the value is non-positive, the diameter will be computed from sigmaSpace.
- **sigmaColor**: Filter sigma in the color space. Large value of sigmaColor means farther colors within the pixel neighborhood will be mixed together, resulting in larger areas of semi-equal color.
- **sigmaSpace**: Filter sigma in the coordinate space. Large value of sigmaSpace means farther pixels will influence each other as long as their colors are close enough. If $d > 0$, it specifies the neighborhood size regardless of sigmaSpace. Otherwise, d is proportional to sigmaSpace.
- **dst**: Destination of image of same size and type as src
- **borderType**: Border mode used to extrapolate pixels outside of the image

In this function, `dst` and `borderType` are optional values which means it is not compulsory to be included. For sigma values, if the values are small which is less than 10, the effect is not very obvious, unlike if the values are large which is more than 150, the effect will be very strong and will make the image look more cartoonish because this filter preserve most edges and shadows of the images but most of the shadings part will be lost.

In proposed solution 2, this filter is used twice as follows

```
1. cv.bilateralFilter(img, 100, 20, 120)
2. cv.bilateralFilter(img3, 31, 20, 10)
```

The first filter is used to reduce the noise from the raw Noise image meanwhile the third filter is used to reduce the noise from the filtered image from previous operation. Both lines use different values of `d` and sigma values.

2. Median Filter

cv.medianBlur(src, ksize[, dst])

- **src**: Source of 8-bit or floating-point image that will be filtered.
- **ksize**: Kernel size of the filter and it must be odd number. (ksize x ksize)
- **dst**: Destination of image of same size and type as src

In this function, `dst` is an optional value. This function uses `ksize` provided to filter the image using a median filter. Median blur function is very useful in removing salt and pepper noise in an image.

In proposed solution 2, this filter is used twice as follows:

```
1. img2 = cv.medianBlur(img1, 3) #using median blur filter
2. dst = cv.medianBlur(img5, 7) #using median blur filter
```

The first median filter is used to reduce the noise from previous operation by blurring the image as well as the second median filter in which it is the final filter used for the image restoration. The first median filter uses 3x3 kernel size, meanwhile the second median filter uses 7x7 kernel size.

3. Mean Filter & High Boost Filter

cv.filter2D(src, depth, kernel[, dst[, anchor[, delta[, borderType]]])

- **src**: Source of 8-bit or floating-point image that will be filtered.
- **depth**: Desired depth of the destination image, usually `depth = -1` is used to make sure that the depth of the image is same as the source
- **kernel**: convolution kernel, written in a form of array
- **dst**: Destination of image of same size and type as `src`
- **anchor**: Anchor of the kernel which indicates the position of the filtered point within the kernel. The default value for anchor is `(-1, -1)` which means it is at the center.
- **delta**: Optional value added to the filtered pixels before storing them.
- **borderType**: Border mode used to extrapolate pixels outside of the image

In this function, `dst`, `anchor`, `delta` and `borderType` are optional values. This function can be used by different filters. The only difference that will make a significant effect on the image would be the kernel because the kernel will determine how the image will be manipulated.

In proposed solution 2, this function is used twice, first is for mean filter and the second is for high boost filter. The kernels for the filters are as follow:

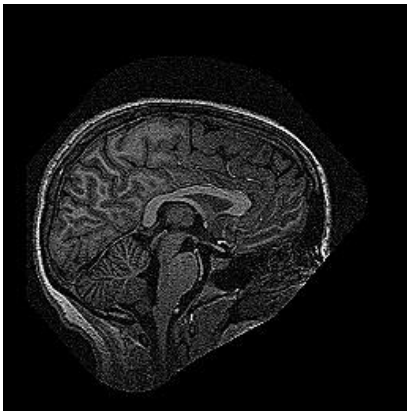
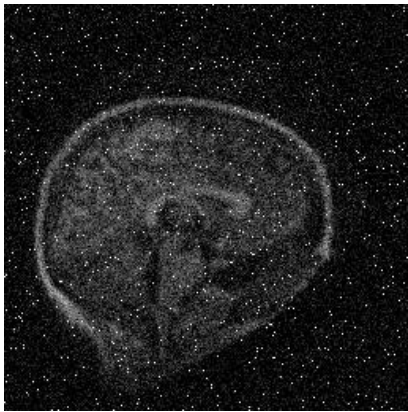
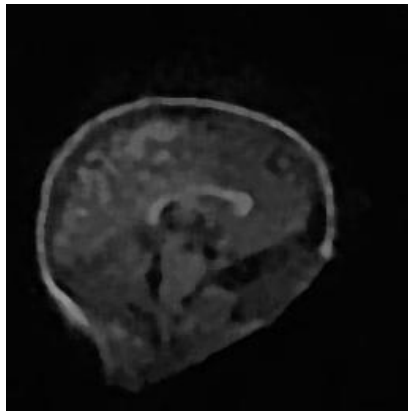
```
1. kernel1 = np.ones((3,3),np.float32)/9 #mean filter
2. kernel2 = np.array([[ -1,  -1,  -1], [ -1,  9,  -1], [ -1,  -1,  -1]]) #high
   boost filter c = 9
```







The function is used as follow:

```
1. img3 = cv.filter2D(img2, -1, kernel1) #using mean filter
2. img5 = cv.filter2D(img4, -1, kernel2) #using high boost filter
```

The first `filter2D` function uses a mean kernel to reduce the remaining noise in the image meanwhile the second `filter2D` function uses a high boost kernel to sharpen the image so the similarity can be closer to the original image.

Results

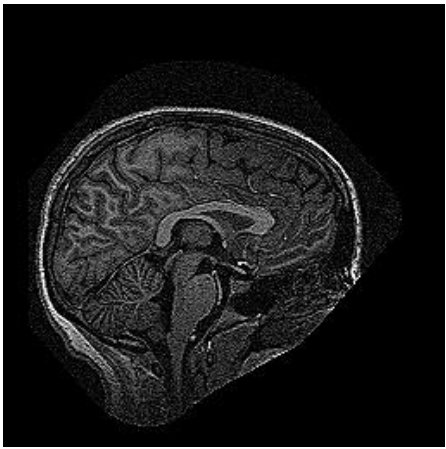
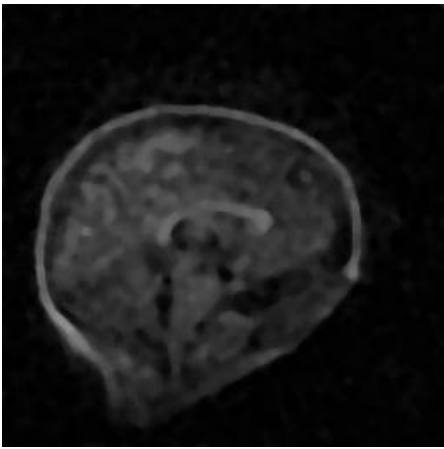
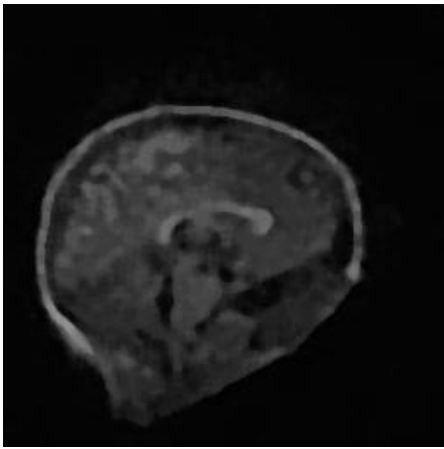
Reference Image (Original Image)	Noised Image	Restored Image	Pixel Similarity Rate
			65.56%







			<p>83.4%</p>
			<p>74.82%</p>

Analysis

Based on the results shown above, it can be seen that solution 2 can remove all salt and pepper noises from all three images. By using a few combinations of different filters, this solution resolves the image by removing the noises and increasing the quality of the image. With the pixel similarity rate of 65.56%, the first image which is the brain image has the lowest similarity rate compared to the two images using this solution. It can be seen that the noises are removed from the image, but the quality of the image itself cannot be restored successfully. The detail parts of the brain are still blurry, only the general parts of the brain can be seen clearly. The third image which is the baby image reached 74.82% of pixel similarity rate which is the second highest percentage of image restoration. Most parts of the third image can be seen without any noises and the image is sharpened which makes some features in the image can be restored. Out of all images, the second image has the highest pixel similarity rate with 83.4% of similarity with the original image. Most edges and shadows can be restored which makes this image have the highest similarity rate. Using solution 2, it can be seen that most noises can be removed. However, as for the quality of the image itself, solution 2 can only restore the quality partially.

Analysis of Proposed Solution

Reference Image	Solution 1 Output Image	Pixel Similarity Result		Solution 2 Output Image
		Solution 1	Solution 2	
		63.64%	65.56%	

		64.26%	83.4%	
		72.31%	74.82%	

From this table of comparison, it can be seen that solution 2 has a higher pixel similarity rate for all images compared to solution 1. For image 1 and 3, there is not much difference in pixel similarity rate, but for image 2, there is a huge difference between the results of solution 1 and solution 2 which is 19.14%. It can be seen that solution 2 has the most consistent image restoration in which at least one of the images has the highest similarity rate. This is caused by the combinations of different filters and kernel sizes which causes the image to be restored smoothly. None of the images in both solutions has a similarity rate lower than 50%, which means both solutions have successfully removed the noise. Both solutions need more operations to make sure the similarity rate can be increased to be closer to the original image. Since there is a huge difference between image 2 in solution 1 and 2, we can see that

image 2 in solution 1 has a little noise left while in solution 2 it has a much smoother effect. This is because the last filter used in solution 2 is the median blur filter which is used to smoothen an image. After the image is sharpened, the image gets smoothen so that the edges do not have much contrast which can cause the similarity rate to decrease. The same thing happens in image 3 in which we can see that image 3 in solution 2 has a smoother effect compared to image 3 in solution 1. From this comparison, we can see that to restore an image, different combinations of filters need to be used to make sure it can correct the pixel of the image so it can have a higher similarity rate with the original image.

Conclusion

To conclude, there are many ways and solutions to remove the noise and restore to the original image with the different similarity percentage. In solution 1, we apply median blur, gaussian blur, high boost filter. However, in solution 2 we apply bilateral filter, median filter, mean filter, and high boost filter. Some filters are used more than once in the same solution, to produce the higher similarity. Since we have 3 sample images, some images have high similarity when applying solution 1 but some have low similarity, same with the solution 2. Because the images have different gray scales and colors. To increase the similarity percentage, we need to combine many filters, rearrange, mix and match those into one program to ensure the filter applied is appropriate with the image needed. The number of filters cannot influence the similarity, we can use less or more filters but the main thing is to produce high similarity and completely restore the image as the original one.

Video presentation link: <https://youtu.be/PEyvyWOcHw>

References

- Bhatt, K. (2022, December 19). *How to sharpen an image in Python using OpenCV*. CodeSpeedy.
<https://www.codespeedy.com/how-to-sharpen-an-image-in-python-using-opencv/>
- Bilateral Filtering*. (n.d.).
https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html
- Edeza, T. (2021, December 25). *Introduction to Image Processing with Python — Image Filtering*. Medium.
<https://towardsdatascience.com/introduction-of-image-processing-with-python-image-filtering-193e9108ea1d>
- Flores, T. (2021, December 9). *Median Filtering with Python and OpenCV - Tony Flores*. Medium.
<https://medium.com/@florestony5454/median-filtering-with-python-and-opencv-2bce390be0d1>
- GeeksforGeeks. (2022, December 8). *Image Sharpening Using Laplacian Filter and High Boost Filtering in MATLAB*.
<https://www.geeksforgeeks.org/image-sharpening-using-laplacian-filter-and-high-boost-filtering-in-matlab/>
- GeeksforGeeks. (2023, January 4). *Python OpenCV | cv2.imwrite() method*.
<https://www.geeksforgeeks.org/python-opencv-cv2-imwrite-method/>
- Maharaj, S. (2021, August 26). *Sharpening An Image using OpenCV Library in Python*. Analytics Vidhya.

<https://www.analyticsvidhya.com/blog/2021/08/sharpening-an-image-using-opencv-library-in-python/>

Manansala, J. (2021, December 28). *Image Processing with Python: Image Effects using Convolutional Filters and Kernels* | by Jephraim Manansala | *The Startup*. Medium.
<https://medium.com/swlh/image-processing-with-python-convolutional-filters-and-kernels-b9884d91a8fd>

OpenCV: Image Filtering. (n.d.).
https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html

Spatial Filters - Median Filter. (n.d.). <https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm>