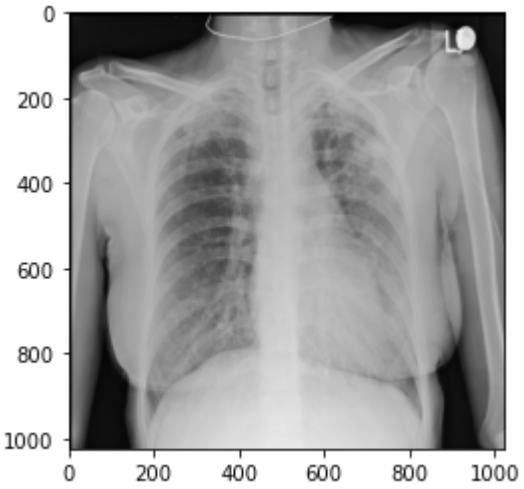
Note:
I have renamed the two images as xray_one and xray_two and will be using the same terminology in the report for convenience.

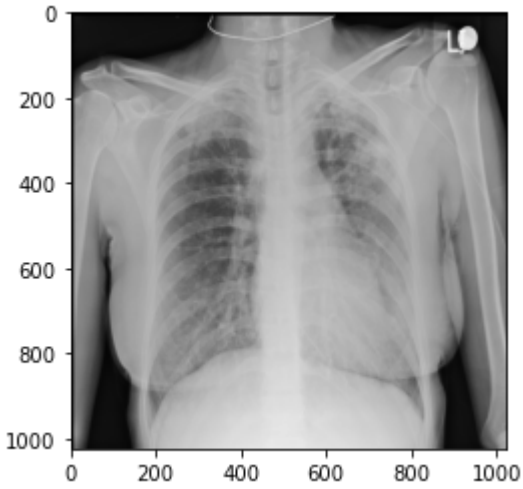
Q1.

Please note that matplotlib uses the upper left corner as the origin.

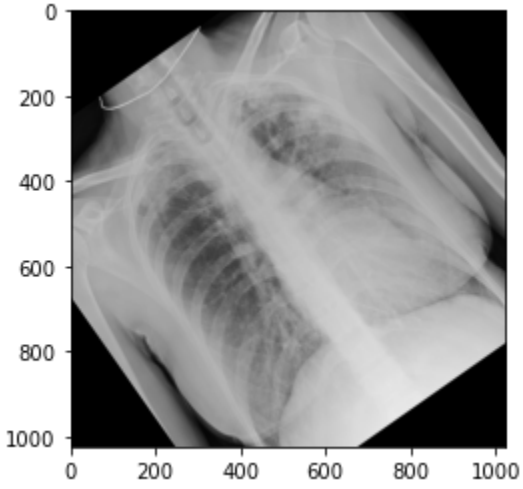
Transformations performed on Xray_one:



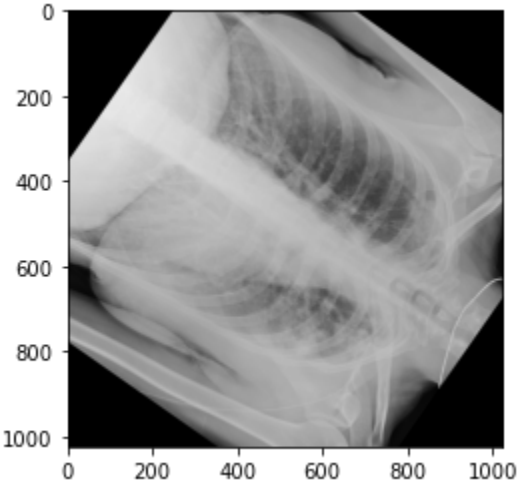
Xray_one original Image



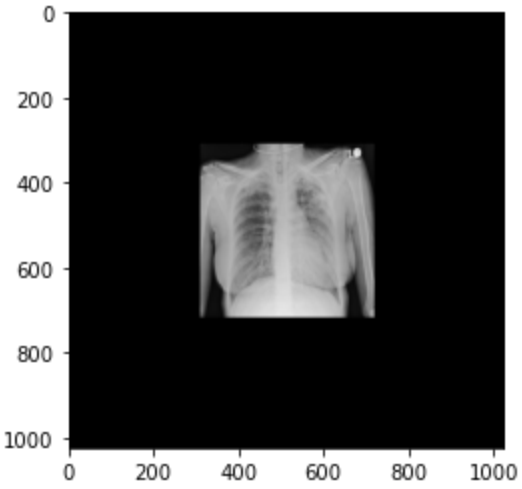
Xray_one translated Image



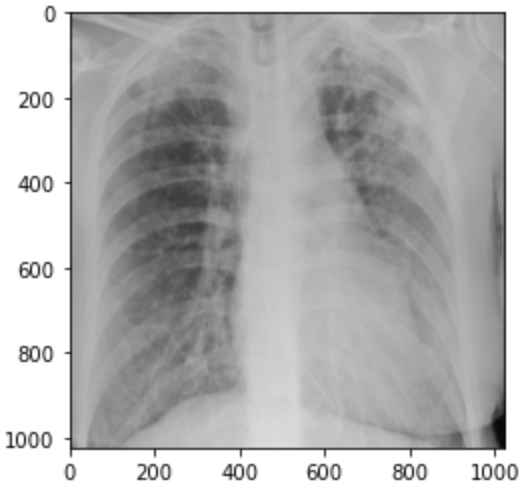
Xray_one Rotated Image (35°)



Xray_one Rotated Image (-125°)

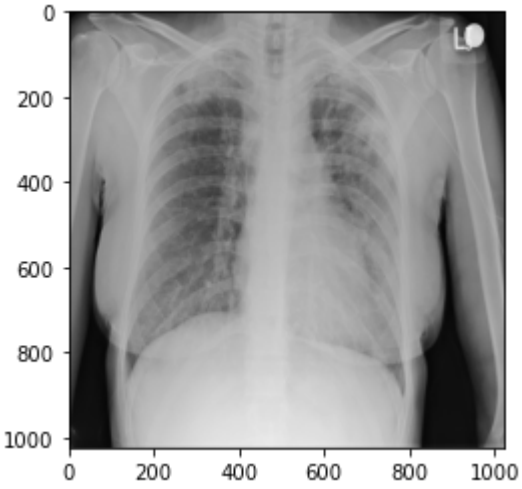


Xray_one Scaled (0.4)

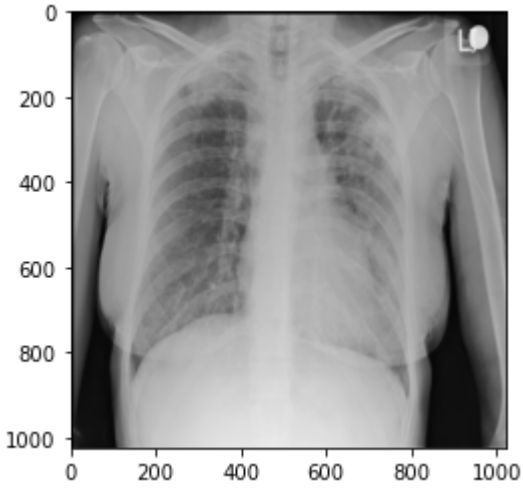


Xray_one Scaled (1.4)

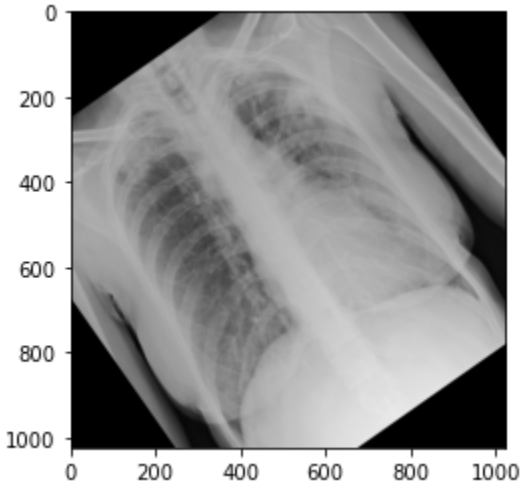
Transformations performed on Xray_two:



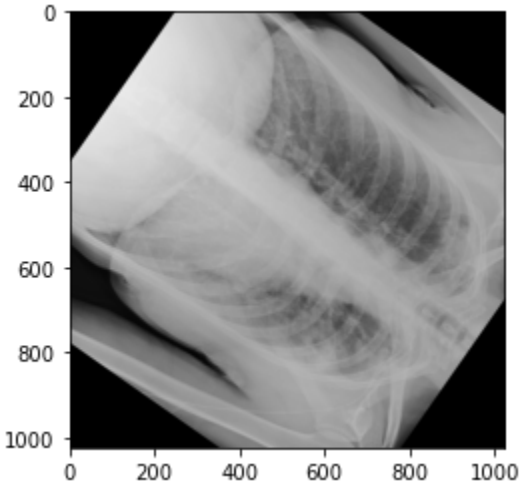
Xray_two original Image



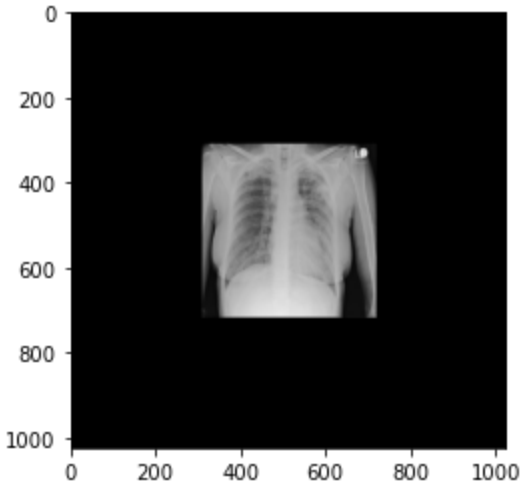
Xray_two translated Image



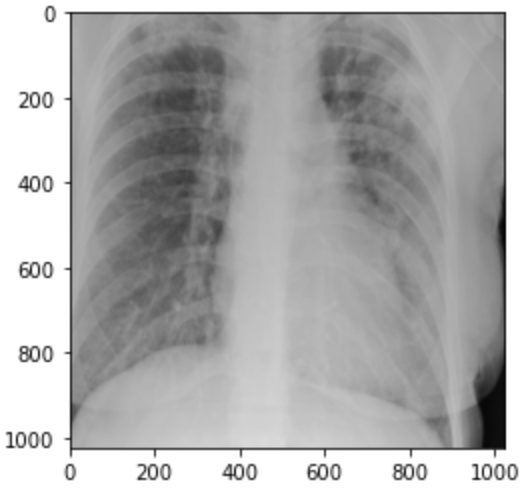
Xray_two Rotated Image (35°)



Xray_two Rotated Image (-125°)



Xray_two Scaled (0.4)

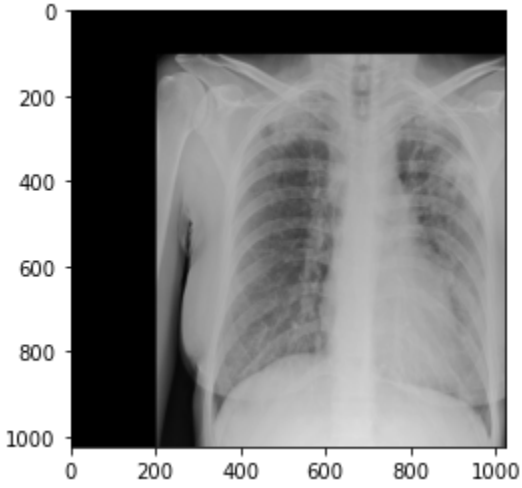


Xray_two Scaled (1.4)

Transformation matrices used to perform mentioned transformations -

Translation:

Since the tx and ty are small compared to the dimensions of the image, here is how it looks when translated by (200,100) pixels. When the transformed coordinates are non-integers, I use the bilinear interpolation method to produce the new image.



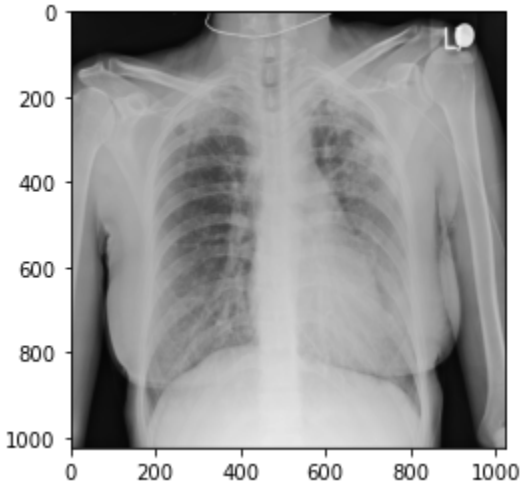
Rotation:

To get the rotation around the center of the image, I first translated the image coordinates by half the width and half the height to match the center of the image with

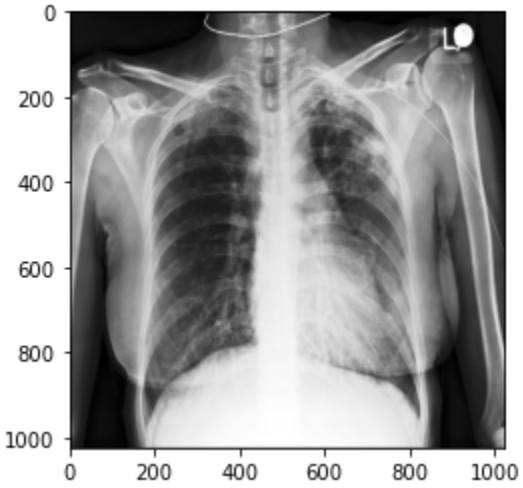
the origin, then performed rotation and then again translation to undo the first translation.

Scaling:
Even for scaling I followed the same procedure as the rotation one to scale w.r.t the center of the image

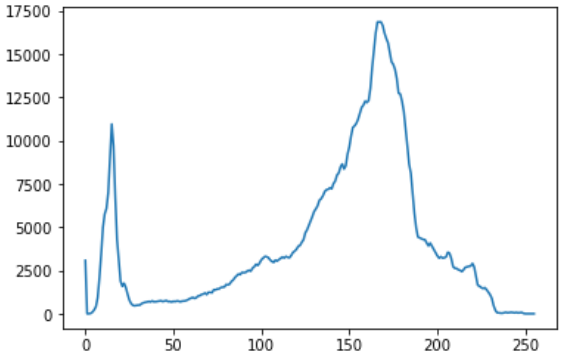
Q2



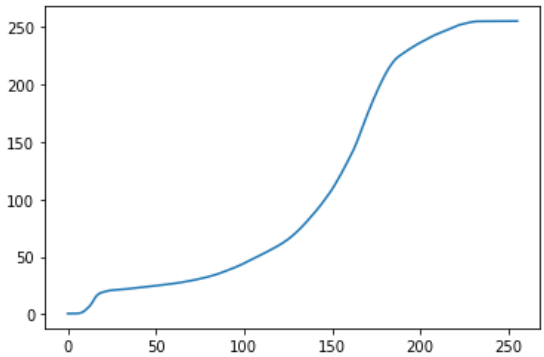
Xray_one Original Image



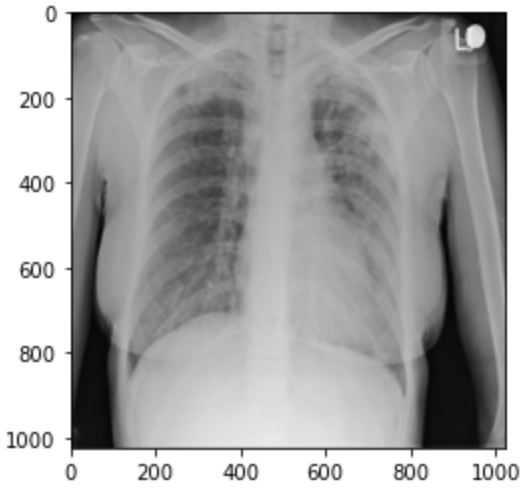
Xray_one after Histogram Equalization



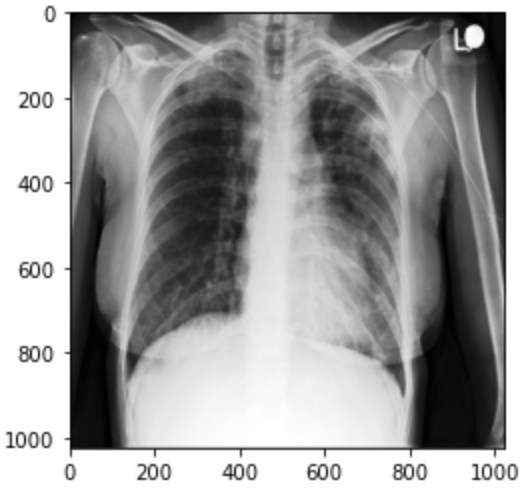
Original Histogram



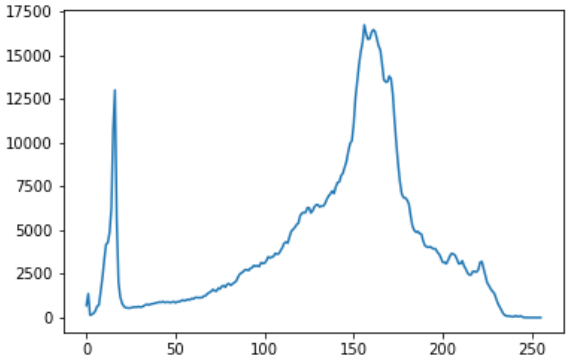
Equalized Histogram



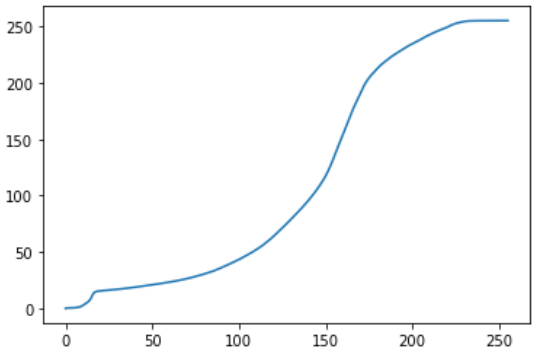
Xray_two Original Image



Xray_two after Histogram Equalization

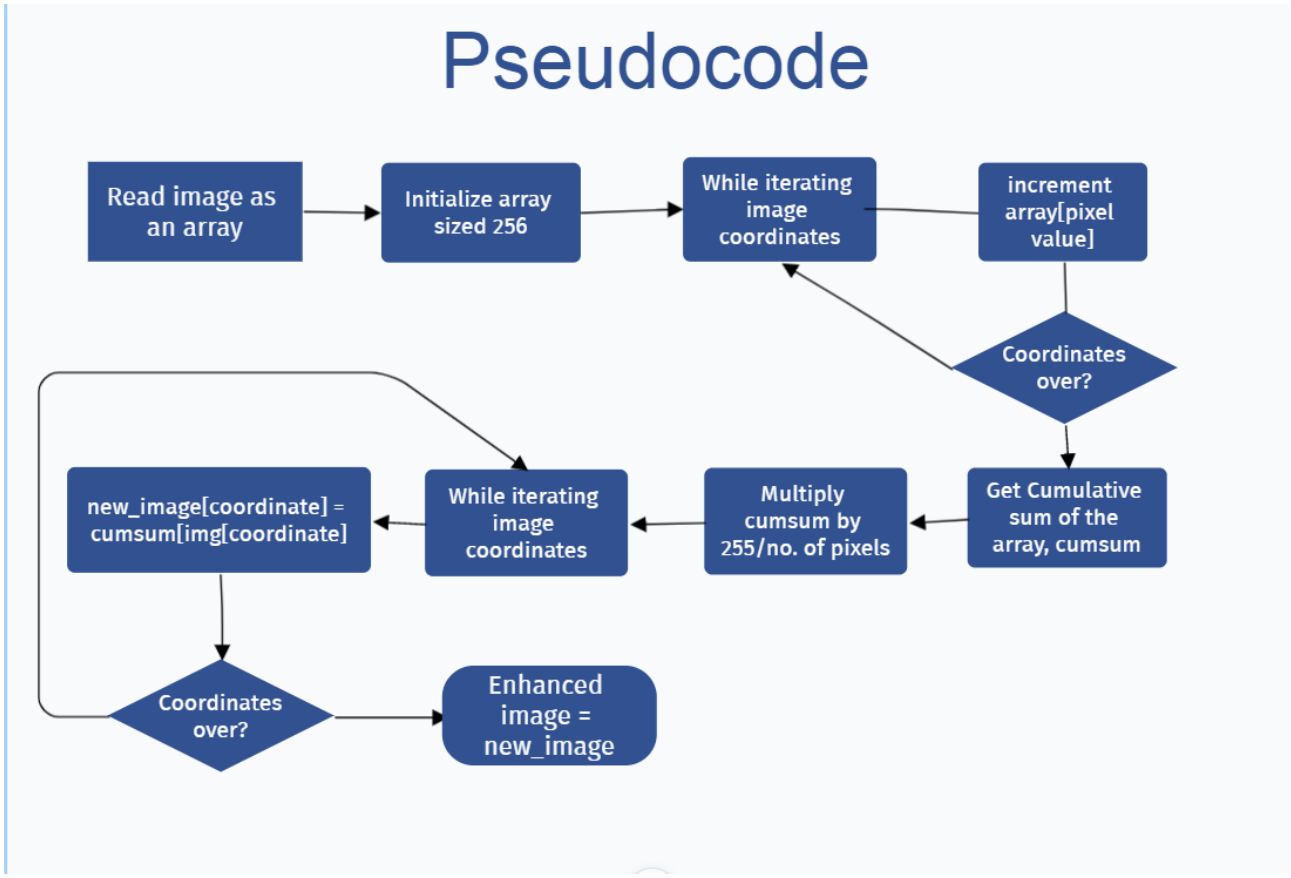


Original Histogram



Equalized Histogram

Pseudocode


Read image as an array
Initialize array sized 256
While iterating image coordinates
increment array[pixel value]
Coordinates over?
Get Cumulative sum of the array, cumsum
Multiply cumsum by 255/no. of pixels
While iterating image coordinates
new_image[coordinate] = cumsum[img[coordinate]
Coordinates over?
Enhanced image = new_image

Q3

PSNR values for different filters after adding different noises -

		Mean Filter	Median Filter	Gaussian Filter
Salt & Pepper Noise	Random Seed 1	10.29	17.32	10.61
	Random Seed 2	12.53	17.97	12.50
Gaussian Noise	Sigma = 1	17.76	17.89	13.59
	Sigma = 0.5	17.80	17.95	13.58

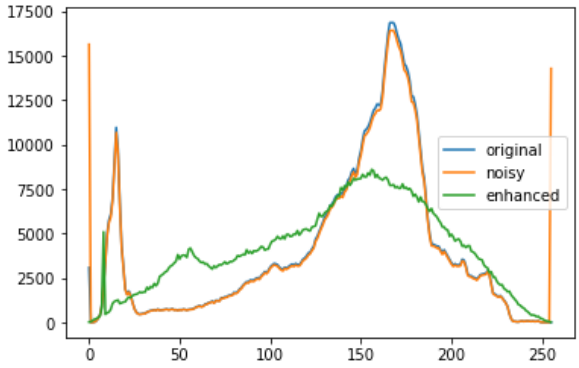
PSNR Values for different filters on Xray_one

		Mean Filter	Median Filter	Gaussian Filter
Salt & Pepper Noise	Random Seed 1	13.87	18.21	13.52
	Random Seed 2	12.53	17.97	12.50
Gaussian Noise	Sigma = 1	13.87	18.17	11.84
	Sigma = 0.5	13.87	18.18	13.67

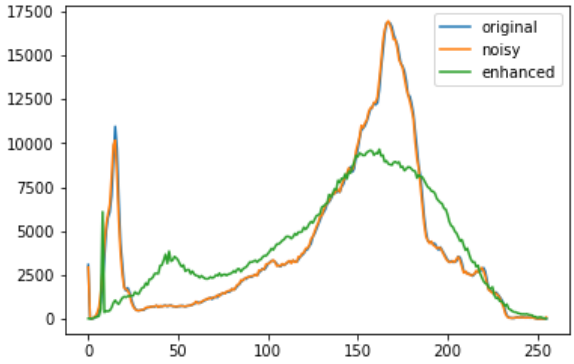
PSNR Values for different filters on Xray_two

Comparison of histograms of the original image, noisy image and enhanced image for various filters applied on Xray_one

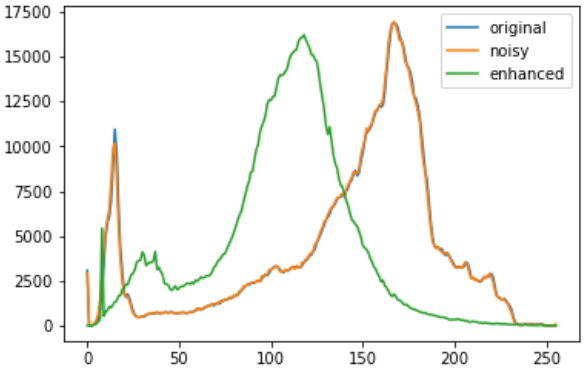
Note - Histogram plots for Xray_two could be found in the notebook



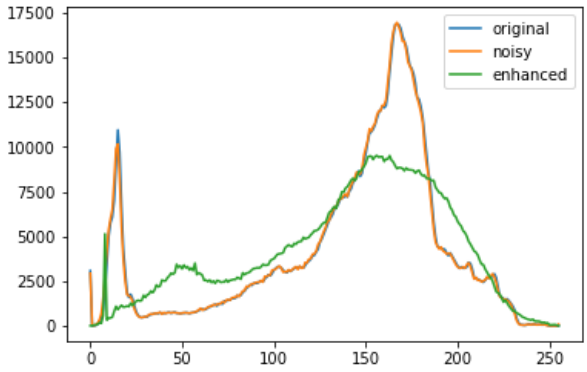
Mean filter, Salt & Pepper Noise



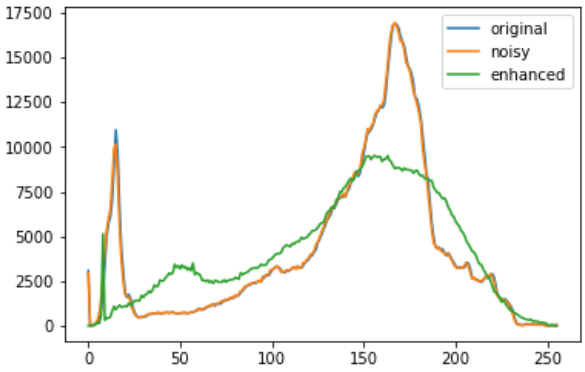
Median filter, Salt & Pepper Noise



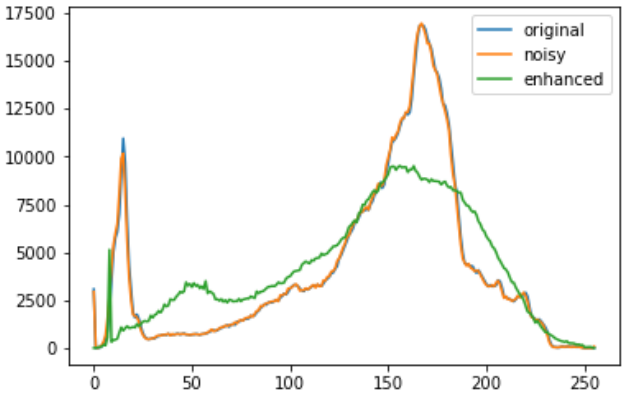
Gaussian filter, Salt & Pepper Noise



Mean filter, Gaussian Noise



Median filter, Gaussian Noise



Gaussian filter, Gaussian Noise

Conclusion:

As observed in the PSMR values table for different filters, the Median filter works the best for image enhancement. In mean and gaussian filters, every pixel in the eight neighbourhood window influences the value of the middle pixel. In salt and pepper filters since the noisy pixels have extreme values (either 0 or 255), mean and gaussian filters fail to denoise the image. On the other hand, the median filter chooses only one pixel from all the pixels in the window (pixel at the middle position when sorted), it does a good job denoising the image.

The gaussian filter does a better job than the mean filter in the salt and pepper case, because it weighs the middle pixel more than those in the eight neighbourhood window . And since the noise is random, this weighted sum performs better than the mean filter which assigns the same importance to every pixel in the window.

On the other hand, the mean filter does a better job than the gaussian filter when dealing with gaussian noise. That’s because the gaussian noise added has mean zero. Therefore when we average the pixels in the window the mean noise is closer to zero.

One more interesting observation is how the Gaussian filter shifts the histogram to the left for the image with the salt and pepper noise. It’s because the kernel values in the Gaussian filter are less than one, which does not affect black pixels since their value is 0 anyway but does lower the intensity of bright pixels.