Unsharp Masking

Unsharp masking is a linear filter that is capable of amplifying high-frequencies of an image. We are trying to use this technique to amplify the edges in our images.

Methodology:

1. The first step of the algorithm is to copy the original image and apply a gaussian blur into it (Blur intensity is defined by a setting appropriate **Radius**).

2. If we subtract the blurred image from the original image, we will obtain only the edges created by the blur. This is what is called **Unsharped mask**.

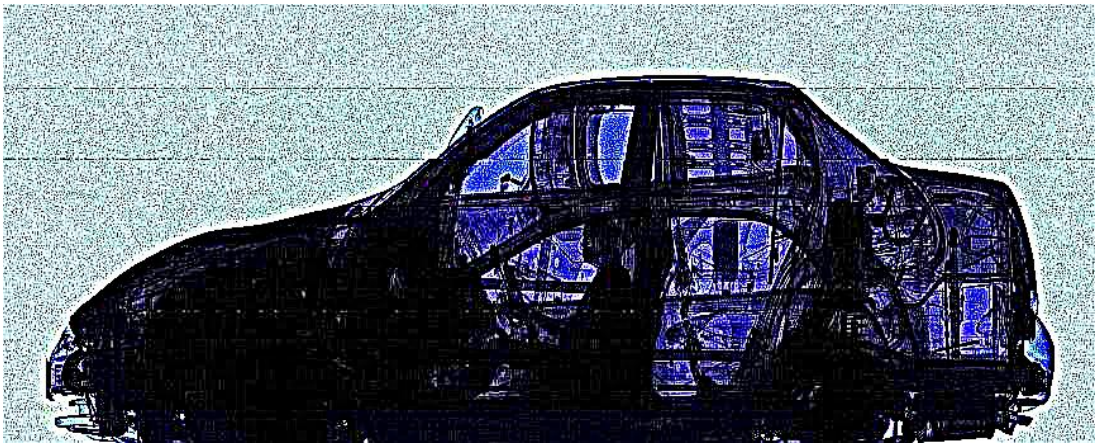
3. Finally, the enhanced image is collected after applying the following formula:

$$sharpened\ image\ =\ original\ image\ +\ amount\ *\ (unsharped\ mask)$$

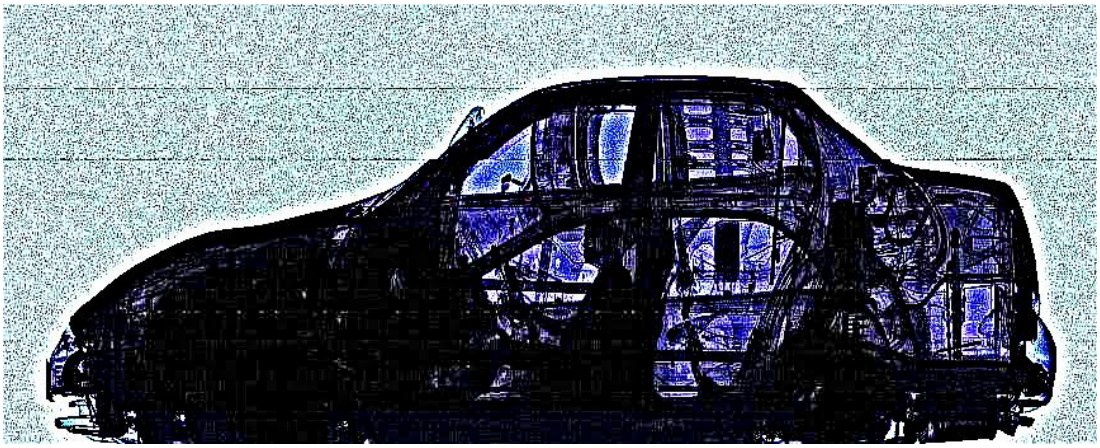
   The **radius** setting is related to the blur intensity (as explained before) because it defines the size of the edges. The **amount** setting, on the other hand, controls the intensity of the edges (how dark or light it will be).

Results:

Original Image

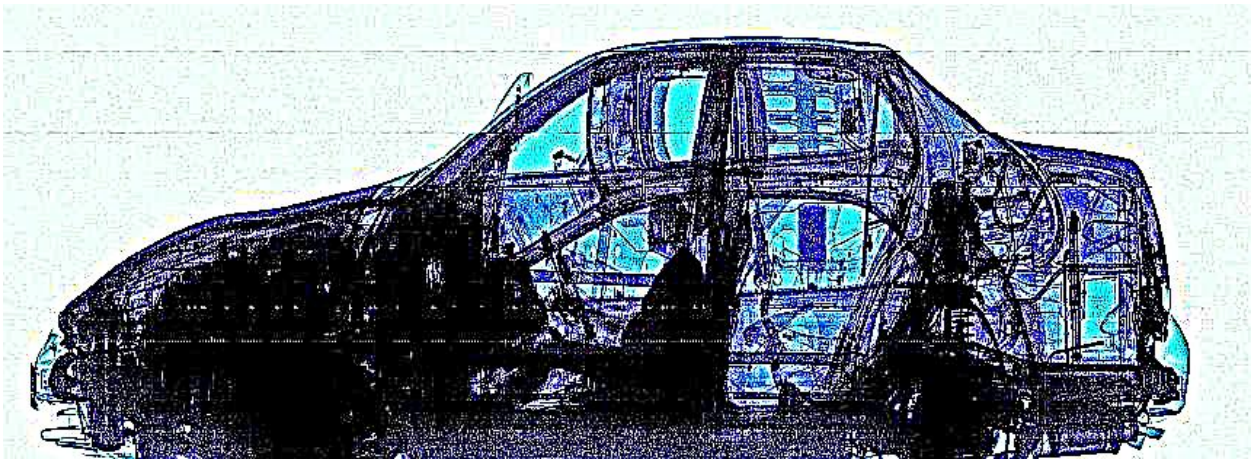


Enhanced Image (Radius = 3)

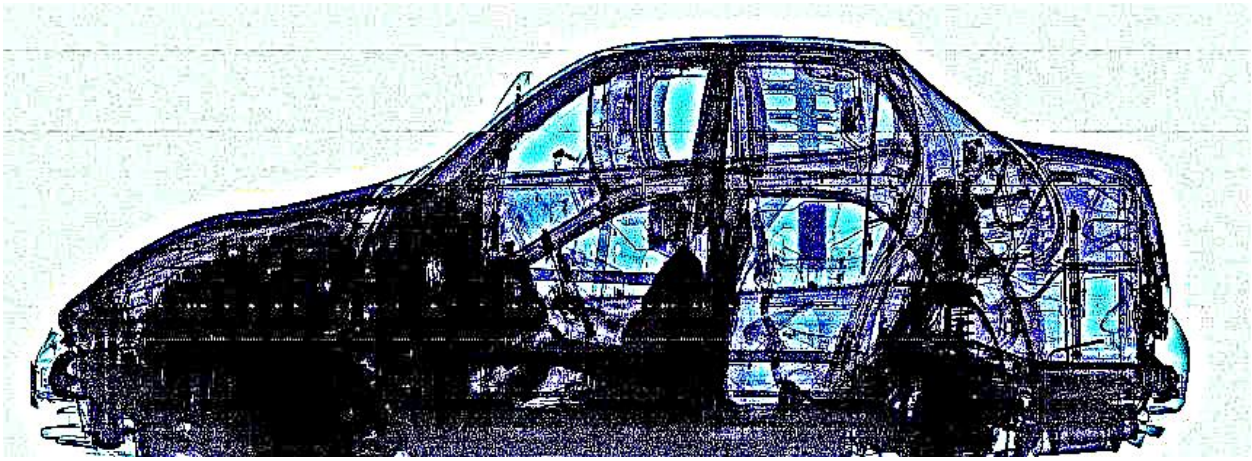


Enhanced Image (Radius = 5)

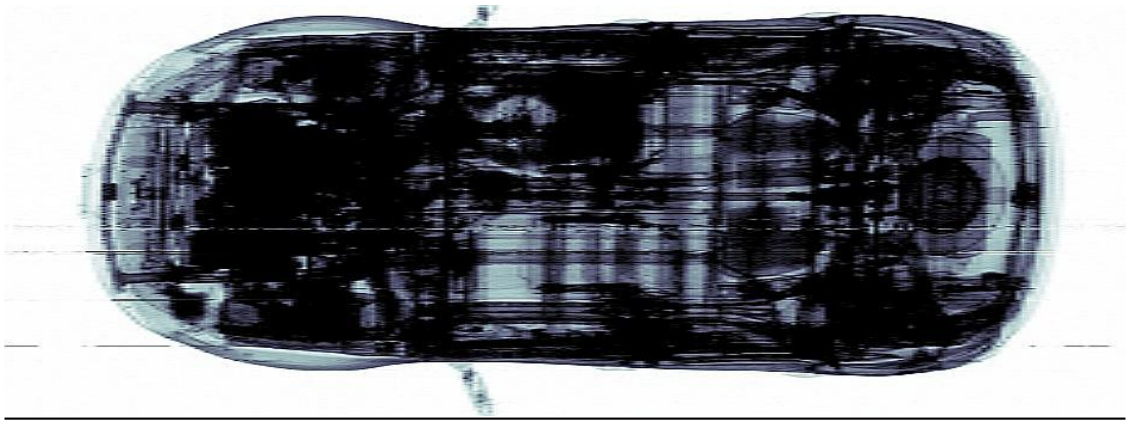
Original Image



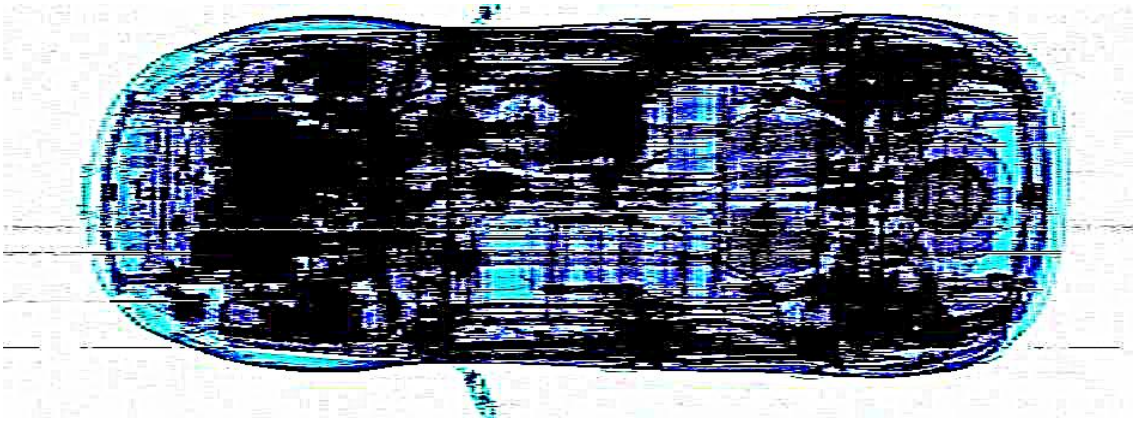
Enhanced Image (Radius = 3)



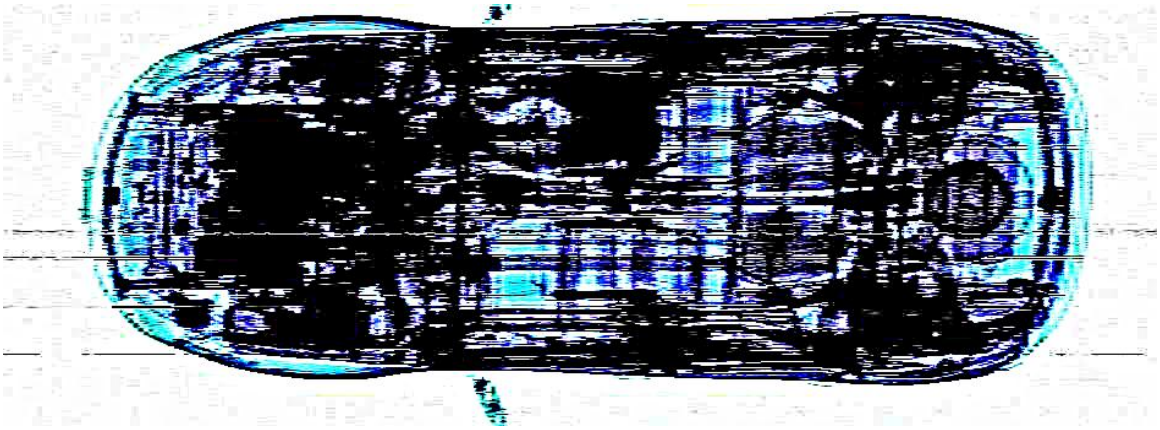
Enhanced Image (Radius = 5)



Original Image



Enhanced Image (Radius = 3)



Enhanced Image (Radius = 5)

CLAHE

The Contrast Limited Adaptive Histogram Equalization method is a histogram-based method used to improve contrast in images.

Methodology:

1. Convert all images to grayscale and normalize pixel values b/w 0 to 255. (in our case pixel value are within 255 but the images aren't grayscale)

2. Based on window size, we pad the image by pixels similar to those on the borders
3. Then, to each pixel in the image, calculate the clipped histogram for the region around it. If the occurrence of a pixel value (from the histogram) is greater than the clip limit, we clip at the limit and redistribute the exceeding to all pixels.
4. With the clipped histogram, we calculate the probability of each pixel in it and compute the cumulative distribution function, using the cumulative sum of the ordered pixels, and multiply each value of the function by 255, to limit the image's values to [0, 255].
5. After calculating the CDF, all pixels will have a transformation value. We now apply this transformation to the pixel in the center of the region.

Experiments:

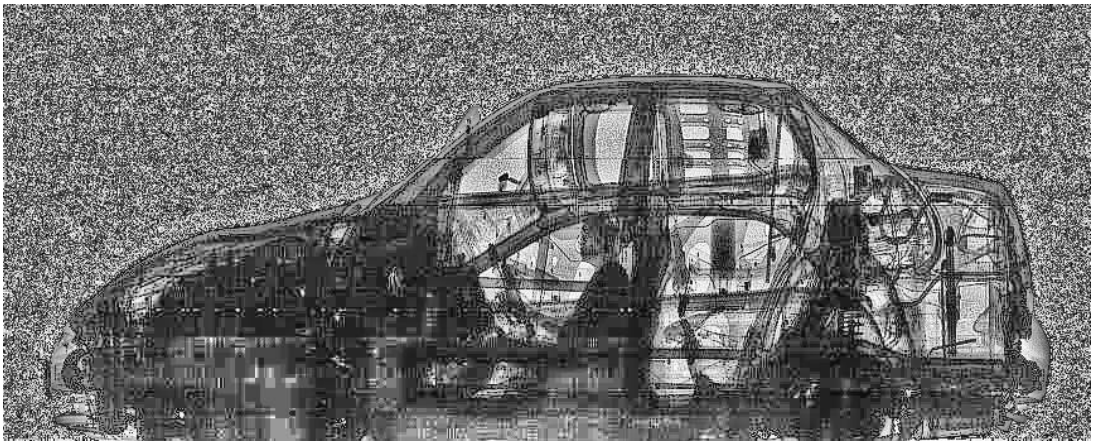
We ran the CLAHE algorithm on all the 8 images for 2 configurations

a. Window Size = 40, Clip Size = 100
b. Window Size = 100, Clip Size = 150

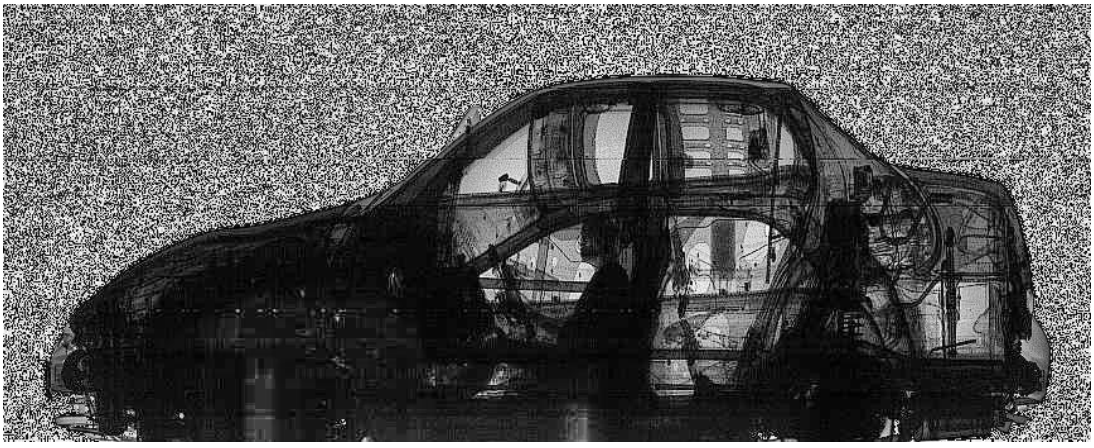
The results for all the 8 images for the 2 configurations can be found here:
https://drive.google.com/drive/folders/1b8pmgZb3Dsm1JRA-NBI0Dl49yFCBQNGa?usp=sharing

Some examples of results:

Original Image

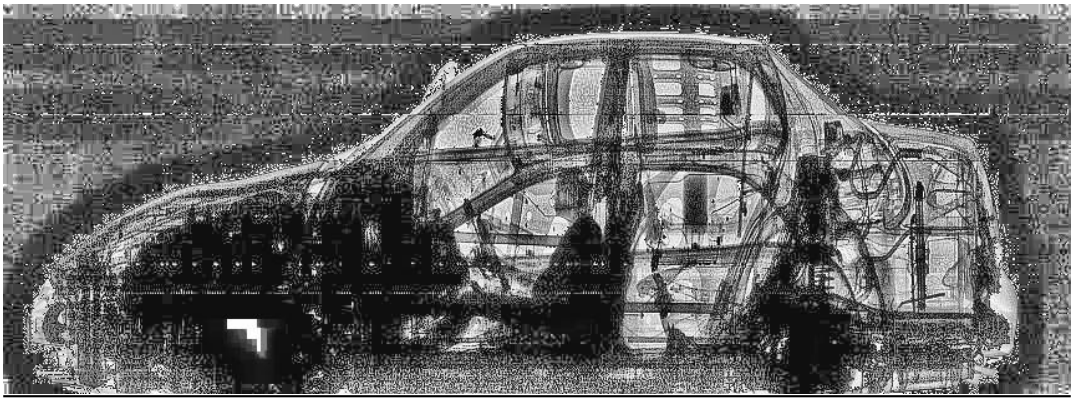


Processed Image (Window Size = 40, Clip Size = 100)



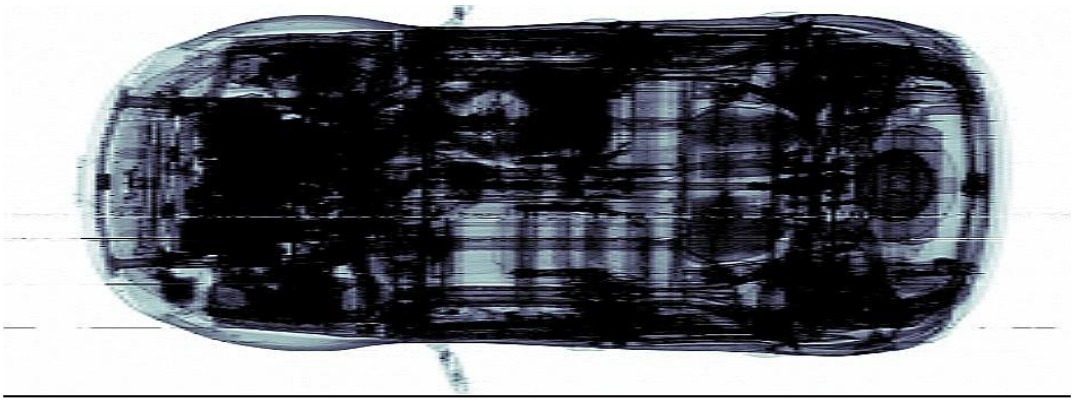
Processed Image (Window Size = 100, Clip Size = 150)

Original Image

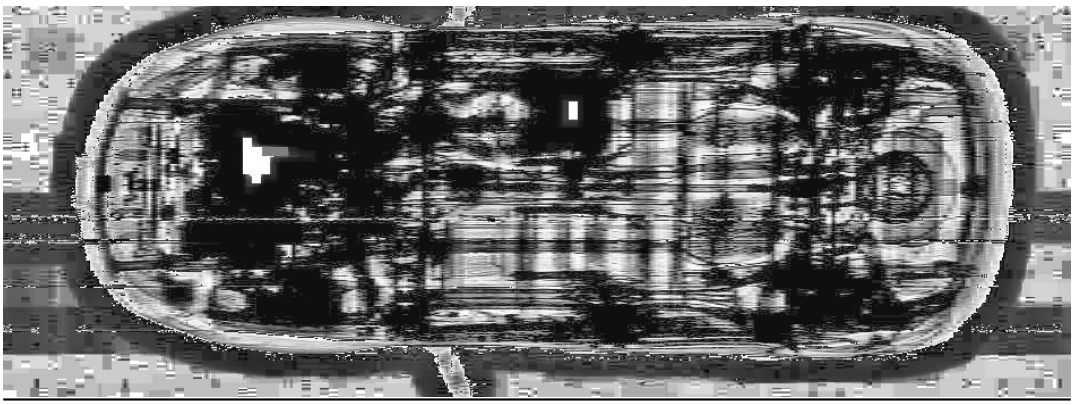


Processed Image (Window Size = 40, Clip Size = 100)

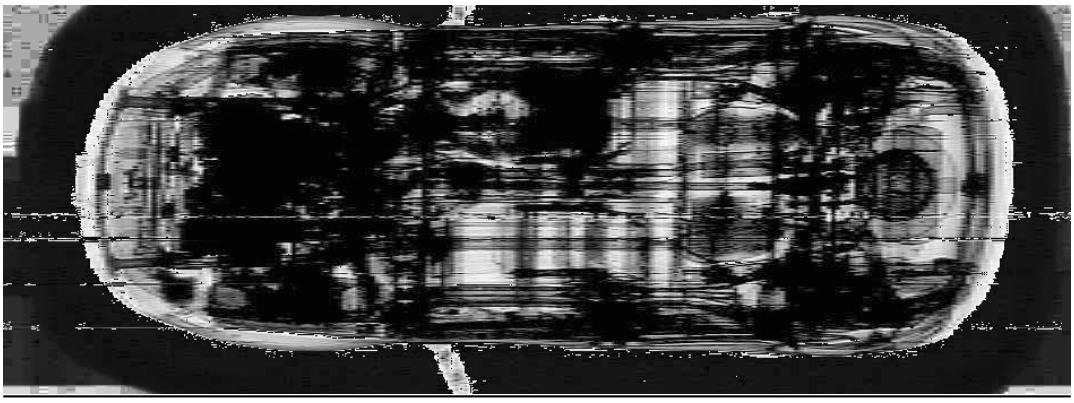
Processed Image (Window Size = 100, Clip Size = 150)



Original Image



Processed Image (Window Size = 40, Clip Size = 100)



Processed Image (Window Size = 100, Clip Size = 150)

Note:
The CLAHE algorithm takes a lot of time (~ 60 seconds to run on Google Colab CPU systems). As we have been informed, this is too long of a time for our purpose. However, if we are able to get satisfactory results with smaller window size and clip sizes, then the time for execution will reduce significantly. Also, the time measured is on Python and a language like C or C++ may run quite faster.