Utrecht University

University College Roosevelt

# ENGDATA201 – Image Processing and Computer Vision

---

## Project 1 - Digital Images, Histograms, Point Operations, and Filters


Author:

Joanikij Chulev

Professor:

Dr. Ir. Frank van der Stappen


February, 2024

# *Abstract*

*This assignment explores various image processing tasks using OpenCV, a powerful library for computer vision and image processing in Python. We address tasks such as applying different smoothing filters to grayscale images, removing noise using median filters, and performing sharpening using unsharp masking. Through practical implementation, we demonstrate OpenCV's versatility and effectiveness in handling common image processing challenges. The assignment also includes techniques for adding noise to images and using median filters for noise reduction.*

# Contents

**1 Introduction** ..... 4

**2 Section A -Digital Images** ..... 4

***Task 1*** ..... 4

***Task 2*** ..... 5

***Task 3*** ..... 6

**3 Section B - Histograms and Point Operators** ..... 8

***Task 1*** ..... 8

***Task 2*** ..... 9

***Task 3*** ..... 10

***Task 4*** ..... 11

**4 Section C - Filters** ..... 12

***Task 1*** ..... 12

***Task 2*** ..... 13

***Task 3*** ..... 15

## Experimental platform

***Hardware technology platform:***

OS Name: Microsoft Windows 11 Pro - Windows version: 22H2

System Type: x64-based PC

Processor: Intel(R) Core (TM) i7-1065G7 CPU @ 1.30GHz, 1498 Mhz, 4 Core(s), 8 Logical Proc.

Installed Physical Memory (RAM): 4.00 GB

GPU: Integrated Intel GPU (Intel Graphics)

***Software technology platform:***

Spyder version: 5.4.3 (conda)

Python version: 3.10.17 64-bit

Qt version: 5.15.2

PyQt5 version: 5.15.7

Operating System: Windows 11

Library - OpenCV: cv2.

# 1 Introduction

An essential component of computer vision is image processing, which finds use in everything from driverless cars to medical imaging. In this assignment, we use OpenCV, a potent Python package for computer vision and image processing, to investigate a variety of image processing problems. OpenCV is an invaluable tool for novices and specialists in the field of computer vision, offering a vast array of functions and algorithms for tasks including image editing, filtering, edge detection, and object recognition. In this assignment, we handle several image processing tasks using OpenCV. These tasks involve sharpening grayscale photos with unsharp masking, eliminating noise with median filters, and applying various smoothing filters. We also look at methods for dealing with noisy photos, such introducing noise and then reducing it using median filters.
OpenCV is a very useful tool for image processing jobs because of its flexibility and simplicity of usage. It is accessible to users of all skill levels thanks to its extensive documentation and vibrant community. Gaining proficiency with OpenCV enables one to carry out a variety of image processing tasks quickly and effectively.

# 2 Section A -Digital Images

## *Task 1*

Using numpy array indexing, we alter the R, G, and B values of a color picture in this exercise. Firstly, we divide the R values in half, or by half. By explicitly assigning 0 to the channels, we set both the R and G values to zero. This enables us to see how these adjustments affect the image's look in terms of color. We can see the images look more blueish in general. Consult the images below.

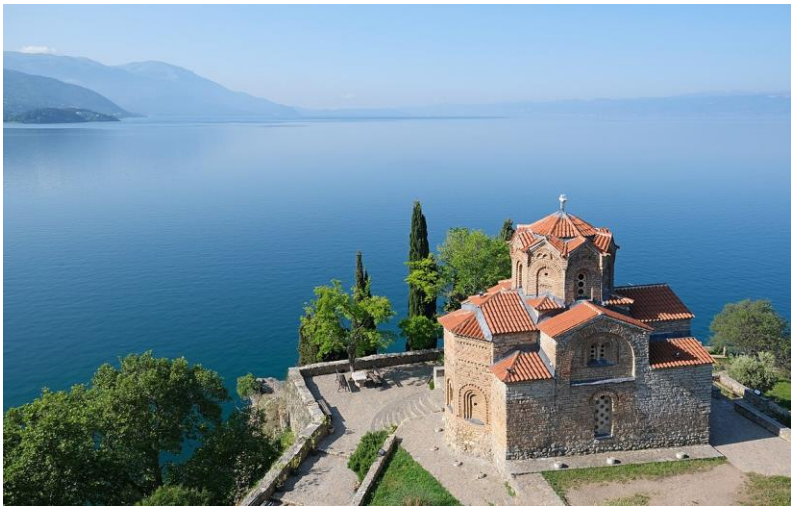

Figure 1: Original Image – Ohrid.

Figure 2: Image with halved red pixel values.

Figure 3: Image with red and green pixel values set to 0.

## *Task 2*

Using various weighted sums of the R, G, and B values, we convert a color picture to grayscale in this exercise. The R, G, and B means for every pixel are averaged using the average technique. Before adding up all of the channels, the weighted technique gives each one a unique weight. Because it accounts for how people see color, the weighted technique typically yields superior results. Consult the images below. We still use the same original image - Ohrid.

Figure 4: Grayscale image using the average method.

Figure 5: Grayscale image using the weighted method.

We can see really no difference in the results we got. It is worth mentioning the weights used. These weights are: 0.2989, 0.5870, 0.1140, for RGB, respectively.

## *Task 3*

In this task, we use OpenCV's resize function to shrink a grayscale image to half of its original size, hence reducing its resolution. We employ the INTER_AREA interpolation technique, which works well for reducing the size of photos. This method which is built-in in OpenCV

replaces/interpolates each 2-by-2 cluster of pixels in the original image by a single pixel (this value can be adjusted based on the method and how you define it).

By using this technique, a single pixel from the reduced-resolution picture is substituted for each 2x2 cluster of pixels from the original image. Using this procedure again will result in an even greater resolution decrease. Consult the images below.

It is feasible to reduce the resolution of color photos, but the process is more involved than it is for grayscale images. To maintain picture quality, strategies might include down sampling, averaging color values in the clustered regions, or using complex interpolation algorithms.

Note: Zoom on the images below to see obvious resolution differences.

Figure 6: Reduced Resolution Image – Greyscale.

Figure 7: Full Resolution Image – Greyscale.

Let us now try the same technique with the colored image. See below.

Figure 8: Reduced Resolution Image – Color.

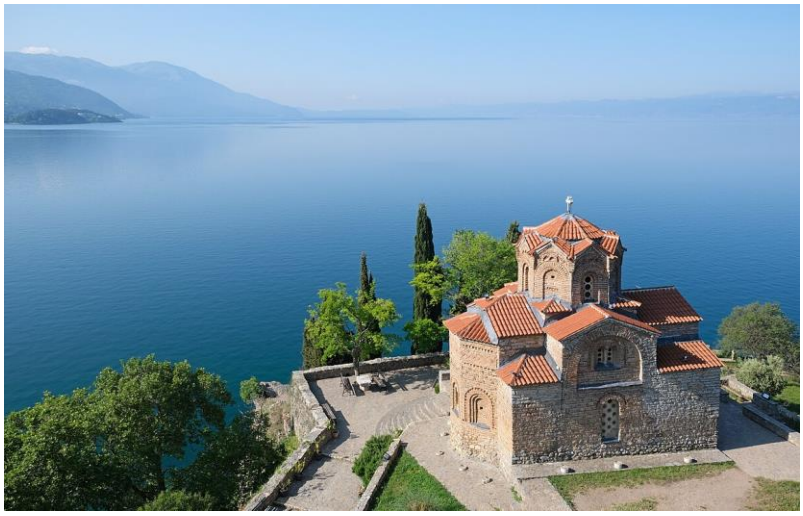

Figure 9: Full Resolution Image – Color.

Note: Zoom in to see obvious difference.

# 3 Section B - Histograms and Point Operators

## *Task 1*

Grayscale images with uniform intensity values have pixel intensity distributions where each intensity level occurs with approximately equal frequency. This means the histogram will be relatively flat of my chosen image. I used OpenCV to load a grayscale image and calculated its

histogram and cumulative histogram using cv2.calcHist() and numpy's cumsum() function, respectively. To invert the image, I subtracted each pixel intensity value from 255 (since the maximum intensity value for an 8-bit image is 255). See figure below.

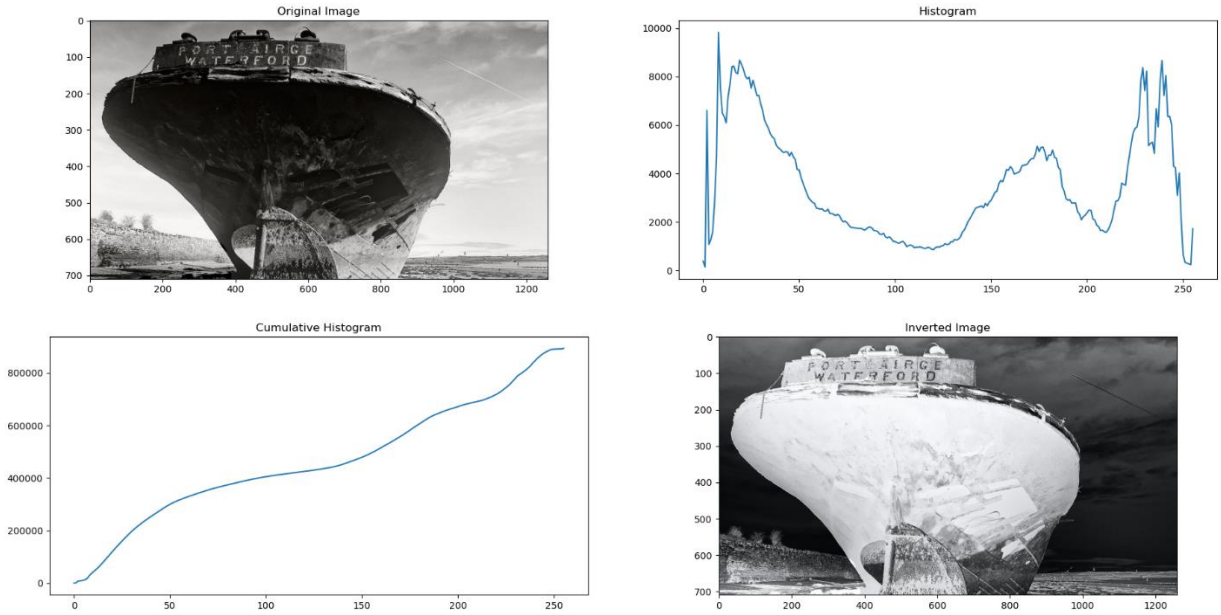


Figure 10: Natural Greyscale Image, Histogram of number of pixels and their intensities, Cumulative Histogram of number of pixels and their intensities, Inverted Greyscale Image.

Note: Zoom in to see graphs clearly.

## *Task 2*

Dark grayscale images have lower pixel intensity values concentrated towards the darker end of the intensity scale, while light grayscale images have higher pixel intensity values concentrated towards the lighter end. After a profound search on Google Images I was able to find very good examples of these properties.

I loaded dark and light grayscale images using OpenCV and calculated the histogram and cumulative histogram for each using cv2.calcHist() and numpy's cumsum() function, respectively. See figure below.

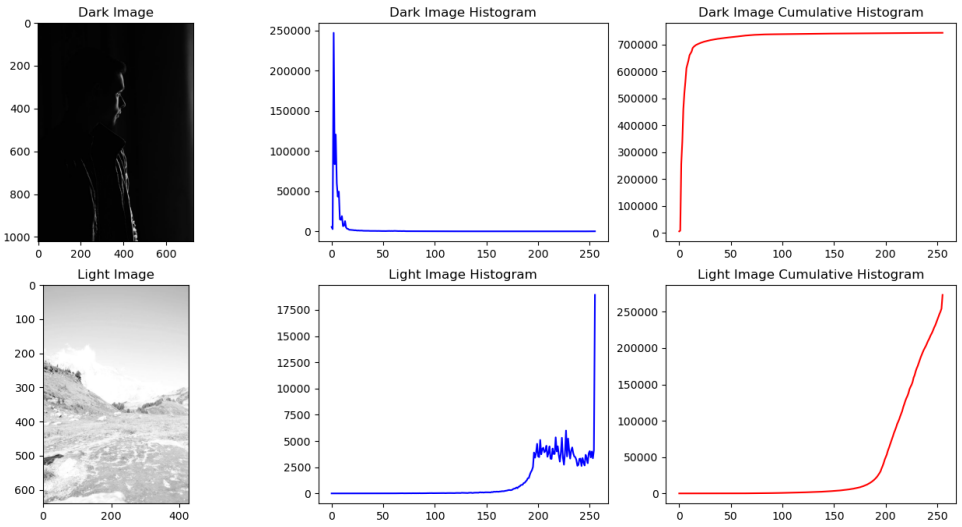


Figure 11: Light and Dark image, Histogram of number of pixels and their intensities for both images, Cumulative Histogram of number of pixels and their intensities for both images.

Note: Zoom in to see graphs clearly.

Indeed, the images I have supplied are very representative of the nature of dark and light images. The histogram confirms this. The cumulative histogram is also easy to interpret. As we can see the dark image cumulative histogram spikes up very quickly due to the number of low intensity pixels, and the opposite is true for the light image cumulative histogram.

## *Task 3*

Histogram equalization is a method used to enhance the contrast of an image by redistributing pixel intensities. It tends to spread out the pixel intensity values, which can be beneficial for images with uneven or skewed histograms.

I applied histogram equalization to both the dark and light images using cv2.equalizeHist(). Then, I calculated the histograms and cumulative histograms for the equalized images. I also printed out the resulting images which do look a lot better and more human. See below.

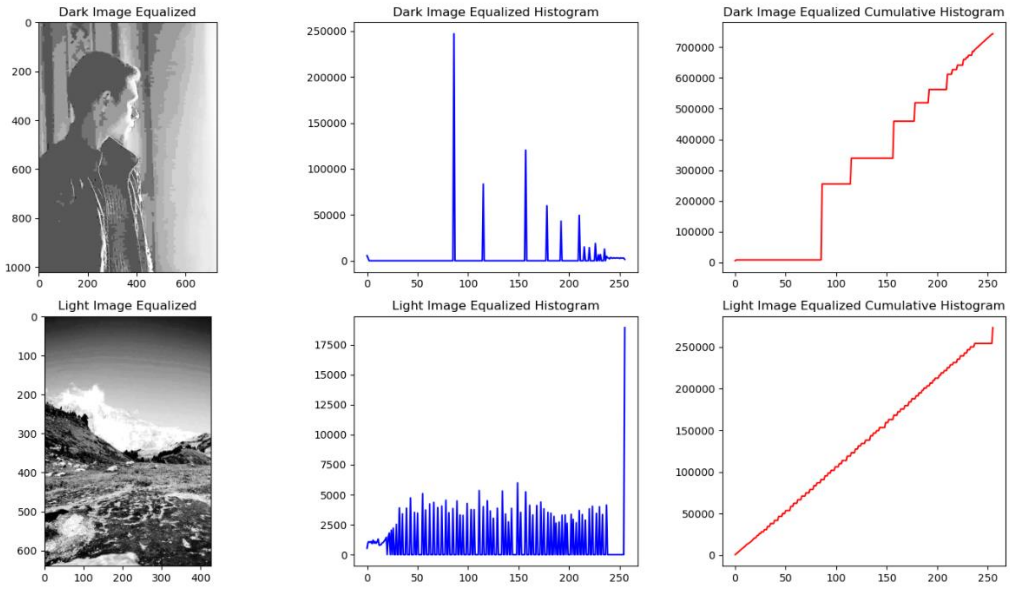


Figure 12: Light and Dark EQ image, Histogram of number of pixels and their intensities for both EQ images, Cumulative Histogram of number of pixels and their intensities for both EQ images.

Note: Zoom in to see graphs clearly.

## *Task 4*

Pictures that have a small pixel range may look flat or lack contrast. The overall look and visual quality of such photos can be improved by increasing the brightness and contrast. Using OpenCV, I imported the original picture and applied two distinct transformations: a brightening and a contrast-boosting one. I used the cv2.convertScaleAbs() method with a larger value for the beta parameter in order to boost brightness. I used the same function, increasing the value of the alpha parameter while maintaining the value of zero for the beta parameter, to boost contrast. See below.

Figure 13: Dull Greyscale image of the Sea.

Figure 14: Increased Brightness image, frame added for clarity of image borders.

Figure 15: Increased Contrast image, frame added for clarity of image borders.

# 4 Section C - Filters

## *Task 1*

Smoothing filters are used to reduce noise and blur images by averaging the pixel values in a neighborhood. Common smoothing filters include the box filter, Gaussian filter, and median filter.

I'm going to smooth the grayscale image using three distinct 3x3 filters: a box filter, a Gaussian filter, and a median filter. Functions to apply these filters are available in OpenCV.

I'll utilize the borderType option to control how borders are handled during filtering in order to manage picture borders.

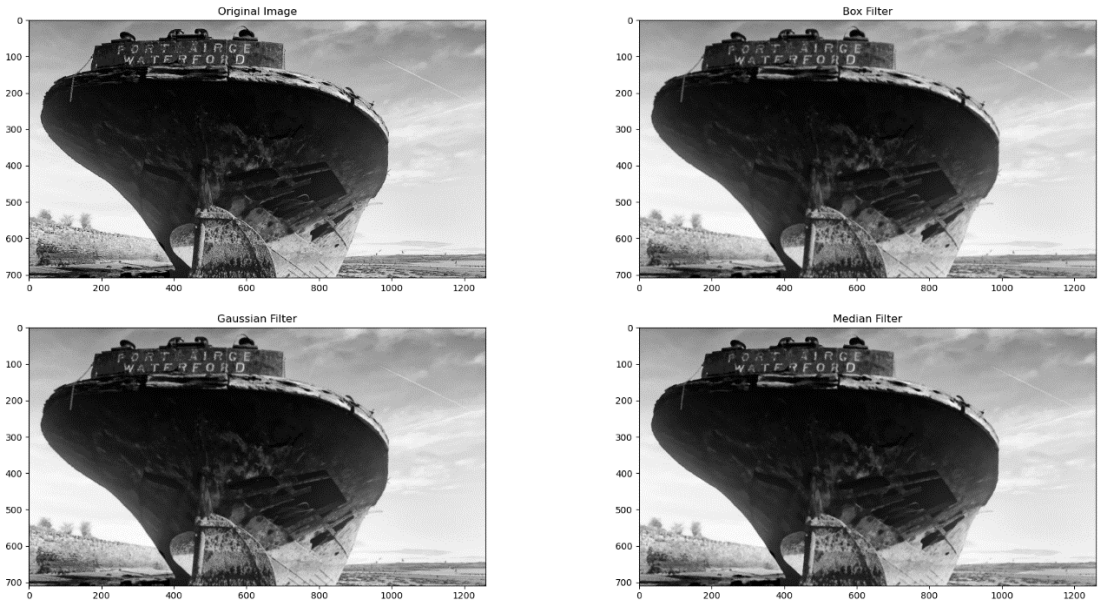


Figure 16: Original image and different smoothing methods applied to it. Note: Zoom in.

## *Task 2*

By adding noise to photos, one may replicate real-world situations in which images could be distorted during transmission or capture. Salt-and-pepper noise can be effectively eliminated with median filtering.

I'll start by randomly changing the intensity of a few hundred pixels (and eventually, 2x2 clusters of pixels) to zero in order to introduce noise to the grayscale image. To fix the noisy image, I'll next apply 3x3 and 5x5 median filters. A median filtering function called medianBlur is included in OpenCV.

We can easily split the picture into non-overlapping 2-by-2 clusters for the second section. Pick a few hundred of these clusters at random. To add noise, set all of the pixel intensities in the chosen clusters to 0. Utilize 3 x 3 and 5 x 5 median filters to fix the image that is noisy. See images below.

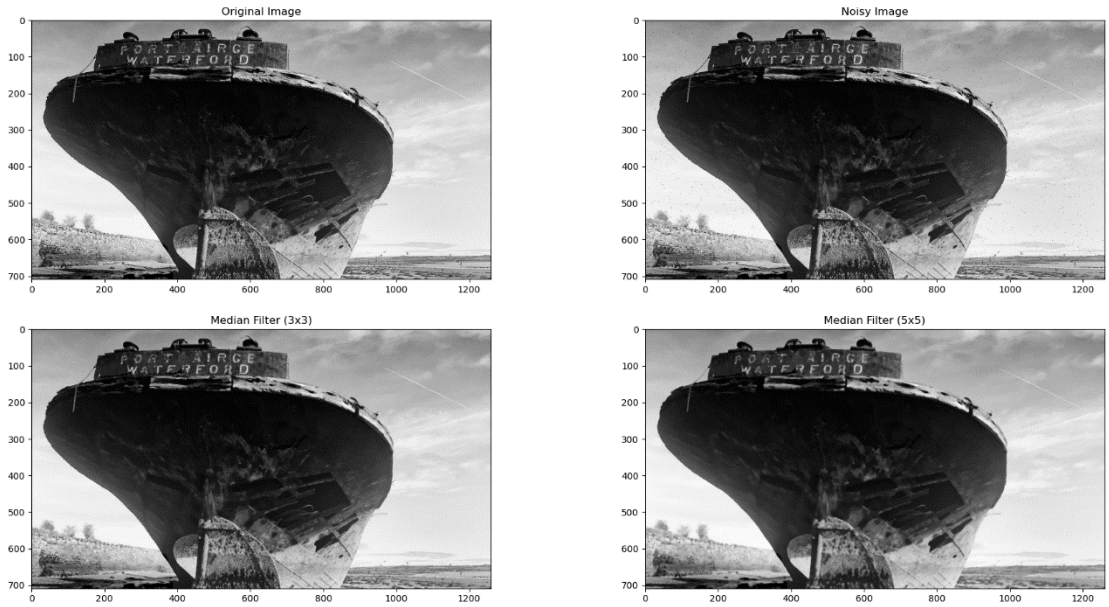


Figure 17: Noisy pixel image and fixed images. Note: Zoom in.

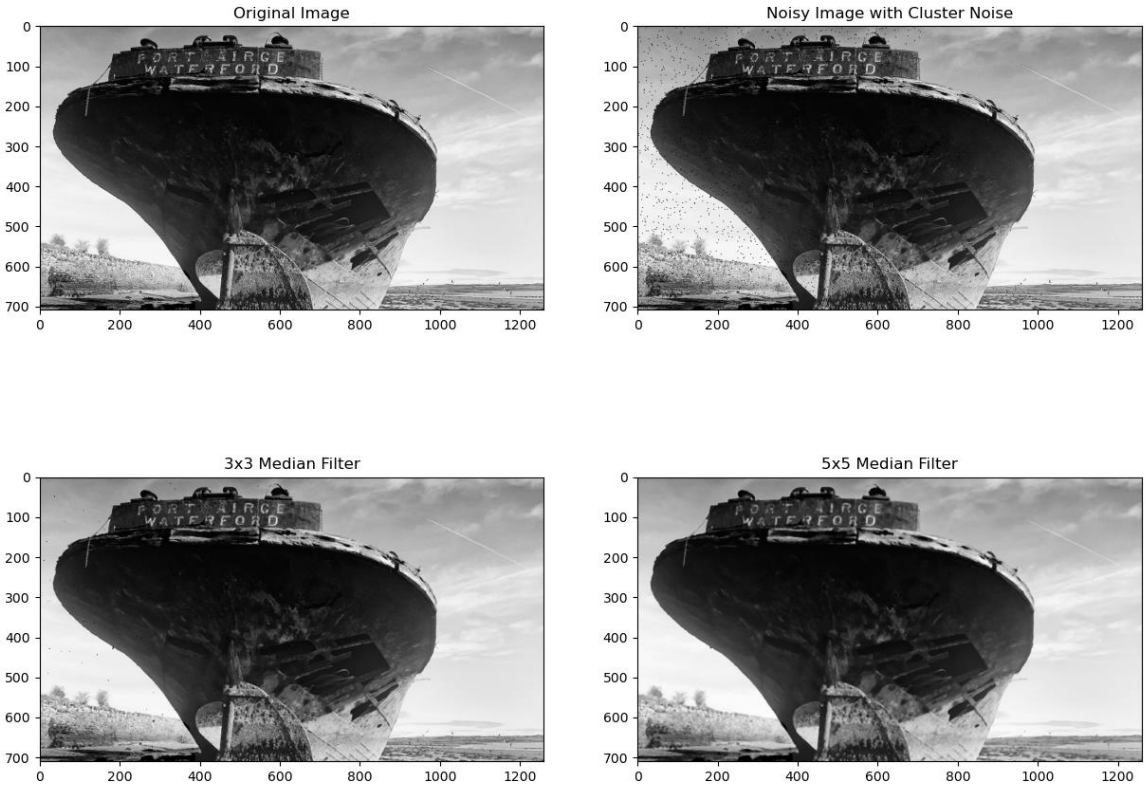


Figure 18: Noisy 2x2 cluster image and fixed images. Note: Zoom in.

## *Task 3*

Unsharp masking is a sharpening method that involves removing a blurry portion of an image from the original to improve the edges. I am going to use OpenCV to do unsharp masking on the grayscale image. Image smoothing is the first step in the process, after which the smoothed version is subtracted from the original and added back. In doing so, noise is decreased and edges are improved.

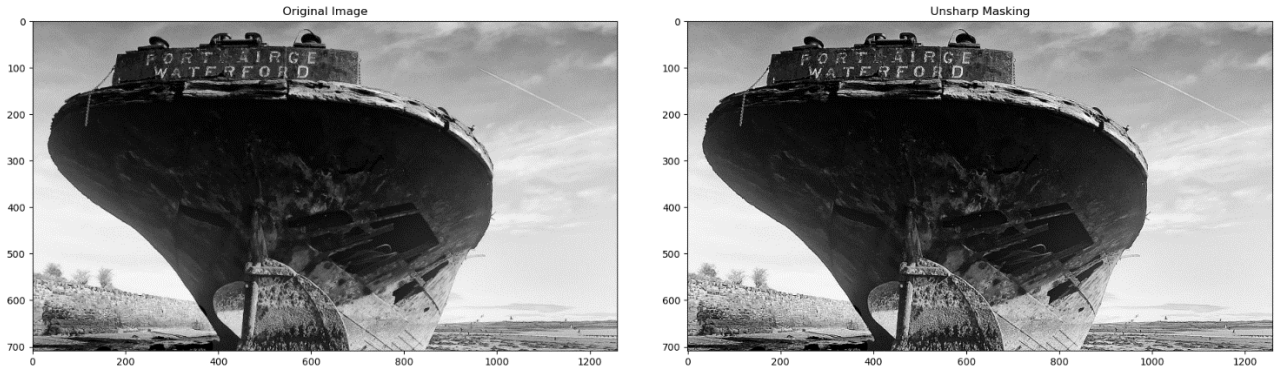


Figure 19: Original image and unsharp mask image.

Note: Zoom in.